XX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais

13-16 de Outubro de 2020

# MINICURSOS

## XX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais

Dados Internacionais de Catalogação na Publicação (CIP)

S612 Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (20. : 2020)

Minicursos do XX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais [recurso eletrônico] – organizadores: Fábio Borges, Raphael Carlos Santos Machado – Porto Alegre : SBC, 2022.

ISBN 978-65-87003-85-6

1. Segurança da informação. 2. Sistemas computacionais. I. Borges, Fábio. II. Machado, Raphael Carlos Santos. III. Título.

CDU 004

Catalogação elaborada por Francine Conde Cabral
CRB-10/2606

# MINICURSOS
# XX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais

**Editora**
Sociedade Brasileira de Computação – SBC

**Organizadores**
Fábio Borges
Raphael Carlos Santos Machado

**Realização**
Sociedade Brasileira de Computacão – SBC
Laboratório Nacional de Computação Científica – LNCC

**Promoção**
Sociedade Brasileira de Computação

# Comitês

## Coordenação CESeg
Altair Santin – PUCPR
Michele Nogueira – UFPR

## Coordenação Geral
Fábio Borges de Oliveira – LNCC
Raphael Carlos Santos Machado – Inmetro

## Comitê Local
Yuri Gonçalves Miguel – LNCC
Aline de Lurdes Zuliani Lunkes – LNCC
Thays Rocha Neri Ferreira – LNCC

## Coordenação de Palestras
Renato Portugal – LNCC
Marcos Simplicio – USP

## Coordenação do Comitê de Programa

- Trilha de Sistemas Computacionais e Redes de Comunicação
  Igor Monteiro Moraes – UFF

- Trilha de Criptografia
  Luis Kowada – UFF

## Coordenação do CTF
Pedro Carlos Lara – Cefet–RJ
Michelle Wangham – UNIVALI

## Coordenação de Minicursos
Eduardo Viegas – PUCPR

## Coordenação do Workshop de Trabalhos de Iniciação Científica e de Graduação (WTICG)
Natalia de Castro Fernandes – UFF
Márjory Da Costa-Abreu - Sheffield Hallam University/UFRN

## Coordenação do Concurso de Teses e Dissertações em Segurança da Informação e Sistemas Computacionais (CTDSeg)
Eduardo Feitosa – UFAM

## Coordenação do Salão de Ferramentas
Roberto Araújo Samarone – UFPA

## Coordenação do Workshop de Gestão de Identidades Digitais (WGID)
André Marins – RNP
Emerson Ribeiro de Mello - IFSC

## Coordenação do Workshop de Forense Computacional (WFC)
Robson Albuquerque – ABIN
João Gondim – UnB

## Coordenação do Fórum de Segurança Corporativa (FSC)
Davidson Boccardo – Clavis Segurança da Informação
Flavia Bernardini – UFF

# Sumário

# Capítulo 1

# Processamento confidencial de dados de sensores na nuvem

Andrey Brito (UFCG), Clenimar Souza (UFCG), Fábio Silva (Scontain), Lucas Cavalcante (UFCG), Matteus Silva (UFCG)

***Abstract***

*This tutorial presents how to use confidential computing tools for developing IoT applications that process potentially sensitive data in the cloud. For confidential processing, our tutorial uses Intel SGX through both Intel SGX SDK (for developing applications and services from scratch) and a runtime environment, SCONE (ideal for existing applications). For the dissemination of IoT data, we use Apache Kafka, and for orchestrating applications, we use Kubernetes. The concepts presented will be illustrated through a distributed application for power consumption monitoring.*

***Resumo***

*Este minicurso apresenta como ferramentas de computação confidencial podem ser usadas para o desenvolvimento de aplicações que processam dados potencialmente sensíveis de aplicações de Internet das Coisas na nuvem. Para o processamento confidencial, nosso minicurso usará tanto o SDK (Software Development Kit) para SGX da Intel (para novas aplicações e serviços) como uma plataforma de execução, SCONE (ideal para execução de aplicações existentes). Para disseminação de dados de Internet das Coisas, nós utilizaremos Apache Kafka e para a orquestração de aplicações, usaremos Kubernetes. Os conceitos apresentados aqui serão ilustrados através de uma aplicação distribuída de processamento de dados de consumo de energia elétrica.*

## 1.1. Introdução

Computação na nuvem e Internet das Coisas têm mudado o papel da computação na vida das pessoas. Por um lado, computação na nuvem mudou como empresas e empreendedores individuais abordam novos projetos. O longo processo de planejamento para investimentos em infraestrutura para hospedar novas aplicações foi substituído pelo modelo pague-por-uso da computação na nuvem, onde um custo mínimo de infraestrutura é necessário para publicar uma nova aplicação na Internet. Além disso, uma vez publicada, a aplicação pode ser expandida sob demanda, com a

nuvem fornecendo tantos recursos quanto necessário para o crescimento da capacidade de atendimento dos usuários.

Por outro lado, a Internet das Coisas trouxe aspectos avançados da computação para ainda mais perto do usuário final. Sensores conectados podem monitorar detalhadamente os hábitos das pessoas e, com a ajuda de técnicas de inteligência artificial, prover sugestões como rotas otimizadas para o caminho de casa no momento em que o carro é ligado (e antes mesmo que o usuário manifeste alguma intenção de usar o GPS) ou sugestões como o aumento do número de passos ao longo do dia ou uma maior obediência dos horários de dormir à noite.

Infelizmente, a Internet da Coisas não traz apenas benefícios para o público em geral. Com hábitos e ações monitorados constantemente, a privacidade e a segurança dos dados têm se tornado uma preocupação cada vez mais pertinente no dia-a-dia das pessoas. Assim, as aplicações inovadoras que usam cada vez mais dados para melhorar o nosso dia-a-dia têm que ser desenvolvidas com mais preocupação a respeito da segurança dos dados que processam. Esta necessidade faz com que mesmo pequenos empreendedores, que só conseguem fazer suas aplicações atingirem o público geral graças às facilidades de provisionamento de infraestrutura permitidas pela computação na nuvem, tenham que desenvolver aplicações que apresentem boas garantias de segurança de dados. Finalmente, e felizmente para o público geral, esta necessidade técnica se torna uma obrigação, dada a vigência das novas leis de proteção de dados no Brasil (LGPD, Lei Geral de Proteção de Dados) e na Europa (GDPR, *General Data Protection Regulation*, ou Regulamento Geral sobre a Proteção de Dados).

Do ponto de vista técnico, as tecnologias de computação na nuvem e segurança de dados têm evoluído para atender às necessidades de eficiência de custo e proteção. **Computação nativa da nuvem** (ou, do inglês, *Cloud Native Computing*) é uma área que foca na gerência eficiente de recursos na nuvem, minimizando, por exemplo, máquinas virtuais ociosas (usando grãos menores de escalonamento, contêineres) e problemas de escalabilidade ou disponibilidade (com monitoramento e orquestração automatizada). Já **computação confidencial** tem como objetivo minimizar a necessidade de confiança assumida nas infraestruturas e seus operadores, deslocando-a para software e hardware, que são menos passíveis de vazamentos acidentais.

Neste tutorial, combinamos estas duas tendências tecnológicas em uma área que chamamos de **computação confidencial nativa da nuvem** (CCNN) e validamos a aplicação deste conceito em uma aplicação de processamento de dados de sensores de forma confidencial na nuvem.

O restante do capítulo está organizado da seguinte forma: na Seção 1.2, apresentamos os principais conceitos de computação nativa da nuvem e duas ferramentas populares que serão utilizadas neste capítulo; na Seção 1.3 apresentamos computação confidencial, com foco na tecnologia Intel SGX, a mais popular atualmente; já na Seção 1.4 combinamos os dois conceitos para a construção de uma aplicação confidencial nativa da nuvem, detalhando seus funcionamento e implantação; finalmente, a Seção 1.5 apresenta boas práticas e caminhos promissores nesta área. Como complemento, o Apêndice A detalha como máquinas virtuais podem ser criadas em um provedor de nuvem público para a execução de aplicações confidenciais.

## 1.2. Computação nativa da nuvem

Embora muitos dos conceitos fundamentais estivessem sendo discutidos há bastante tempo, o termo Computação Nativa da Nuvem (do inglês, *Cloud Native Computing*) ganhou impulso com a fundação da ***Cloud Native Computing Foundation*** (CNCF) em 2015[1]. A CNCF é parte da Linux Foundation, uma organização sem fins lucrativos criada para disseminar o uso de Linux e a criação e adoção de projetos de código aberto. A CNCF, por sua vez, tem como objetivo promover o uso de contêineres e o alinhamento da indústria em torno do desenvolvimento desta tecnologia, que permite o empacotamento de aplicações de forma mais compacta e sua instanciação de forma mais ágil, melhorando a eficiência da operação das aplicações.

A CNCF define computação nativa da nuvem a partir das características de suas aplicações:

- As aplicações são criadas e executam de forma escalável em ambientes modernos e dinâmicos, como nuvens públicas, privadas e híbridas;
- As aplicações usam tecnologias como contêineres, redes de serviços (*service meshes*), microsserviços, infraestrutura imutável, e APIs declarativas;
- Os componentes da aplicação são desacoplados e sua implantação é suportada por altos níveis de automação, que permitem mudanças frequentes com impactos previsíveis e limitados.

O foco em nuvem ajuda a priorizar a redução de custos de operação e o uso de infraestruturas genéricas. Na mesma direção, o conceito de **infraestrutura imutável** valoriza a automação e repetibilidade. A ideia é que uma vez que servidores não podem ser ajustados depois de colocados em operação, elimina-se a ideia de que servidores precisam de cuidado individual, o que tipicamente é uma atividade manual e custosa. Assim, diminui-se o tempo gasto em operações de manutenção e aumenta-se a utilização de mecanismos automatizados para retirada e inserção de servidores.

Em cima dessa infraestrutura genérica e imutável vêm as tecnologias para gerência das aplicações. O uso de tecnologias como contêineres, gerenciados por plataformas de redes de serviços, e que hospedam aplicações de responsabilidade limitada na forma de microsserviços permitem a agilidade de ativação e desativação das instâncias de um componente, mantendo a infraestrutura ajustada à real necessidade momentânea, o que garante tanto escalabilidade como eficiência. Este conjunto de características é então complementado com o uso de APIs declarativas, que priorizam a especificação do que deve ser feito e não a forma que deve o objetivo deve ser atingido, em contraste com a ideia de programação e especificação imperativa, onde o processo é descrito e não o objetivo. Uma vez que o sistema tem informações do estado esperado, o monitoramento é simplificado, aumentando o suporte do sistema a operações de auto-recuperação (do inglês, *self-healing*).

### 1.2.1. Microsserviços

Um microsserviço é um componente da arquitetura de uma aplicação que tem uma responsabilidade clara e limitada. Microsserviços idealmente seguem estilos de

[1] https://www.cncf.io/

interface usando padrões de troca de dados abertos e populares como **JSON** (*JavaScript Object Notation*). Estas mensagens JSON são então transportadas por sistemas de **comunicação direta ou indireta**.

Um sistema de comunicação é direto quando existe uma ligação entre o remetente e os destinatários. Por outro lado, a comunicação é indireta quando existe um intermediário que desacopla o remetente dos destinatários. A ausência de intermediários permite maior eficiência e desempenho na comunicação direta, no entanto, implica em maior acoplamento. Já a comunicação indireta permite um desacoplamento de espaço e de tempo entre os remetentes e destinatários. O desacoplamento de espaço significa que o remetente não precisa conhecer o destinatário, conhecendo apenas o intermediário, o sistema de comunicação. Este desacoplamento facilita a replicação de mensagens e a substituição gradual de componentes (ex., ter duas versões de um microsserviço operando em paralelo por um tempo). O desacoplamento no tempo retira a limitação que o remetente e o destinatário estejam ativos ao mesmo tempo. As mensagens podem então ser armazenadas temporariamente no sistema de comunicação e entregues posteriormente aos destinatários. Esta abordagem ajuda a lidar com problemas de conectividade (como em Internet das Coisas e computação na borda) e com a recuperação após falhas, já que mensagens podem ser mantidas temporariamente.

Para comunicação direta, o estilo REST favorece a padronização e, portanto, a interoperabilidade. Isso é consequência do fato que o uso de um protocolo bem definido (HTTP, *HyperText Transfer Protocol*), com operações uniformes (POST, GET, PUT, DELETE), reusa o conhecimento dos desenvolvedores e são melhor suportados pelas ferramentas de desenvolvimento.

Para comunicação indireta, os dois estilos dominantes são filas de mensagens e barramentos de mensagens. O estilo de **fila mensagens** implementa uma fila (seja ela centralizada ou distribuída) que armazena as mensagens temporariamente, desacoplando remetentes e destinatários. Este estilo é mais popular quando a comunicação é de um para um. O estilo de **barramento de mensagens** é mais popular quando a comunicação é de um para muitos. Neste caso, um componente pode passar a mensagem para o sistema de comunicação e vários outros podem estar conectados no mesmo barramento, estando aptos a receber a mensagem. A lógica de roteamento pode variar, sendo o padrão publicar-assinar baseado em tópicos (do inglês, *topic-based publish-subscribe* [Euster et al. 2003]) o mais popular. O padrão publicar-assinar será o estilo usado no restante deste trabalho e será descrito em mais detalhes na próxima seção. É importante notar que mesmo quando a comunicação é um para um na maior parte do tempo, ter a possibilidade de conectar dinamicamente outros componentes e replicar as mensagens é útil para fins de teste (ex., alimentando uma nova versão de um componente), registro (ex., adição de um componente de *log* que armazena mensagens) ou tolerância a falhas (ex., duplicação das responsabilidades). Além disso, as ferramentas mais populares para barramentos de mensagens permitem o seu uso flexível como fila ou barramento.

Seja sobre um mecanismo de comunicação por fila, sobre um barramento ou sobre REST, JSON é um formato de troca bastante popular. JSON é um formato compacto, de padrão aberto independente, e que permite a troca de dados de forma simples entre os componentes. Além de também usufruir dos benefícios da

popularidade, como o conhecimento prévio dos desenvolvedores e a ampla disponibilidade de bibliotecas para sua manipulação, o padrão JSON é legível para humanos através de sua notação em texto plano e estrutura de pares chave-valor.

A popularidade de ferramentas para a implementação das abordagens usadas para comunicação entre microsserviços (REST e JSON) tem outra consequência desejável conhecida como programação poliglota. Uma vez que a comunicação entre os serviços é explícita e as bibliotecas para implementação dos canais (REST, filas, ou barramentos) e processamento das mensagens (JSON) estão disponíveis e bem documentadas nas linguagens populares, existe uma grande liberdade nas escolhas para a implementação de cada microsserviço. Assim, diferentes desenvolvedores podem implementar seus microsserviços usando as linguagens de programação mais familiares e as bibliotecas mais adequadas à realização das responsabilidades daquele microsserviço. Por este mesmo motivo, inovar é simples, já que uma vez que alguma ferramenta se torna disponível para realizar uma tarefa bem a reescrita daquele microsserviço na linguagem de programação recomendada para aquela nova biblioteca terá impacto limitado no resto da aplicação.

### 1.2.2. Barramentos de mensagens e Apache Kafka

Também conhecidos como sistemas baseados em eventos distribuídos, os sistemas publicar-assinar são caracterizados pelo baixo acoplamento das interações entre as diferentes entidades participantes, sendo este um requisito de sistemas de grande escala [Eugster et al., 2003]. Esse tipo de sistema é composto por publicadores, assinantes, e um serviço de eventos, também conhecido como barramento de mensagens ou, do inglês, *broker*.

Os publicadores divulgam eventos estruturados para o *broker*, e assinantes expressam interesse em eventos ou padrões de eventos específicos por meio de assinaturas. Em outras palavras, publicadores publicam informações para o *broker*, e através de uma etapa de filtragem baseado nas assinaturas recebidas, é feito o roteamento e entrega dessas mensagens para os respectivos assinantes. Esse paradigma para comunicação é bastante aplicado onde há disseminação e processamento de informações em larga escala e de forma contínua, como sistemas financeiros, sistemas de monitoramento de recursos, aplicações de IoT, entre outros.

O desacoplamento provido pelos serviços de eventos em conjunto com os publicadores e assinantes ocorre em três dimensões: espaço, tempo e sincronização [Eugster et al., 2003]. O desacoplamento espacial está relacionado com a ausência de comunicação direta entre os publicadores e assinantes. Mais especificamente, os publicadores sequer conhecem os endereços dos assinantes. O desacoplamento temporal está relacionado com a falta de necessidade das partes estarem interagindo ao mesmo tempo, uma vez que publicações podem ser feitas, e podem ser consumidas eventualmente. E, por fim, o desacoplamento de sincronização ocorre pois os consumidores são notificados de que existem novos eventos de maneira assíncrona, enquanto estão realizando atividades concorrentes.

Existem diferentes esquemas para assinaturas em sistemas publicar-assinar, onde os mais populares são o baseado em tópico (do inglês, *topic-based*) e o baseado em

conteúdo (do inglês, *content-based*) [Eugster et al., 2003]. Aqui usaremos o esquema baseado em tópico, onde os eventos publicados são associados a um tópico específico por seus respectivos publicadores, e os assinantes, por sua vez, assinam a tópicos. Na medida em que eventos vão sendo publicados, serão entregues para os assinantes dos tópicos ao qual estes pertencem. Existem diferentes implementações de serviços de eventos que suportam o esquema baseado em tópico, neste tutorial utilizaremos o Apache Kafka[2].

O Apache Kafka, aqui chamado simplesmente de Kafka, foi desenvolvido pelo LinkedIn[3] e posteriormente doado em 2011 para a Apache Software Foundation. O Kafka é um sistema publicar-assinar de código aberto bastante popular e escalável, usado em inúmeras empresas e sistemas de grande porte[4]. A sua implantação é flexível, permitindo que seja usado de forma portável em aplicações de pequeno porte ou em *clusters* para implantações escaláveis e tolerantes a falhas. Quando usado em modo de *cluster*, cada nó é tipicamente chamado de ***broker***. Além de funcionar como barramento de mensagens, integrandos produtores e consumidores assíncronos, o Kafka se tornou popular também como hub de mensagens em um sistema baseado em microsserviços.

No Apache Kafka, o **tópico** é a entidade para a qual as mensagens são publicadas e consumidas. Um tópico é dividido em **partições**. As partições dividem os dados de um tópico para permitir a escrita de vários produtores e leituras de vários consumidores no mesmo. Toda mensagem em uma partição possui um identificador chamado ***offset***, que é sequencial, monotonicamente crescente e denomina a posição de uma mensagem numa partição. Consequentemente, toda mensagem em um sistema Kafka pode ser identificada únicamente através da 3-upla *(Tópico, Partição, Offset)*.

Um tópico possui um fator de replicação, que determina a quantidade de réplicas, que será distribuída pelos *brokers* do *cluster*. Um *broker* pode ser eleito o líder de uma determinada partição, o que implica que todas as leituras e escritas em uma partição devem ser feitas exclusivamente a partir dele. É o líder que também coordena a atualização das réplicas com novas mensagens, e se um líder falha, uma réplica do *broker* se torna o novo líder. Uma vez que diferentes publicações para diferentes partições são gerenciadas por diferentes *brokers*, o sistema atinge escalabilidade.

Consumidores para um dado tópico podem ser independentes ou se organizarem na forma de grupos de consumidores (chamados *ConsumerGroups*). Quando independentes, todos recebem as mesmas mensagens. Quando em grupo, cada consumidor dentro do grupo lê de uma única partição, de maneira que o grupo como um todo consome todas as mensagens de um tópico.

É delegado a um dos *brokers* do *cluster* Kafka o papel chamado coordenador de grupo. Isso é feito para cada novo grupo de consumidores que se inscreve no sistema. Dentre suas responsabilidades, o coordenador de grupo detecta através do uso de pacotes de controle, se todos os consumidores de um grupo estão em funcionamento. Caso algum venha a parar de funcionar, ele é responsável por engatilhar o rebalanceamento através do líder do grupo.

[2] https://kafka.apache.org/
[3] https://www.linkedin.com
[4] https://kafka.apache.org/powered-by

É considerado líder do grupo o primeiro consumidor a juntar-se ao grupo. O líder recebe do coordenador de grupo uma lista de todos os consumidores do grupo, e é responsável por atribuir um subconjunto de partições a cada consumidor. Se existem mais consumidores do que partições, alguns deles ficarão em espera, mas se existem mais partições do que consumidores, alguns consumidores irão ler mensagens de mais de uma partição. Felizmente, a atuação do líder do grupo é feita de forma transparente, de acordo com a implementação do cliente ou biblioteca Kafka utilizada pela aplicação.

A Figura 1.1 apresenta algumas configurações de grupos de consumidores com diferentes balanceamentos. Nesta figura temos um tópico *xpto*, que possui replicação 2 e quatro partições. O nó de *id* 0 lidera as partições *P1* e *P4*, enquanto o nó de id 1 lidera as partições *P2* e *P3*. Como discutido acima, cada líder gerencia a entrega de mensagens aos seus consumidores.

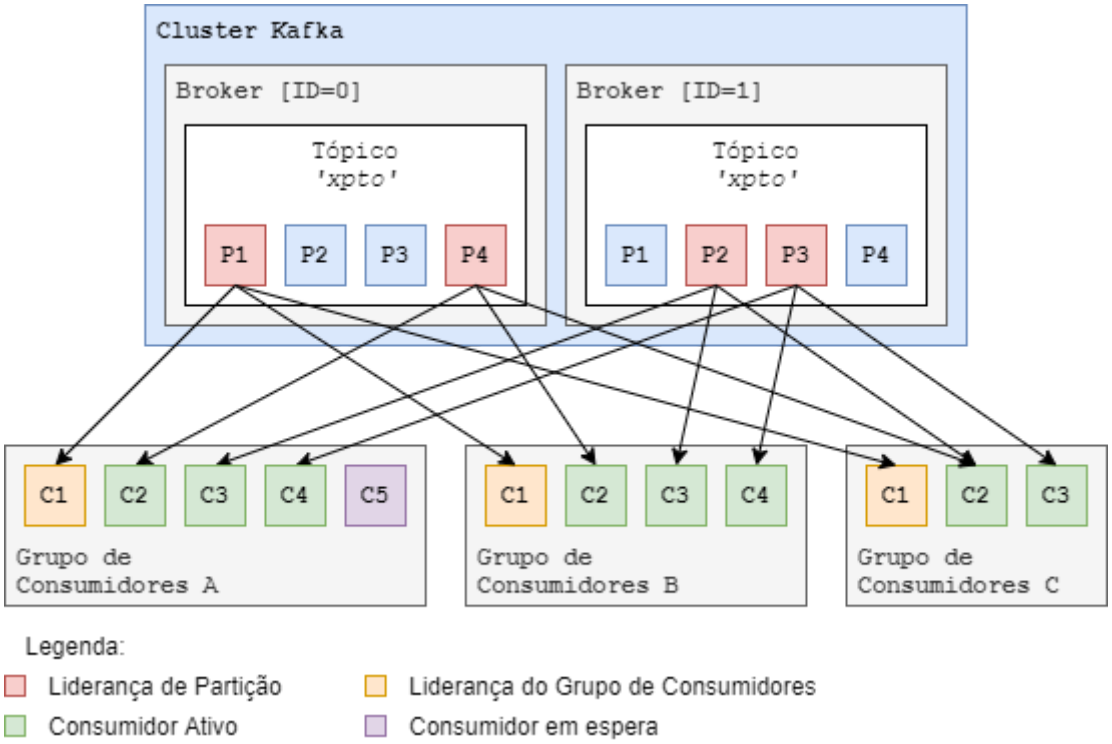



**Figura 1.1. Balanceamento de grupos de consumidores**

Na Seção 1.2.4 faremos uma implantação do Apache Kafka para nossa aplicação caso de uso. Embora o Apache Kafka tenha mecanismos de proteção para dados em trânsito e mecanismos para autenticação, ele não dispõe de mecanismos para proteção para dados em repouso e em processamento. Como discutido na Seção 1.4.3, estes níveis adicionais de proteção são importantes para o nosso modelo de ameaças, que considera que dados sensíveis podem estar em infraestruturas compartilhadas e gerenciadas por terceiros. Finalmente, a Seção 1.6 apresenta discussões sobre outras boas práticas e otimizações para o caso de uso apresentado, incluindo aspectos relacionados ao Kafka.

### 1.2.3. Kubernetes

A popularização de microsserviços e a crescente adoção de contêineres como principal forma de implantação e distribuição de aplicações nativas da nuvem impulsionaram o surgimento de aplicações para gerenciar grandes conjuntos de contêineres em produção. Estas aplicações, conhecidas como orquestradores de contêineres, atuam no topo da infraestrutura (seja esta composta por máquinas físicas ou máquinas virtuais), por vezes abstraindo-a complemente, o que diminui acoplamento e, por conseguinte, provê mais portabilidade para as aplicações.

Kubernetes é, atualmente, o orquestrador de contêineres padrão da indústria. Lançado oficialmente em 2014, o projeto foi desenvolvido pela Google e teve sua arquitetura fortemente influenciada pelo orquestrador de contêineres interno da empresa, Borg [Verma et al., 2015]. Após o lançamento, Google e Linux Foundation fundaram a CNCF, que passou a gerenciar o projeto Kubernetes, além de formular e disseminar as diretrizes do que seria a computação nativa da nuvem.

Um *cluster* Kubernetes é composto por dois tipos de nós: *master* e *worker*. Nós *master* são responsáveis por executar os componentes do painel de controle de Kubernetes, como o servidor de API (*api-server*) e o gerenciador de controladores (*controller manager*). É recomendado que nós *master* sejam usados exclusivamente para esse fim, e não executem aplicações de usuários. Nós *worker*, por sua vez, são dedicados a executar aplicações de usuários. É comum que *clusters* tenham apenas um *master*. Contudo, para *cluster* com requisitos de alta disponibilidade e tolerância a falha, é recomendável que se use ao menos três nós *master*. A persistência de todos os objetos e de todo o estado do *cluster* é efetuada no sistema de armazenamento distribuído chave-valor, *etcd*[5].

Um dos pontos-chave da arquitetura de Kubernetes é sua abordagem centrada em aplicação. Suas primitivas (*pods*, *deployments*, *daemonsets*, *services*, etc., discutidas abaixo) e poderosa API declarativa permitem que o usuário não precise se preocupar com detalhes da infraestrutura ou de escalonamento e alocação de recursos. Essa característica possibilita que Kubernetes gerencie infraestruturas muito grandes de forma eficiente e transparente pro usuário. De fato, atualmente, o orquestrador suporta oficialmente *clusters* de até 5000 nós[6]. Uma outra consequência dessa arquitetura é que a API de Kubernetes, por ser independente da infraestrutura, permite a portabilidade e interoperabilidade de aplicações em infraestruturas distintas (por exemplo, de provedores de nuvem diferentes).

Um conjunto de controladores assegura que a aplicação definida pelo usuário esteja sempre no seu estado desejado. Dessa forma, é possível delegar para o orquestrador, através de sua API, tarefas como gerenciamento de falhas, replicação, atualização e escalabilidade automática de aplicações. Além disso, ao longo dos anos, a API de Kubernetes fora estendida e alavancada por diversas outras aplicações, o que criou um ecossistema de computação nativa da nuvem extremamente rico e versátil. Uma infinidade de ferramentas de monitoramento, registro, controle de acesso, armazenamento, dentre outras, integra-se de forma rápida e fácil ao orquestrador,

[5] https://etcd.io/
[6] https://kubernetes.io/docs/setup/best-practices/cluster-large/

estendendo seu potencial e tornando a operação e provisionamento de infraestruturas e aplicações mais prático e robusto[7].

Das primitivas de Kubernetes, destacam-se *Pod*, *Deployment, Service*, *Ingress*, *DaemonSet* e *StatefulSet*, a seguir explicadas em maior detalhe.

***Pod.*** O menor grão de uma aplicação, representam um ou mais contêineres que podem ser enxergados como um conjunto coeso, servindo a um propósito em comum. Por exemplo, um contêiner de banco de dados e um contêiner auxiliar que funciona como um *proxy* desse mesmo banco de dados. *Pods* têm seu próprio espaço de rede (um endereço IP e portas) e sistema de arquivos, que são compartilhados por todos os seus contêineres. *Pods* também são considerados efêmeros pelo escalonador de Kubernetes, que pode terminá-los e movê-los por alguma razão (liberar recursos em um nó sob pressão, por exemplo). Por esse motivo, *Pods* devem sempre ser criados com um controlador de replicação associado, o que é oferecido por outras primitivas, como *Deployments* e *StatefulSets*.

***Deployment***. Representa um *Pod* associado a um controlador de replicação, que garante o estado desejado definido. É possível definir número de réplicas desejadas e políticas de recuperação de falhas. Por exemplo, caso um nó da infraestrutura fique indisponível em razão de uma falha, deixando um *Deployment* com menos réplicas que o desejado, o controlador rapidamente criará tantas réplicas quantas forem necessárias para que o número desejado de réplicas ativas seja restabelecido.

***Service***. Permite que *Pods* sejam descobertos e acessados por outras entidades. Como *Pods* são efêmeros, seus endereços IP podem variar constantemente. Um *Service* é, na prática, um balanceador de carga de camada 4 (de acordo com o modelo OSI) para *Pods*. Através de seletores e etiquetas é possível selecionar quais *Pods* são expostos por determinado *Service*. *Services* têm um endereço IP interno fixo e uma entrada no servidor de nomes do *cluster* (DNS) que podem ser acessados por outras entidades. Para que *Services* sejam acessíveis de fora do *cluster*, existem dois tipos especiais de *Service*: *NodePort*, que expõe uma porta na rede dos nós, permitindo que o *Service* seja alcançável através do endereço IP de qualquer nó na porta escolhida, e *LoadBalancer*, que aciona o provedor da infraestrutura na criação de um balanceador de carga dinâmico com um endereço IP público.

***Ingress.*** Atua como um balanceador de carga de camada 7 (de acordo com o modelo OSI) para *Services*, o que possibilita um conjunto maior de funcionalidades, já que um *Ingress* atua na camada de aplicação (suportando, por exemplo, endereços HTTP, como *www.example.com/app*), enquanto *Services* expõem apenas um endereço IP e uma porta. Através de um *Ingress* é possível particionar tráfego de forma mais complexa, levando em consideração nomes e *hosts* virtuais ou implantações canário, além de possibilitar o término de conexões seguras (SSL). Kubernetes não implementa a lógica de *Ingress*. Em vez disso, a API de *Ingress* precisa ser implementada por um controlador especial, chamado controlador de *Ingress*, que é responsável por implantar os balanceadores de carga e gerenciar objetos do tipo *Ingress* no *cluster*. Os

[7] https://landscape.cncf.io/

balanceadores de carga mais populares do mercado, como NGINX, HAProxy e Envoy, possuem seus próprios controladores de *Ingress* disponíveis.

***DaemonSet.*** Representa um *Pod* associado a um controlador de replicação que garante a existência de exatamente uma réplica da aplicação em cada nó do *cluster*, sendo possível filtrar nós através de seletores e etiquetas. Útil para implantar aplicações como coletores de registro, agentes de armazenamento, de monitoramento ou de atestação, entre outros.

***StatefulSet.*** Representa um *Pod* associado a um controlador de *StatefulSet,* que persiste a identidade de cada réplica. Dessa forma, é possível implantar aplicações que dependem de um estado, como um gerenciador de banco de dados. *Pods*, apesar de efêmeros, passam a ter uma identidade única e persistente que, combinada com estratégias de persistência de dados (provisionamento de volumes, por exemplo), permite a manutenção de estado mesmo em caso de falha.

Objetos da API de Kubernetes são declarados através de manifestos na linguagem YAML (acrônimo do inglês *Yet Another Markup Language*), como mostrado na Figura 1.2, que cria um *Deployment* e um *Service* para a aplicação NGINX. Os manifestos definem a versão da API do Kubernetes que irá processar a requisição (*apiVersion*), metadados (*spec.selector.matchLabels*, por exemplo, define uma etiqueta que permite que o *Service* encontre os *Pods* a serem expostos). No *Deployment* é possível, ainda, ver o número de réplicas desejadas (3), a imagem do contêiner a ser utilizada (*nginx:1.14.2*, a ser obtida de repositórios cadastros, como *hub.docker.com*), e a porta na qual a aplicação estará acessível (80). O *Service*, por sua vez, escuta na porta 80 (*port: 80*) e balanceia o tráfego para a porta 80 (*targetPort*) dos contêineres selecionados (através das etiquetas em *spec.selector*). Assim, é possível acessar o serviço através do nome *my-nginx:80*.

```
01 | apiVersion: apps/v1
02 | kind: Deployment
03 | metadata:
04 |  name: my-nginx
05 | spec:
06 |  selector:
07 |   matchLabels:
08 |      run: my-nginx
09 |  replicas: 3
10 |  template:
11 |    metadata:
12 |      labels:
13 |        run: my-nginx
14 |    spec:
15 |      containers:
16 |      - name: my-nginx
17 |        image: nginx:1.14.2
18 |        ports:
19 |        - containerPort: 80
20 | ---
21 | apiVersion: v1
22 | kind: Service
23 | metadata:
24 |   name: my-nginx
25 |  labels:
26 |    run: my-nginx
27 | spec:
28 |  ports:
29 |  - port: 80
30 |    targetPort: 80
31 |    protocol: TCP
```

```
32 |  selector:
33 |    run: my-nginx
```

**Figura 1.2. Exemplo de manifesto Kubernetes para aplicação NGINX**

Atualmente, a maior parte dos provedores de nuvem já oferecem serviços de Kubernetes gerenciado, que permitem a criação de *clusters* de forma simples e rápida. Para instalação em infraestrutura própria, há vários instaladores certificados pela CNCF, que cobrem casos de uso e infraestruturas diversas (*baremetal*, *clusters* de borda, *clusters* locais para teste e desenvolvimento). Um dos mais versáteis instaladores, MicroK8s[8], é provido pela Canonical, e permite a instalação rápida em máquinas ou *clusters* locais através do gerenciador de pacotes do Ubuntu. Os parceiros certificados deste projeto incluem distribuições, instaladores e ambientes de hospedagem[9,10].

## 1.3. Caso de uso: processamento de sensores de consumo de energia

O nosso caso de uso é uma versão simplificada do sistema LiteCampus[11] [Silva, Silva e Brito, 2020]. O LiteCampus é um sistema de processamento de dados de sensores, especializado na gerência de consumo de energia elétrica. Este sistema permite aos usuários, por exemplo, acompanhar o histórico de consumo, estimar contas, identificar consumo por equipamento, além de receber alertas para anomalias que possam danificar equipamentos ou implicar em multas na conta de energia.

O LiteCampus usa o modelo publicar-assinar, onde os diferentes componentes interagem através do barramento de mensagens Kafka, conforme ilustrado na Figura 1.3. Através do Kafka são disseminados dados potencialmente confidenciais e, portanto, estes dados devem ser protegidos. Os principais componentes figura são os seguintes.

- ***Smart meters***: Também conhecidos como medidores de energia inteligentes, são dispositivos computacionais capazes de monitorar o consumo em uma rede elétrica e transmiti-lo através de um canal de comunicação. Neste caso, os medidores estão instalados nos quadros elétricos e transmitem as informações relativas ao seu consumo para o LiteKafka Gateway, através do uso de uma conexão segura HTTPS (HTTP sobre TLS).
- ***LiteKafka Gateway***: Se comunica com os medidores usando protocolos padrão ou próprios para recuperar e publicar as informações de consumo. Ele é necessário pois os medidores, por serem dispositivos limitados, tipicamente não suportam protocolos complexos. As informações coletadas são transformadas em mensagens Kafka e publicadas em um tópico no barramento de mensagens.
- ***Cluster* Kafka**: Conjunto de brokers Kafka configurados de modo prover escalabilidade e disponibilidade.
- **Detector de Anomalias de Energia**: Componente que analisa os dados do sensores de energia e emite alertas em situações relevantes. Este componente é

[8] https://microk8s.io/
[9] https://kubernetes.io/partners/#conformance
[10] Atualmente, o suporte a Kubernetes com mecanismos de computação confidencial com Intel SGX é provido apenas pela Microsoft Azure [Microsoft, 2020]. A instalação em outros provedores é possível através de instalação do Kubernetes em máquinas físicas (*bare-metal*) com suporte a Intel SGX.
[11] https://litecampus.lsd.ufcg.edu.br/

assinante do tópico de medições, publicadas pelo LiteKafka Gateway, e publica os alertas em um tópico específico.

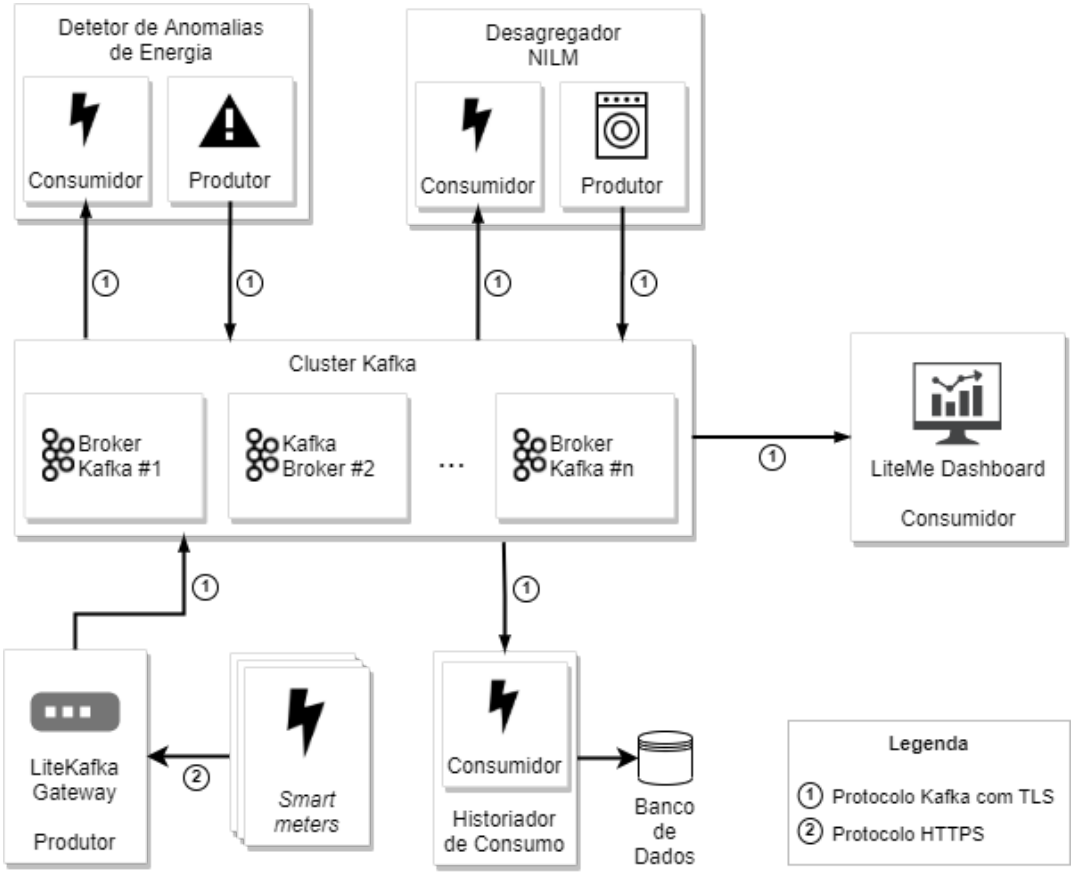



**Figura 1.3. Arquitetura da aplicação LiteCampus**

- **Desagregador NILM**: Possui a função de identificar, dado um perfil de consumo, quais equipamentos estão consumindo energia naquele instante através da aplicação de uma técnica conhecida por monitoramento não-intrusivo de carga (do inglês, *Non-Intrusive appliance Load Monitoring*, NILM [Armel et al., 2013]). Este componente é assinante do tópico de medições, publicadas pelo LiteKafka Gateway, e publica as decomposições em um tópico específico.
- **Historiador de Consumo**: Persiste medições para análises posteriores, como por exemplo, treinamento de modelos de detecção de anomalia ou de cargas. Este componente é assinante do tópico de medições, publicadas pelo LiteKafka Gateway e poderia armazenar os dados em banco de dados ou em sistemas de arquivos para processamento em *batch* (como HDFS). Sendo este componente seguro, poderia também anonimizar os dados sensíveis antes da exportação.
- **LiteMe Dashboard**: Interage com o usuário final, por exemplo, exibindo alertas. Podendo ser baseado em tecnologias abertas (como Grafana[12]), proprietárias (como PowerBI[13]), ou desenvolvidas para o ambiente de uso.

[12] https://grafana.com/
[13] https://powerbi.microsoft.com/pt-br/

O uso de componentes como o desagregador NILM para gerar alertas e recomendações gera um grande potencial de economia [Armel et al, 2013], mas também um risco de privacidade [Barbosa, Brito e Almeida, 2016]. Informações de quando e quais equipamentos foram usados revelam os hábitos detalhados das pessoas naquele ambiente ou dos processos industriais e, portanto, devem ser protegidos.

Para o caso de uso explorado neste trabalho, simplificamos a aplicação acima da seguinte forma: (1) os *Smart Meters* e LiteKafka Gateway serão combinados em um simulador de medições que produzirá dados diretamente no Kafka; (2) ao invés de um *cluster* Kafka, utilizaremos um único nó e não usaremos conexões TLS mutuamente autenticadas ou configurações de controle de acesso aos tópicos; no entanto, é importante notar que mesmo sem essas configurações de autenticação e autorização, os dados sensíveis estão criptografados com mecanismos sofisticados de compartilhamento de chaves, tornando inútil o acesso de pessoas não autorizadas aos dados no barramento; (3) os componentes de análise, detecção de anomalias, e desagregador NILM, serão substituídos por um componente que consome dados sensíveis e publica alertas simples, como de excesso de demanda ou sobretensão; (4) sendo fora do escopo deste tutorial e para economia de espaço, o dashboard será substituído por clientes de linha de comando do próprio Kafka, que mostrarão os alertas produzidos.

Finalmente, é importante adicionar que esta arquitetura de aplicação não se aplica apenas para dados de energia. Em uma aplicação de monitoramento de segurança, o sensores poderiam ser câmeras e os componentes de análise de dados poderiam realizar operações sensíveis como a contagem ou identificação de pessoas. Já em uma aplicação de trânsito, os dados de localização dos usuários, também sensíveis, seriam coletados pelos smartphones e seriam analisados por diferentes componentes que mediriam níveis de congestionamento, tempos de rotas, etc.

### 1.3.1. Instalação do Kafka

Nesta seção mostramos uma instalação simples do Kafka, que inclui a criação de certificados e a configuração básica do mesmo, assim como um teste rápido.

Para o nosso caso de uso, os passos para a criação de certificados incluem a criação de uma autoridade certificadora (AC), para que esta assine uma requisição de certificado para o nosso *broker*. Em um sistema fechado, como em uma empresa ou instituição, a AC seria gerada pelos operadores e confiadas por todas as aplicações. Já em um sistema aberto, público, a requisição de certificado seria gerada pelos operadores da aplicação mas assinada por uma AC pública[14]. Os passos para geração da entidade certificadora estão detalhados na Figura 1.4.

```
01 | $ openssl req -new -x509 -keyout ca-key -out ca-cert -days 365
02 | Generating a RSA private key
03 | ...+++++
04 | ..................+++++
05 | writing new private key to 'ca-key'
06 | Enter PEM pass phrase: senha-ca
07 | Verifying - Enter PEM pass phrase: senha-ca
08 | -----
09 | You are about to be asked to enter information that will be incorporated
10 | into your certificate request.
```

[14] https://estrutura.iti.gov.br/

```
 11 | What you are about to enter is what is called a Distinguished Name or a DN.
 12 | There are quite a few fields but you can leave some blank
 13 | For some fields there will be a default value,
 14 | If you enter '.', the field will be left blank.
 15 | -----
 16 | Country Name (2 letter code) [AU]:BR
 17 | State or Province Name (full name) [Some-State]:Paraiba
 18 | Locality Name (eg, city) []:Campina Grande
 19 | Organization Name (eg, company) [Internet Widgits Pty Ltd]:UFCG
 20 | Organizational Unit Name (eg, section) []:LSD
 21 | Common Name (e.g. server FQDN or YOUR name) []:kafka.lsd.ufcg.edu.br
 22 | Email Address []:sbseg2020@lsd.ufcg.edu.br
```

**Figura 1.4. Criação da entidade certificadora**

Na Figura 1.4, o comando gerará um certificado X.509 chamado *ca-cert* e uma chave privada chamada *ca-key* com validade de um ano. O uso de uma senha de proteção da chave privada evita que um eventual vazamento da mesma (arquivo *ca-key*) comprometa imediatamente a geração de certificados. As outras informações podem ser personalizadas para a aplicação em mãos. O arquivo *ca-cert* pode então ser compartilhado com outros sistemas (entre eles o Kafka), que devem confiar em certificados assinados pela recém-criada AC como veremos a seguir.

Como o Kafka é escrito em Java, a gerência dos certificados de uma aplicação e dos certificados que ela confia é baseado em *Keystores*[15]. Na nossa aplicação, como tipicamente é utilizado, teremos dois armazenamentos do tipo *Keystore*. Um *Keystore* para armazenar os certificados a serem confiados, chamado de *Truststore*, que poderia ser compartilhado por várias aplicações. E um *Keystore* para armazenar as chaves e certificados do Kafka em si. Cada um desses armazenamentos é um arquivo local. A criação do Truststore que confia na nossa AC recém-criada está detalhada na Figura 1.5

```
 01 | $ # Atualiza Truststore
 02 | $ keytool -keystore broker.truststore.jks -alias CARoot -importcert -file
ca-cert
 03 | Enter keystore password:  senha-truststore
 04 | Re-enter new password: senha-truststore
 05 | Owner: EMAILADDRESS=sbseg2020@lsd.ufcg.edu.br, CN=kafka.lsd.ufcg.edu.br,
OU=LSD,
      O=UFCG, L=Campina Grande, ST=Paraiba, C=BR
 06 | Issuer: EMAILADDRESS=sbseg2020@lsd.ufcg.edu.br,CN=kafka.lsd.ufcg.edu.br,
OU=LSD,
      O=UFCG, L=Campina Grande, ST=Paraiba, C=BR
 07 | Serial number: 3fa682094049f344bc026ec673ce3e73ad71c8f8
 08 | Valid from: Tue Sep 15 11:57:51 BRT 2020 until: Wed Sep 15 11:57:51 BRT 2021
… (detalhes do certificado omitidos)
 09 | Trust this certificate? [no]:  yes
 10 | Certificate was added to keystore
```

**Figura 1.5. Ensinando o *broker* a confiar na AC recém-criada**

Em seguida criamos o *Keystore* para as chaves privadas e certificados do próprio Kafka. A Figura 1.6 mostra dois comandos, um para criação de uma *Keystore* e de uma geração da chave privada já armazenada nela e outro para a exportação de uma solicitação de assinatura para um certificado associado a esta chave privada. Note que este *Keystore* é específico para cada aplicação, neste caso, nosso *broker* Kafka.

[15] https://docs.oracle.com/en/java/javase/15/security/general-security1.html

```
01 | $ # Cria uma Keystore para chaves próprias e gera uma chave privada
02 | $ keytool -keystore broker.keystore.jks -alias kafka -validity 365 -genkey
-keyalg
     RSA -storepass keystore-senha -dname
"cn=kafka.lsd.ufcg.edu.br,ou=LSD,o=UFCG,c=BR"
03 | $ # Agora exportar um certificado com base
04 | $ keytool -keystore broker.keystore.jks -alias kafka -certreq -file
     kafka-cert-file -storepass keystore-senha
```

**Figura 1.6. Geração do certificado no *broker* e exportação para assinatura**

De posse do certificado ainda não assinado do nosso *broker*, podemos levá-lo para a validação pela AC. No caso de uma aplicação interna, isso seria feito junto à instalação onde foi gerada a AC, como visto na Figura 1.7. No caso de uma aplicação pública, o arquivo *cert-file* da Figura 1.6 seria enviado para a AC pública, que poderia exigir outras comprovações que o requerente é quem diz ser.

```
01 | $ openssl x509 -req -CA ca-cert -CAkey ca-key -in kafka-cert-file -out
     kafka-cert-signed -days 365 -CAcreateserial -passin pass:senha-ca
```

**Figura 1.7. Assinatura do certificado pela AC**

Com o certificado pronto, importamos ele de volta na *Keystore* do *broker*, como visto na Figura 1.8. Antes do certificado assinado (arquivo *cert-signed*), importamos o certificado da própria AC (arquivo *ca-cert*).

```
01 | $ keytool -keystore broker.keystore.jks -alias CARoot -importcert -file ca-cert
     -storepass keystore-senha
02 | Owner: EMAILADDRESS=sbseg2020@lsd.ufcg.edu.br, CN=kafka.lsd.ufcg.edu.br,
OU=UFCG,
     O=LSD, L=Campina Grande, ST=Paraiba, C=BR
03 | Issuer: EMAILADDRESS=sbseg2020@lsd.ufcg.edu.br, CN=kafka.lsd.ufcg.edu.br,
OU=UFCG,
     O=LSD, L=Campina Grande, ST=Paraiba, C=BR
04 | Serial number: 7f9cb40754386489be49c353697046b5f77e86d8
05 | Valid from: Tue Sep 15 14:17:00 BRT 2020 until: Wed Sep 15 14:17:00 BRT 2021
06 | Certificate fingerprints:
     … (detalhes do certificado omitidos)
07 | Trust this certificate? [no]:  yes
08 | Certificate was added to keystore
09 | $ # Importando a resposta do certificado assinado
10 | $ keytool -keystore broker.keystore.jks -alias kafka -importcert -file
     kafka-cert-signed -storepass keystore-senha
11 | Certificate reply was installed in keystore
```

**Figura 1.8. Inserindo o certificado assinado de volta no Keystore do *broker***

Com os certificados e os armazenamentos *Keystore* configurados, podemos partir para a configuração do Kafka em si. A instalação básica de um Kafka, com apenas um *broker* e sem configurações de autenticação e autorização, é um processo simples e bem documentado. A partir da página *Quickstart*[16] é possível baixar um arquivo compactado com a distribuição mais recente. Um resumo destes passos é mostrado na Figura 1.9.

```
01 | $ tar -xzf kafka_2.13-2.6.0.tgz
02 | $ cd kafka_2.13-2.6.0
03 | $ # Exige Java 8 ou mais recente
04 | $ # Zookeeper é usado para armazenar estado e coordenação entre os brokers
05 | $ bin/zookeeper-server-start.sh config/zookeeper.properties
     … (saída do Zookeeper omitida)
06 | # Em outro terminal, inicie o Kafka
07 | $ bin/kafka-server-start.sh config/server.properties
```

**Figura 1.9. Instalação básica do Kafka**

[16] https://kafka.apache.org/quickstart

Conforme ilustrado na Figura 1.9, primeiro iniciamos um serviço de coordenação, o Zookeeper[17], que pode ser usado com suas configurações padrão, mas deve ser acessível apenas pelas máquinas que executam os *brokers* Kafka. Em seguida podemos iniciar o teste do Kafka, com os comandos exibidos na Figura 1.10 a partir da máquina onde o Kafka foi instalado e usando a porta padrão (parâmetro *--bootstrap-server localhost:9092*). Criamos um novo tópico chamado *sbseg* e consultamos suas propriedades: ele tem apenas uma partição e não tem replicação. Em seguida, produzimos dois eventos, na forma de linhas de texto. Mais detalhes sobre o funcionamento do Kafka podem ser encontrados na página do *Quickstart*.

```
 01 | $ bin/kafka-topics.sh --create --topic sbseg --bootstrap-server localhost:9092
 02 | Created topic sbseg.
 03 | $ bin/kafka-topics.sh --describe --topic sbseg --bootstrap-server
localhost:9092
 04 | Topic: sbseg    PartitionCount: 1     ReplicationFactor: 1  Configs:
      segment.bytes=1073741824
 05 |          Topic: sbseg   Partition: 0   Leader: 0     Replicas: 0    Isr: 0
 06 | $ # Produzindo alguns eventos pela entrada padrão, parar com Control-C
 07 | $ bin/kafka-console-producer.sh --topic sbseg --bootstrap-server localhost:9092
 08 | >Evento em string número 1
 09 | >Evento em string número 2
```

**Figura 1.10. Teste do Kafka (criação de tópico e produção de eventos)**

Complementando o teste da Figura 1.10, podemos consumir os eventos gerados, usando o comando da Figura 1.11. Neste exemplo, o parâmetro *from-beginning* define que o consumidor que resgatar todo o histórico de mensagens disponível (e o período padrão de retenção é de 7 dias). Sem estes parâmetros, apenas as mensagens publicadas após a iniciação do consumidor seriam recebidas.

```
 01 | $  bin/kafka-console-consumer.sh --topic sbseg --from-beginning
--bootstrap-server
      localhost:9092
 02 | Evento em string número 1
 03 | Evento em string número 2
 04 | ^C
```

**Figura 1.11. Teste do Kafka (consumo de eventos)**

Note que todos estes comandos estão sendo executados a partir da máquina onde o Kafka foi instalado. Para execução de outra máquina, o servidor de contato inicial (*bootstrap-server*) deveria ser alterado para um nome igual ao especificado nos arquivos de configuração do Kafka, como visto a seguir, e que seja mapeável por DNS,.

```
 01 | zookeeper.connect=localhost:2181
 02 | security.protocol=SSL
 03 | security.inter.broker.protocol = SSL
 04 | ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1
 05 | listeners=SSL://kafka.lsd.ufcg.edu.br:9092
 06 | ssl.truststore.location=/home/andrey/sbseg/broker.truststore.jks
 07 | ssl.truststore.password=senha-truststore
 08 | ssl.keystore.location=/home/andrey/sbseg/broker.keystore.jks
 09 | ssl.keystore.password=keystore-senha
```

**Figura 1.12. Configuração mínima do servidor para TLS (arquivo *server.tls.properties*)**

Na Figura 1.12, a linha 1 especifica o endereço do Zookeeper levantado antes. Já as linhas 2-5 especificam o protocolo TLS e o endereço e porta onde o serviço espera

[17] https://zookeeper.apache.org/

conexões. As linhas 6-9 especificam os diretórios dos armazenamentos para os certificados que o Kafka confia (*truststore*) e os próprios que ele serve (*keystore*).

Para conectar com um servidor que serve conexões SSL/TLS, o cliente precisa confiar na AC que emitiu o certificado daquele servidor. A Figura 1.13 mostra a arquivo de configuração para o cliente de linha de comando do Kafka usado antes (Figura 1.11), agora configurado para usar SSL/TLS. Note que especificamos um armazenamento de certificados confiáveis (*truststore*), que é uma cópia do usado para o *broker*. Além disso, especificamos que a conexão deve ser SSL/TLS. Finalmente, a Figura 1.14 mostra o comando usado para o teste. É importante destacar que o nome do servidor na configuração da Figura 1.12 deve ser o mesmo do servidor na Figura 1.14, que deve ser o campo CN (*common name*) do certificado emitido para o *broker* (Figura 1.6) e, por fim, deve ser um nome mapeável pelo DNS (ou pelo arquivo *hosts* local na máquina).

```
01 | ssl.truststore.location=/home/andrey/kafka/broker.truststore.jks
02 | ssl.truststore.password=senha-truststore
03 | security.protocol=SSL
```

**Figura 1.13. Teste do Kafka (consumo de eventos)**

```
01 | $  bin/kafka-console-consumer.sh --bootstrap-server kafka.lsd.ufcg.edu.br:9092
     --topic sbseg --from-beginning --consumer.config client.properties
02 | Evento em string número 1
03 | Evento em string número 2
```

**Figura 1.14. Teste do Kafka (consumo de eventos)**

Embora esta configuração básica do Kafka seja um bom ponto de partida, existem várias configurações interessantes e importantes, como replicação, particionamento, tempo de retenção de mensagens, entre outros.

## 1.4. Computação confidencial

Nesta seção apresentamos conceitos de computação confidencial, incluindo a tecnologia de foco neste mini-curso, Intel SGX. Além disso, apresentamos dois conjuntos de ferramentas para desenvolvimento de aplicações confidenciais baseadas em Intel SGX.

### 1.4.1. Intel Software Guard Extensions

Intel *Software Guard Extensions* (SGX) é um conjunto de instruções adicionadas e mudanças no acesso à memória adicionados à arquitetura Intel x86 [Costan e Devadas, 2016]. Intel SGX é um ambiente de execução confiável assistido por *hardware* que permite a criação de regiões protegidas e isoladas de memória dentro do espaço de endereçamento de uma aplicação. Tais regiões protegidas são denominadas enclaves e têm seu controle de acesso reforçado pelo processador. Um processador com SGX habilitado verifica as decisões de mapeamento de memória do sistema operacional, garantindo que apenas instruções que pertencem ao código do enclave tenham acesso às páginas de memória protegidas. Além disso, o conteúdo da memória dedicada a um enclave é criptografada pelo processador.

Infelizmente, a área de memória dedicada para criação de enclaves é pequena, quando comparada a quantidade de memória principal disponível em computadores de propósito geral atuais. A região de memória dedicada para o funcionamento do Intel SGX é chamada de Memória Reservada do Processador (PRM, do Inglês *Processor*

*Reserved Memory*). O tamanho máximo mais comum para a PRM é de 128 MB, e apenas alguns processadores recentes possuem um tamanho máximo de PRM de 256 MB. Como é necessário manter outros metadados, a porção de memória realmente disponível para alocação de enclaves, a Cache de Páginas de Enclaves (EPC, do inglês *Enclave Page Cache*), tem um tamanho em torno de 93 MiB. Em máquinas virtuais, esses limites podem ser diferentes, por exemplo, no Microsoft Azure esses valores são de 28, 56, 112, ou 168 MB [Microsoft, 2020]. Se dados ou códigos carregados por enclaves excederem esse o tamanho da EPC, páginas de memória protegidas são desalojadas e enviadas para a memória regular, processo que aumenta a latência de acesso à memória em ordens de magnitude. Deve-se então tentar manter a memória de trabalho dos enclaves em execução em uma máquina dentro desses limites sob o risco de perdas consideráveis de desempenho [Arnautov, 2016].

Uma característica importante das aplicações SGX é que elas são sempre compostas por ao menos uma parte que não é protegida pelo ambiente de execução confiável, mas pode ter vários enclaves independentes. Como um enclave não consegue fazer chamadas de sistema ao sistema operacional, a porção não confiável da aplicação executa chamadas de entrada e saída, e também é responsável por iniciar os enclaves. De todo modo, os enclaves devem ser construídos de maneira a nunca passar dados sensíveis em texto plano para a parte não confiável da aplicação.

### 1.4.2. Atestação Remota

Além da proteção memória através de criptografia, que entrega confidencialidade e integridade, Intel SGX também implementa o suporte à atestação remota. O processo de atestação remota permite que uma aplicação desafiante ganhe confiança de que um enclave está executando realmente em uma máquina com Intel SGX habilitado [Costan e Devadas, 2016]. Através deste processo, a aplicação obtém propriedades de segurança importantes sobre a máquina, como se a máquina está com o *firmware* atualizado ou se a funcionalidade de *hyperthreading* está habilitada (o que facilitaria ataques de canal lateral). Ao final do processo de atestação, o desafiante pode verificar as identidades do enclaves: a identidade do enclave e a identidade do assinante.

A identidade do enclave, chamada de **MRENCLAVE**, é o resultado de uma operação de *hash*, utilizando o algoritmo SHA-256, envolvendo um registro de todas as operações realizadas no processo de criação do enclave. Dessa maneira, todo o conteúdo relacionados às páginas de memória de um enclave, incluindo código e *flags* de segurança das páginas, influi na identidade do enclave. Já a identidade do assinante, chamada de **MRSIGNER**, permite a identificação daquele que assina a aplicação SGX.

Atualmente, a Intel provê duas maneiras de realizar o processo de atestação. A primeira, usando o Serviço de Atestação da Intel (IAS, do inglês *Intel Attestation Service*), foca no serviço de verificação de enclaves provido pela própria Intel e utiliza um esquema de assinatura de grupo, que pode prover privacidade e é verificável somente pela Intel. Utilizando o esquema de assinatura EPID (do inglês *Enhanced Privacy ID*) é possível ainda escolher entre dois modos de uso do IAS: um modo que produz *quotes* vinculáveis que permite identificar se duas assinaturas foram geradas por uma mesma plataforma; e um modo que produz *quotes* não vinculáveis que não permite permite essa identificação. A segunda forma de atestação se dá através do uso das

Primitivas de Atestação de Datacenter (DCAP, do inglês *Datacenter Attestation Primitives*). Usar o DCAP é mais flexível e é baseada em assinaturas ECDSA (do inglês, *Elliptic Curve Digital Signature Algorithm*, ou Algoritmo de Assinatura Digital de Curvas Elípticas). Essas primitivas de atestação permitem a construção de serviços de atestação locais e têm como alvo aplicações em ambientes em que não se deseja delegar decisões de confiança para terceiros, como a Intel. Além disso, atestação via DCAP é mais indicada para ambientes onde a latência do acesso à internet precisa ser evitada.

A Figura 1.15 ilustra, em alto nível, o processo de atestação. No passo 1, a aplicação desafiante inicia o processo de atestação para verificar se a aplicação SGX está de fato executando em uma máquina com Intel SGX habilitado. O desafio gerado no passo 1 inclui um número aleatório, para garantir o atualidade da resposta da aplicação SGX. A parte não confiável da aplicação SGX serve de interface para os enclaves durante o processo. Contudo, criptografia e algoritmos de autenticação de mensagens são utilizados para garantir propriedades como integridade e confidencialidade quando necessário. No passo 2, a parte não confiável requisita que o enclave gere um relatório de atestação, passando o número aleatório recebido da aplicação desafiante. O enclave então gera um relatório e um manifesto, e os envia como resposta para a parte não confiável, no passo 3.

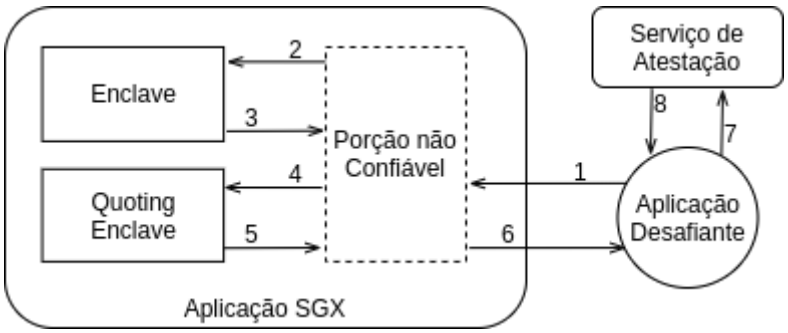



**Figura 1.15. Processo de Atestação Remota**

No passo 4 da Figura 1.15, a parte não confiável da aplicação entrega os artefatos gerados pelo enclave para um outro enclave, um enclave especial provido pela Intel chamado de *Quoting Enclave*, que tem acesso a chaves de assinatura que nunca são expostas para fora do processador. O relatório é assinado, juntamente com o manifesto, gerando uma estrutura chamada de ***quote***, que é encaminhada para a parte não confiável, no passo 5, que por sua vez, encaminha para a aplicação desafiante, no passo 6. De posse do *quote*, a aplicação desafiante agora pode conferir se deve confiar no conteúdo. Então, no passo 7, a aplicação desafiante verifica junto a um serviço de atestação a validade do *quote* apresentado. A resposta do serviço de atestação, passo 8, inclui um relatório de verificação de atestação (AVR, do inglês *Attestation Verification Report*) que contém informações sobre a segurança da plataforma onde o enclave desafiado diz estar executando. Após verificar que pode confiar no *quote* apresentado pela aplicação SGX, a aplicação desafiante pode verificar a identidade do enclave apresentada no *quote* junto aos seus valores de referência.

### 1.4.3. Modelo de ameaças

A implementação do ambiente de execução confiável baseado em Intel SGX assume um modelo de ameaça em que um atacante tem controle sobre a máquina com privilégios de superusuário. Assim, um atacante pode controlar toda a pilha de software executando na máquina, incluindo o sistema operacional. O atacante usa então esses poderes para extrair dados, chaves de criptografia ou modificar código das aplicações.

As motivações para tal modelo de ataque são variadas. Em primeiro lugar, a identidade dos operadores podem ser roubadas por um atacante, por exemplo, através de ataques elaborados e direcionados (*spear phishing*). Além disso, operadores ou seus provedores podem ser forçados a fornecer os dados (por exemplo, através de recursos legais). Mesmo se provedor e operadores forem confiáveis, as pilhas de software têm uma base de código muito grande e complexa. Considerando apenas porções do sistema que podem permitir o ganho de privilégios (como o núcleo do Linux em si, o gerente de nuvem, como o OpenStack, ou o virtualizador, como o KVM), ainda assim há dezenas de milhões de linhas de código. Finalmente, subestimar o modelo de ameaças é caro, o custo médio de um vazamento de dados está estimado em US$ 3,86 milhões, com tempo de detecção e contenção médio de 280 dias [IBM Security, 2020].

Assim, assumindo que a infraestrutura e a pilha de software não são confiáveis, nossa base de confiança é reduzida para o software da aplicação, o software de suporte do SGX (por exemplo, as bibliotecas que suportam atestação) e o hardware do SGX em si. Assim como no modelo típico, assumimos que o software do enclaves da aplicação e as bibliotecas de desenvolvimento estão livres de *bugs*. Além disso, assumimos que ataques de canal lateral estão fora do escopo do SGX. Esta consideração tem impactos que dependem de como o software foi desenvolvido. Para código desenvolvido diretamente com o kit de desenvolvimento da Intel (discutido a seguir), o total controle do que é código confiável exige que o desenvolvedor se responsabilize por mecanismos de redução de riscos. Já para código desenvolvido com apoio de um ambiente de execução (*runtime*) como o SCONE, mecanismos implementados no próprio ambiente de execução permitem que riscos sejam mitigados nas aplicações. Como um exemplo, Varys [Oleksenko et al., 2018] mitiga ataques de canal lateral através da detecção de taxas anormais de interrupção na execução do código do enclave e do uso de *threads* irmãs executando simultaneamente.

### 1.4.4. Preparação do ambiente

Para executar aplicações Intel SGX, é necessário que o ambiente possua o *driver* Intel SGX instalado. As demais dependências necessárias para a execução, como o SGX PSW, já estão embutidas nos contêineres que usaremos. Considerando que o ambiente utilize a distribuição Linux Ubuntu 16.04 ou superior, a instalação do *driver* pode ser feita como ilustrado na Figura 1.16.

```
01 | $ sudo apt-get install build-essential git linux-headers-$(uname -r)
02 | $ git clone https://github.com/intel/linux-sgx-driver.git
03 | $ cd linux-sgx-driver
04 | $ make && sudo make install
05 | $ sudo depmod -a
06 | $ sudo modprobe isgx
```

**Figura 1.16 - Passos para instalação do driver SGX**

Para verificar se o *driver* foi instalado corretamente, verifique a existência do arquivo de dispositivo */dev/isgx*. Mais informações sobre a instalação do *driver* podem ser encontradas no repositório do GitHub[18]. Uma opção para instalação é utilizando o *script* de instalação disponibilizado pela Scontain[19], criadora do SCONE. Além do *driver* básico, o *script* permite instalar um *driver* que possui extensões que permitem o monitoramento de recursos SGX da máquina.

### 1.4.5. Desenvolvendo com o Intel SGX SDK

A Intel provê um kit de desenvolvimento de software para que desenvolvedores possam criar aplicações que executam em enclaves SGX, o **Intel SGX SDK**. O SGX SDK permite criar enclaves utilizando as linguagens de programação C e C++. Junto com esse kit de desenvolvimento, a Intel disponibiliza também o SGX PSW (do inglês, *Platform SoftWare*). O PSW contém enclaves especiais como o já mencionado *Quoting Enclave* e o *Launch Enclave*, que permite a criação de novos enclaves. Portanto, para executar aplicações SGX é necessário ter o SGX PSW instalado (além do *driver* SGX), mas para desenvolver, também é necessário ter o SGX SDK.

O desenvolvimento utilizando o SGX SDK tem algumas particularidades. Uma aplicação SGX é dividida em duas porções. Uma porção é a parte confiável da aplicação, que pode ser composta de um ou mais enclaves, e é responsável pela computação sobre os dados sensíveis. A outra porção é a porção não confiável, que executa em nível de usuário comum e é responsável pelas operações que necessitam de chamadas ao sistema operacional, como comunicação com outras aplicações.

É no código da porção não confiável que reside o ponto de início de uma aplicação SGX. Após o início da aplicação, a função *sgx_create_enclave* do enclave deve ser chamada para criação de enclaves. A partir da criação de um enclave, a comunicação entre o enclave criado e a porção não confiável da aplicação deve ser especificada utilizando uma linguagem especial, chamada de linguagem de definição de enclave (EDL, do inglês *Enclave Definition Language*). Utilizando essa linguagem, o desenvolvedor pode definir quais chamadas um enclave pode realizar para a parte não confiável e quais chamadas a parte não confiável pode realizar para um enclave. Quando a chamada é de um enclave para a parte não confiável, ela é denominada ***ocall***, Já quando a chamada é realizada da parte não confiável para o enclave, ela é denominada ***ecall***. A Figura 1.17 ilustra um exemplo de um arquivo EDL que define a interface de comunicação da aplicação "Alô, Mundo" que estudaremos a seguir.

```
 01 | enclave {
 02 |     trusted{
 03 |         public void app_to_enclave([in, string] char *secretIn);
 04 |            public void enclave_to_app([out, size = len] char *secretOut, size_t
len);
 05 |     };
 06 |     untrusted{
 07 |         void print_debug([in, string] char *dbg_message);
 08 |     };
 09 |
 10 | };
```

**Figura 1.17. Trecho de Código - EDL: aplicação Alô, mundo!**

[18] https://github.com/intel/linux-sgx-driver
[19] https://github.com/scontain/SH

O desenvolvedor deve tomar o cuidado de não expor dados sensíveis através de *ecalls* e *ocalls*. As bibliotecas de criptografia disponíveis no SGX SDK podem ajudar na transferência de dados de maneira segura. Bibliotecas como o *mbedtls-compat-sgx*[20] permitem estabelecer comunicação segura entre enclaves e aplicações remotas.

A seguir criamos uma aplicação SGX simples para entender os principais componentes envolvidos. Como outras aplicações SGX SDK, ela é composta de uma parte não confiável, que chamaremos de *App* e um enclave, que chamaremos de *Enclave*[21]. O *App* é responsável pela criação do enclave e pela intermediação da comunicação de e para o enclave. O *Enclave* receberá os bytes enviados pelo *App* e os guardará em uma variável local, em memória protegida, para entregar de volta em uma consulta futura. Em maneira geral, para o desenvolvedor, os artefatos mais importantes do *Enclave* são o arquivo EDL, que contém a definição da interface entre o enclave e a parte não confiável da aplicação, e os arquivos C/C++ que contém o código responsável por implementar as *ecalls* definidas no EDL. A Figura 1.18 apresenta o código do *App*.

```
01 | #include "enclave_u.h"
02 | #include "sgx_urts.h"
03 | #include <string.h>
04 | #include <stdio.h>
05 |
06 | #define ENCLAVE_NAME "enclave.signed.so"
07 |
08 | void print_debug(char *dbg_message)
09 | {
10 |     printf("%s", dbg_message);
11 | }
12 |
13 | int main()
14 | {
15 |
16 |     sgx_status_t ret = SGX_SUCCESS;
17 |     sgx_launch_token_t launch_token;
18 |     int updated = 0;
19 |     sgx_enclave_id_t eid;
20 |     ret = sgx_create_enclave(
21 |                 ENCLAVE_NAME,//         const char *file_name,
22 |                 SGX_DEBUG_FLAG,//         const int debug,
23 |                 &launch_token,//         sgx_launch_token_t *launch_token,
24 |                 &updated,//         int *launch_token_updated,
25 |                 &eid,//         sgx_enclave_id_t *enclave_id,
26 |                 NULL//         sgx_misc_attribute_t *misc_attr
27 |     );
28 |
29 |     if (ret != SGX_SUCCESS)
30 |     {
31 |         printf("\nUnable to create enclave!\n");
32 |         return -1;
33 |     }
34 |     printf("\nSGX enclave successfully created!\n");
35 |
36 |     char *secretIn = "MyNewSecret";
37 |     ret = app_to_enclave(eid, secretIn);
38 |
39 |     if (ret != SGX_SUCCESS)
40 |     {
41 |         printf("\nUnable to pass secret to enclave!\n");
42 |         return -1;
43 |     }
44 |     printf("\nSuccessfully passed secret to enclave!\n");
45 |
46 |     char *secretOut = (char *) malloc (strlen(secretIn)+1);
```

[20] https://github.com/ffosilva/mbedtls-compat-sgx

[21] O código completo está disponível em https://git.lsd.ufcg.edu.br/lsd-sbseg-2020/alo-mundo-sgx-sdk

```
47 |     ret = enclave_to_app(eid, secretOut, strlen(secretIn)+1);
48 |
49 |     printf("\n%s\n", secretOut);
50 |
51 |     ret = sgx_destroy_enclave(eid);
52 |     if (ret != SGX_SUCCESS)
53 |     {
54 |         printf("\nUnable to destroy enclave!\n");
55 |         return -1;
56 |     }
57 |     printf("\nSGX enclave successfully destroyed!\n");
58 |     return ret;
59 |
60 | }
```

**Figura 1.18. Trecho de Código - App: aplicação Alô, mundo!**

Na linha 20, o *App* faz uma chamada à função *sgx_create_enclave* para inicializar o *Enclave*. Após a criação do nosso enclave, o *App* então cria uma variável chamada *secretIn* que aponta para o valor "*MyNewSecret*" que vai ser enviado para o *Enclave*, na linha 36. Na linha 37, o *App* invoca a nossa primeira *ecall*: *app_to_enclave*. Esta *ecall* está definida no EDL abaixo, e envia o valor apontado por *secretIn* para o *Enclave*. Em seguida, o *App* invoca nossa segunda *ecall*, camada *enclave_to_app*, para recuperar o valor guardado no enclave SGX. Depois trocar dados com o enclave da aplicação, o *App* faz uma chamada à função *sgx_destroy_enclave*, para destruir o *Enclave*, e então termina a execução.

O EDL que define a interface entre o *Enclave* e o *App* para esta aplicação é o ilustrado na Figura 1.17 e é dividido em duas seções: *trusted* e *untrusted*. Na seção *trusted* são definidas as chamadas ao enclave, *ecalls*. Nas linhas 4 e 5 são declaradas as duas funções do *Enclave* que o *App* utiliza. A declaração das *ecalls* é similar à declaração de funções em C/C++. No entanto, entre colchetes precisamos indicar, no caso de ponteiros, qual a direção do dado e também tamanho do dado em bytes. Em especial, se o tipo de ponteiro for *char* * e o último byte for *0x00*, podemos substituir a indicação de tamanho pelo identificador *string*. Na seção *untrusted* temos a definição de uma *ocall*, chamada de um enclave para a parte não confiável da aplicação, que utilizamos neste exemplo para imprimir o valor recebido dentro do *Enclave*.

Finalmente, a Figura 1.19 ilustra o código do *Enclave* que implementa as interfaces definidas no EDL. Apesar de escrito em C/C++, o código do enclave não pode dispor da biblioteca C padrão. Isso ocorre em função das limitações impostas pela propriedade de isolamento do ambiente de execução confiável.

```
01 | #include "enclave_t.h"
02 | #include "stdlib.h"
03 | #include "string.h"
04 | char *secret;
05 | void app_to_enclave(char *secretIn)
06 | {
07 |     secret = (char *) calloc(strlen(secretIn)+1,1);
08 |     memcpy(secret, secretIn, strlen(secretIn));
09 |     print_debug(secret);
10 | }
11 | void enclave_to_app(char *secretOut, size_t len)
12 | {
13 |     memcpy(secretOut, secret, len);
14 | }
```

**Figura 1.19. Trecho de Código - Enclave: aplicação Alô, mundo!**

A execução é realizada com ajuda de um script bash que constrói um contêiner e executa a aplicação. É necessário ter o Docker instalado na máquina e também o driver do SGX, que foi mencionado na seção "Preparação do ambiente."

### 1.4.6. Desenvolvendo com SCONE

Como discutido acima, o uso do Intel SGX SDK reduz severamente sua aplicabilidade para aplicações preexistentes, cujo custo de reescrita (no melhor caso, de alguns funções, caso a aplicação já seja escrita em C/C++, e no pior caso, da aplicação inteira, caso ela seja escrita em outra linguagem) pode ser proibitivo. Desta forma, ambientes de execução foram criados para permitir o uso de código não modificado.

Assim como sistemas como serviços alternativos como Anjuna[22] e Fortanix[23], SCONE[24] (Secure CONtainer Environment) foi criado para permitir que aplicações inteiras e não modificadas rodem dentro de enclaves Intel SGX. SCONE utiliza versões modificadas de bibliotecas do sistema como *musl-libc* ou *glibc*, e um compilador *gcc* especial, o que permite que o código-fonte de aplicações já existentes seja compilado para execução confidencial com a mínima necessidade de modificação por parte do desenvolvedor. SCONE suporta uma variedade de linguagens de programação, como C, C++, Java, Python, Go e JavaScript.

Além disso, SCONE provê uma plataforma para desenvolvimento e implantação de aplicações confidenciais com Intel SGX. Esta plataforma conta com ferramentas de atestação local e remota, geração, compartilhamento e entrega segura de segredos, além de criptografia transparente de arquivos e de tráfego de rede, o que retira do desenvolvedor a responsabilidade de implementar manualmente estas funcionalidades com o SGX SDK.

A seguir são apresentados alguns desses conceitos e funcionalidades, usando como base uma aplicação-exemplo simples escrita na linguagem Python.

#### 1.4.6.1. Alô, mundo!

Para começar, considere um programa Python simples que exibe a mensagem "Alô, mundo!" na tela, *programa.py* (Figura 1.20).

```
01 | print("Alo, mundo!")
```

**Figura 1.20. Trecho de Código - *programa.py*: Alô, mundo! na linguagem Python**

Para executar essa aplicação simples dentro de um contêiner Docker, é necessária a criação de um arquivo de definição de imagem, ou *Dockerfile* (Figura 1.23).

```
01 | FROM python:3
02 | COPY programa.py .
03 | ENTRYPOINT [ "python3", "programa.py" ]
```

**Figura 1.21. Trecho de Código - *Dockerfile*: aplicação Alô, mundo!**

Para construir a imagem e executar o contêiner, basta executar os comandos descritos na Figura 1.22.

[22] https://www.anjuna.io/microsoft-azure
[23] https://fortanix.com/products/enclave-manager/
[24] https://sconedocs.github.io/

```
01 | $ docker build . -t sbseg-alo-mundo
02 | Sending build context to Docker daemon  3.072kB
03 | Step 1/3 : FROM python:3
04 |  ---> 28a4c88cdbbf
05 | Step 2/3 : COPY programa.py .
06 |  ---> 33ae2928e067
07 | Step 3/3 : ENTRYPOINT [ "python3", "programa.py" ]
08 |  ---> Running in 1744daef5b94
09 | Removing intermediate container 1744daef5b94
10 | ---> b050e55a0823
11 | Successfully built b050e55a0823
12 | Successfully tagged sbseg-alo-mundo:latest
13 |
14 | $ docker run -it --rm sbseg-alo-mundo
15 | Alo, mundo!
```

**Figura 1.22. Criação e execução do contêiner Alô, mundo!**

Para construir executar a mesma aplicação dentro de um enclave SGX com SCONE, basta utilizar um interpretador Python seguro. Por conveniência, SCONE oferece um conjunto de imagens pré-compiladas de aplicações populares, entre elas o interpretador Python. Dessa forma, para executarmos a mesma aplicação em um enclave, basta modificarmos a imagem-base no *Dockerfile* (instrução *FROM*, na linha 1). A Figura 1.23 mostra a *scone.Dockerfile* utilizada.

```
01 | FROM sconecuratedimages/public-apps:python-3.7.3-alpine3.10
02 | COPY programa.py .
03 | ENTRYPOINT [ “python3”, “programa.py” ]
```

**Figura 1.23. Trecho de Código - *scone.Dockerfile*: aplicação Alô, mundo!**

Com SCONE, a construção e execução da imagem é similar, como ilustrado na Figura 1.24. No entanto, agora a aplicação necessita acessar o *driver* de Intel SGX, o que pode ser feito pelo parâmetro *--device* do Docker (assume-se que o *driver* está instalado e disponibiliza uma interface em */dev/isgx*). O MRENCLAVE da aplicação pode ser obtido ao executá-la com a variável de ambiente *SCONE_HASH=1* definida.

```
01 | $ docker build . -t sbseg-alo-mundo-scone -f scone.Dockerfile
02 | Sending build context to Docker daemon  4.096kB
03 | Step 1/3 : FROM sconecuratedimages/public-apps:python-3.7.3-alpine3.10
04 |  ---> 7985d7286c75
05 | Step 2/3 : COPY programa.py .
06 |  ---> 801811ae27ea
07 | Step 3/3 : ENTRYPOINT [ "python3", "programa.py" ]
08 |  ---> Running in 350ad362fb0d
09 | Removing intermediate container 350ad362fb0d
10 |  ---> 2622e5882129
11 | Successfully built 2622e5882129
12 | Successfully tagged sbseg-alo-mundo-scone:latest
13 |
14 | $ docker run -it --rm --device /dev/isgx sbseg-alo-mundo-scone
15 | Alo, mundo!
16 |
17 | $ docker run -it --rm --device /dev/isgx -e SCONE_HASH=1 sbseg-alo-mundo-scone
18 | 41f0117a3c62966b48ef6e2388b5fe7ff719b1f48abbf417e855fff0546a8e0d
```

**Figura 1.24. Criação, execução e obtenção de MRENCLAVE do Alô, mundo! no SCONE**

Como a aplicação inteira é executada dentro de um enclave Intel SGX, SCONE gerencia a comunicação com o sistema operacional (que não é confiável, de acordo com seu modelo de ameaça) através de filas e *threads* especiais que gerenciam as chamadas de sistema requisitadas pela aplicação, numa abordagem assíncrona. Note que, devido à limitação na quantidade de memória protegida, ou EPC, disponível, é comum que as

aplicações necessitem paginar seu conteúdo para a memória principal, o que incorre em perda de desempenho, como discutido na Seção 1.4.1.

SCONE também oferece, como mencionado anteriormente, uma plataforma que entrega ao desenvolvedor mais controle sobre a execução segura de suas aplicações, através de mecanismos como atestação remota e entrega de segredos. Essa plataforma, bem como algumas de suas funcionalidades, são exploradas nas seções a seguir.

#### 1.4.6.2. Atestação remota

Para entender como funciona a atestação remota no SCONE, é necessário apresentar dois componentes essenciais: o **CAS** (do inglês, *Configuration and Attestation Service*) e o **LAS** (do inglês, *Local Attestation Service*). O CAS é o responsável por atestar uma aplicação SCONE. Uma vez que uma aplicação é atestada, o CAS pode provisionar segredos e configurações de forma segura. O LAS, por sua vez, é o agente responsável pela atestação local da aplicação SGX, gerando um *quote* de atestação que é, então, enviado ao CAS. O *quote*, como mencionado na Seção 1.4.2, contém informações do enclave a ser atestado (como MRENCLAVE e MRSIGNER) e da plataforma (incluindo dados sobre funcionalidades que influenciam a atestação, como a versão do *firmware* e se o *hyperthreading* está habilitado). Se o *quote* for o esperado, o CAS então finaliza o processo de atestação, provisionando também segredos e configurações.

O CAS é o ponto central da arquitetura de atestação, sendo também o local de armazenamento de segredos e configurações. Um CAS pode servir múltiplas aplicações. Novas aplicações são registradas no CAS por meio de sua API, através de manifestos no formato YAML, onde a identidade de enclave esperada é descrita, e segredos e configurações são definidos. Esses manifestos são chamados de arquivos de sessão. O LAS, por sua vez, serve apenas para o nó em que ele está sendo executado, uma vez que se comunica diretamente com o *hardware* para efetuar a atestação local do enclave. Assim, é necessário executar um LAS por nó.

A Figura 1.25 ilustra como iniciar um CAS e um LAS localmente em contêineres Docker. O LAS expõe a porta 18766 para a atestação local de enclaves. O CAS, por sua vez, expõe duas portas: 8081, para submissão de sessões, e 18765, para atestação remota de enclaves. Note-se que o CAS também se atesta localmente, razão pela qual é definida a variável de ambiente *SCONE_LAS_ADDR*, cujo valor, *172.17.0.1*, refere-se à interface de rede padrão do Docker.

```
 01 | $ docker run -dt --rm --name las --device /dev/isgx -p 18766:18766
      sconecuratedimages/services:las
 02 | $ docker run -dt --rm --name cas --device /dev/isgx -p 18765:18765 -p 8081:8081
-e
      SCONE_LAS_ADDR=172.17.0.1 sconecuratedimages/services:cas
```

**Figura 1.25. Iniciando CAS e LAS localmente via Docker**

```
01 | name: sessao-exemplo
02 | version: 0.3
03 | services:
04 |    - name: alo-mundo
05 |                                                                    mrenclaves:
[41f0117a3c62966b48ef6e2388b5fe7ff719b1f48abbf417e855fff0546a8e0d]
06 |      command: python3 programa.py
07 |      pwd: /
08 |      environment:
09 |           SCONE_MODE: "hw"
```

```
10 |           VARIAVEL_SECRETA: "conteudosecreto"
11 | security:
12 |   attestation:
13 |     tolerate: [debug-mode, hyperthreading, outdated-tcb]
14 |     ignore_advisories: "*"
```

**Figura 1.26. Trecho de Código - *sessao.yml*: exemplo de arquivo de sessão SCONE**

A Figura 1.26 mostra um exemplo simples de arquivo de sessão SCONE. A sessão *sessao-exemplo* (linha 1) é criada e define um único *service* (linhas 4 a 10). Em SCONE, um *service* equivale a uma instância de aplicação. Mais especificamente, a aplicação *python3* (linha 6) está sendo registrada, e seu MRENCLAVE esperado é *41f0117a3c62966b48ef6e2388b5fe7ff719b1f48abbf417e855fff0546a8e0d* (linha 5). Note que o MRENCLAVE, ou identidade de enclave, define a aplicação. Alterações no código resultam em um MRENCLAVE completamente diferente. Caso uma aplicação possua um MRENCLAVE diferente do esperado, o processo de atestação é abortado e a aplicação é finalizada sem receber nenhum segredo ou configuração do CAS. Argumentos da aplicação (linha 6) e variáveis de ambiente (linhas 8 a 10) também são protegidos, apenas sendo entregues após o processo de atestação ser completado com sucesso. Por padrão, o CAS inicia em modo de produção, e não tolera vulnerabilidades no *hardware* onde o enclave está sendo executado. Estas vulnerabilidades, que são descritas no *quote* do processo de atestação, englobam *firmwares* defasados, a presença de *hyperthreading*, enclaves em modo *debug* (que podem ter seu conteúdo inspecionado), entre outros. Para fins de teste e desenvolvimento, é possível relaxar os requisitos do CAS através do campo *security* (linhas 11 a 14). Nela é possível definir não só quais vulnerabilidades são toleradas (no campo *tolerate*), como quais recomendação de atualização de *firmware* da Intel (ou *Intel Update Advisories)* podem ser ignoradas. A lista completa de vulnerabilidades de *hardware* para atestação pode ser encontrada na documentação do SCONE.

No caso de aplicações compiladas, como as escritas em linguagens como C, C++ e Go, o MRENCLAVE identifica o arquivo binário executável da aplicação em si. Em linguagens interpretadas, o MRENCLAVE identifica o interpretador. No exemplo da Figura 1.28, portanto, o MRENCLAVE apenas diz se o interpretador foi ou não modificado, sem oferecer garantias sobre o código sendo interpretado. Para contornar essa limitação, SCONE permite a atestação e criptografia de código-fonte (e arquivos em geral) através de uma funcionalidade chamada de FSPF (do inglês, *FileSystem Protection File*). A próxima seção aborda FSPF e outras funcionalidades do SCONE, como geração e entrega de segredos e criptografia transparente.

O CAS oferece uma API para criação e gerenciamento de sessões, disponível na porta *8081*. A Figura 1.27 mostra como obter certificados para contactar um CAS localizado num contêiner Docker local, enquanto a Figura 1.28 mostra como enviar uma nova sessão.

```
01 | $ cat > clientcertreq.conf <<EOF
02 | [req]
03 | [req]
04 | distinguished_name = req_distinguished_name
05 | x509_extensions = v3_req
06 | prompt = no
07 |
08 | [req_distinguished_name]
```

```
09 | C = EU
10 | ST = Germany
11 | L = Dresden
12 | O = Scontain
13 | OU = CLI
14 | CN = cli.scontain.com
15 | [ v3_req ]
16 |
17 | # Extensions to add to a certificate request
18 | basicConstraints = CA:FALSE
19 | keyUsage = nonRepudiation, digitalSignature, keyEncipherment
20 | subjectAltName = @alt_names
21 |
22 | [alt_names]
23 | DNS.1 = cli.scontain.com
24 | DNS.2 = scone-cli.scontain.com
23 | EOF
24 |
25 | $ openssl req -newkey rsa:4096 -days 365 -nodes -x509 -out client.pem -keyout
     client-key.pem -config clientcertreq.conf
```

**Figura 1.27. Obtendo certificados para CAS via *openssl***

```
01 | $ export SCONE_CAS_ADDR=172.17.0.1
02 | $ curl -v -k -s --cert client.pem  --key client-key.pem  --data-binary
@sessao.yml -X POST https://$SCONE_CAS_ADDR:8081/session
```

**Figura 1.28. Criando uma nova sessão em um CAS local. Para contactar outro CAS, basta mudar o valor de *SCONE_CAS_ADDR*. O arquivo de sessão é *sessao.yml***

Para executar uma aplicação SCONE com atestação remota, é necessário definir as variáveis de ambiente *SCONE_CAS_ADDR*, *SCONE_LAS_ADDR* e *SCONE_CONFIG_ID*, que apontam, respectivamente, para o endereço do CAS, o endereço do LAS, e o nome da sessão e do *service*, no formato *SESSÃO/SERVIÇO*. A aplicação da Figura 1.26, por exemplo, seria referenciada com *SCONE_CONFIG_ID=sessao-exemplo/alo-mundo*.

### 1.4.6.3. Segredos

Uma vez atestada, a aplicação pode receber segredos e configurações do CAS de forma segura. Segredos são definidos no arquivo de sessão do SCONE, no campo *secrets*, que permite definir segredos de vários tipos, como texto, binário e até certificados e chaves privadas. O desenvolvedor tem a opção de não especificar o conteúdo dos segredos, o que fará com que o CAS gere segredos de conteúdo aleatório. Uma vez criados, segredos podem ser referenciados pelas aplicações definidas no arquivo de sessão (por exemplo, podem ser injetados no ambiente ou em arquivos), e mesmo exportados para outros arquivos de sessão, inclusive para outro CAS.

```
01 | name: sessao-exemplo-segredos
02 | version: 0.3
03 |
04 | services:
05 |    - name: alo-mundo
06 |                                                                    mrenclaves:
[41f0117a3c62966b48ef6e2388b5fe7ff719b1f48abbf417e855fff0546a8e0d]
07 |      command: python3 programa.py
08 |      pwd: /
09 |      image_name: alo-mundo
10 |      environment:
11 |           SCONE_MODE: hw
12 |           UM_SEGREDO: $$SCONE::segredo1$$
13 | images:
14 |    - name: alo-mundo
```

```
15 |     injection_files:
16 |          - path: /etc/segredo.txt
17 |            content: $$SCONE::segredo2$$
18 | secrets:
19 |   - name: segredo1
20 |     kind: ascii
21 |     size: 16
22 |   - name: segredo2
23 |     kind: ascii
24 |     value: "isto eh um segredo!!!"
25 | security:
26 |   attestation:
27 |     tolerate: [debug-mode, hyperthreading, outdated-tcb]
28 |     ignore_advisories: "*"
```

**Figura 1.29. Trecho de Código - *sessao-segredos.yml*: exemplo de arquivo de sessão SCONE com segredos**

Na Figura 1.29, o arquivo de sessão do *Alô, mundo!* é estendido para incluir dois segredos do tipo *ascii*, ou seja, texto. O segredo *segredo1* (linhas 19 a 21), não possui seu conteúdo definido, o que fará com que o CAS gere um texto aleatório de 16 bytes. O segredo *segredo2* (linhas 22 a 24), por sua vez, tem seu conteúdo definido: "isto é um segredo!!!". Estes segredos são então injetados no ambiente (linha 12) e no sistema de arquivos (linhas 15 a 17) da aplicação. A notação *$$SCONE::NOME$$* serve para referenciar segredos e seu conteúdo ao longo do arquivo de sessão, onde *NOME* é o nome do segredo. A aplicação *Alô, mundo!* pode ser modificada para exibir também o conteúdo desses dois segredos (Figura 1.30).

```
01 | import os
02 | print("Alo, mundo!")
03 | print("UM_SEGREDO: %s" % os.environ.get("UM_SEGREDO"))
04 | arquivo = open("/etc/segredo.txt", "r")
05 | print(arquivo.read())
06 | arquivo.close()
```

**Figura 1.30. Trecho de Código - programa.py: Alô, mundo! com segredos**

A construção da aplicação não sofre alterações, e a mesma *scone.Dockerfile* pode ser utilizada. A execução da aplicação sem as variáveis de atestação do SCONE ocasionará um erro, já que nem a variável de ambiente *UM_SEGREDO*, nem o arquivo */etc/segredo.txt* existirão a menos que a aplicação seja atestada (Figura 1.31).

```
01 | $ docker build . -t sbseg-alo-mundo-scone-segredos -f scone.Dockerfile
02 | Sending build context to Docker daemon  4.096kB
03 | Step 1/3 : FROM sconecuratedimages/public-apps:python-3.7.3-alpine3.10
04 |  ---> 179f05bee7c7
05 | Step 2/3 : COPY programa.py .
06 |  ---> 9129296afc6d
07 | Step 3/3 : ENTRYPOINT [ "python3", "programa.py" ]
08 |  ---> Running in 72a81ea76a51
09 | Removing intermediate container 72a81ea76a51
10 |  ---> dd7f07ee4214
11 | Successfully built dd7f07ee4214
12 | Successfully tagged sbseg-alo-mundo-scone-segredos:latest
13 |
14 | $ docker run -it --rm --device /dev/isgx sbseg-alo-mundo-scone-segredos
15 | Alo, mundo!
16 | UM_SEGREDO: None
17 | Traceback (most recent call last):
18 |   File "programa.py", line 4, in <module>
19 |     arquivo = open("/etc/segredo.txt", "r")
20 | FileNotFoundError: [Errno 2] No such file or directory: '/etc/segredo.txt'
21 |
22 | $ docker run -it --rm --device /dev/isgx \
23 |          -e SCONE_CAS_ADDR=$SCONE_CAS_ADDR \
24 |          -e SCONE_LAS_ADDR=$SCONE_LAS_ADDR \
25 |          -e SCONE_CONFIG_ID=sessao-exemplo-segredos/alo-mundo \
```

```
26 |          sbseg-alo-mundo-scone-segredos
27 | Alo, mundo!
28 | UM_SEGREDO: ^/Z/!Cm1D&Q84BP'
29 | isto eh um segredo!!!
```

**Figura 1.31. Criação, execução e obtenção de MRENCLAVE do *Alô, mundo!* no SCONE**

### 1.4.6.4. FSPF e volumes

Outra funcionalidade oferecida pelo SCONE é atestação e criptografia do sistema de arquivos, através de SCONE FSPF (*FileSystem Protection File*) e volumes. FSPF permite ao desenvolvedor criar regiões do sistema de arquivos que podem ser autenticadas e criptografadas. Uma das vantagens é que o gerenciamento de chaves e o processo de criptografia são feitos pelo *runtime* do SCONE, com o auxílio do CAS, o que torna a criptografia transparente para as aplicações. Por exemplo, quando uma aplicação SCONE tenta ler um arquivo numa região protegida por FSPF, o *runtime* intercepta a chamada de sistema de leitura do arquivo e, caso a aplicação tenha sido atestada, recebe as chaves do CAS que permitem a descriptografia do arquivo automaticamente dentro do enclave. Dessa forma, aplicações não precisam ser modificadas para lidar com criptografia. Essa funcionalidade é particularmente útil no caso de aplicações escritas em linguagens de programação interpretadas (Python, JavaScript), já que o código-fonte (e bibliotecas) pode ser criptografado, sendo lido apenas pelo interpretador autorizado.

Para começar a definir regiões protegidas por FSPF, é necessário criar o arquivo FSPF em si. Para isso, utiliza-se a interface de linha de comando do SCONE (*scone-cli*). Após a criação do arquivo FSPF, é possível definir regiões, que podem ser autenticadas (*authenticated*) ou criptografadas (*encrypted*). Regiões autenticadas têm a integridade do conteúdo verificada, o que permite detectar modificações não-autorizadas. Contudo, o conteúdo dos arquivos pode ser lido por aplicações fora do enclave. Regiões criptografadas têm seu conteúdo criptografado, então aplicações fora do enclaves não podem acessar seu conteúdo. Regiões criptografadas são também autenticadas.

Para incluir o FSPF no processo de atestação remota feito pelo CAS é necessário criptografar o arquivo de FSPF em si, também através da *scone-cli*, que então gera uma chave (*key*) e um código de autenticação de mensagem (*tag* ou MAC, do inglês *Message Authentication Code*), essenciais para acessar o arquivo de FSPF e, por consequência, as regiões e arquivos. Por fim, *key* e *tag* são adicionados no arquivo de sessão, permitindo que regiões e arquivos sejam acessados apenas por aplicações atestadas. Caso a região tenha persistência ativada (*--kernel* na criação da região), escritas efetuadas nessas regiões são automaticamente persistidas pelo CAS, e uma nova *tag* é gerada para o novo estado do sistema de arquivos. Caso a persistência esteja desativada (*--ephemeral*), modificações não serão persistidas. A criação de FSPF através da *scone-cli* pode ser usando um cliente instalado localmente na máquina ou através de uma imagem Docker.

```
01 | scone fspf create /fspf/volume.fspf
02 | scone fspf addr /fspf/volume.fspf / --not-protected --kernel /
03 | scone fspf addr /fspf/volume.fspf /app --encrypted --kernel /app
```

```
04 | scone fspf addf /fspf/volume.fspf /app /native-files /app
05 | scone fspf encrypt /fspf/volume.fspf > /native-files/keytag
```

**Figura 1.32. Trecho de Codigo - *fspf.sh*: Criação de FSPF e regiões criptografadas**

O arquivo da Figura 1.32 mostra uma sequência de comandos executados em um contêiner de *scone-cli* para a criação de um arquivo FSPF em */fspf/volume.fspf* (linha 1). A criação de regiões *(scone fspf addr*) e adição de arquivos (*scone fspf addf*), ilustradas nos comandos seguintes, sempre se referem a um arquivo de FSPF, que é o primeiro argumento desses comandos. Ao FSPF são adicionadas duas regiões: */* (linha 2) e */app* (linha 3). A região */*, a raiz do sistema de arquivos, não é protegida (*--not-protected*) por FSPF. A região */app*, por sua vez, é criptografada (*--encrypted*), ou seja, todos os arquivos adicionados a */app* serão criptografados automaticamente. Ambas as regiões são persistidas pelo SCONE (*--kernel*). A linha 4 mostra a adição de arquivos à região */app*. Os arquivos originais estão localizados em */native-files* e os arquivos criptografados resultantes serão adicionados ao diretório */app* do sistema de arquivos. Por fim, o arquivo de FSPF é criptografado, e a chave e *tag* necessárias para acessar o FSPF são escritas no arquivo */native-files/keytag*, podendo ser depois adicionadas a um arquivo de sessão SCONE.

A Figura 1.33 ilustra como executar o arquivo acima e, assim, criar o FSPF, através da imagem Docker de *scone-cli*. Os arquivos a serem criptografados estão no diretório *native-files* local, e são injetados no contêiner de *scone-cli* via volumes Docker (*-v native-files:/native-files*). Os arquivos criptografados serão salvos no diretório *encrypted-files* local (*-v encrypted-files:/encrypted-files*), criado antes de executar o contêiner. Por fim, o arquivo de FSPF, *volume.fspf* será persistido no diretório local (*-v $PWD:/fspf-file*). Os comandos de criação estão no arquivo *fspf.sh*, que será executado pelo contêiner de *scone-cli*. O arquivo *programa.py*, da aplicação *Alô, mundo!* será usado como exemplo.

```
01 | $ mkdir fspf native-files encrypted-files
02 | $ cp programa.py native-files/
03 | $ chmod +x fspf.sh
04 | $ cp fspf.sh fspf/
05 |
06 | $ docker run -it --rm --device /dev/isgx \
07 |          -v $PWD/fspf:/fspf \
08 |          -v $PWD/native-files:/native-files \
09 |          -v $PWD/encrypted-files:/app \
10 |          sconecuratedimages/sconecli:alpine3.7-scone4.2.1 \
11 |          bash -c /fspf/fspf.sh
12 |
13 | $ cat encrypted-files/programa.py
14 | ��ڻ�zoA!�^�
```

**Figura 1.33. Usando a imagem Docker de *scone-cli* para criação de FSPF com região criptografada**

Uma vez criado o FSPF, basta verificar a chave e *tag* em */native-files/keytag* e incluí-las no arquivo de sessão, através dos parâmetros *fspf_path*, *fspf_key* e *fspf_tag* (substituir *$SCONE_FSPF_KEY* e *$SCONE_FSPF_TAG*). O arquivo de sessão da Figura 1.34 ilustra uma sessão que considera FSPF para o *service alo-mundo*.

```
01 | name: sessao-exemplo-fspf
02 | version: 0.3
03 | services:
04 |   - name: alo-mundo
```

```
05           |                                                                mrenclaves:
[41f0117a3c62966b48ef6e2388b5fe7ff719b1f48abbf417e855fff0546a8e0d]
06 |     command: python3 /app/programa.py
07 |     image_name: alo-mundo
08 |     pwd: /
09 |     environment:
10 |       SCONE_MODE: hw
11 |       UM_SEGREDO: $$SCONE::segredo1$$
12 |     fspf_path: /fspf/volume.fspf
13 |     fspf_key: $SCONE_FSPF_KEY
14 |     fspf_tag: $SCONE_FSPF_TAG
15 | images:
16 |   - name: alo-mundo
17 |     injection_files:
18 |     - path: /etc/segredo.txt
19 |       content: $$SCONE::segredo2$$
20 | secrets:
21 |    - name: segredo1
22 |      kind: ascii
23 |      size: 16
24 |    - name: segredo2
25 |      kind: ascii
26 |      value: "isso eh um segredo"
27 | security:
28 |   attestation:
29 |     tolerate: [debug-mode, hyperthreading, outdated-tcb]
30 |     ignore_advisories: "*"
```

**Figura 1.34. Trecho de Código - *sessao-fspf.yml*: exemplo de arquivo de sessão SCONE com FSPF**

Como ilustrado na Figura 1.35, é necessário enviar o novo arquivo de sessão para o CAS, e reescrever a definição de imagem Docker, *scone.Dockerfile*, para considerar o novo arquivo Python criptografado, bem como o arquivo FSPF.

```
01 | FROM sconecuratedimages/public-apps:python-3.7.3-alpine3.10
02 | COPY encrypted-files/programa.py /app/programa.py
03 | COPY volume.fspf /fspf-file/volume.fspf
04 | ENTRYPOINT [ “python3”, “/app/programa.py” ]
```

**Figura 1.35. Trecho de Código - scone.Dockerfile: aplicação Alô, mundo! com FSPF e código criptografado**

A Figura 1.36 ilustra a construção e execução da nova imagem. Note que agora *SCONE_CONFIG_ID=sessao-exemplo-fspf/alo-mundo* e a tentativa de execução não atestada resulta em erro pois o interpretador Python não consegue compreender o arquivo *programa.py* criptografado. Uma vez atestado, o *runtime* do SCONE descriptografa o arquivo transparentemente, e a aplicação executa da forma esperada.

```
 01 | $ docker build . -t sbseg-alo-mundo-scone-fspf -f scone.Dockerfile
 02 | Sending build context to Docker daemon  23.55kB
 03 | Step 1/4 : FROM sconecuratedimages/public-apps:python-3.7.3-alpine3.10
 04 |  ---> 179f05bee7c7
 05 | Step 2/4 : COPY encrypted-files/programa.py /app/programa.py
 06 |  ---> e9af77581bbd
 07 | Step 3/4 : COPY fspf/volume.fspf /fspf/volume.fspf
 08 |  ---> 80b9d856016e
 09 | Step 4/4 : ENTRYPOINT [ "python3", "/app/programa.py" ]
 10 |  ---> Running in 709b8c8a65ec
 11 | Removing intermediate container 709b8c8a65ec
 12 |  ---> a24945984a80
 13 | Successfully built a24945984a80
 14 | Successfully tagged sbseg-alo-mundo-scone-fspf:latest
 15 |
 16 | $ docker run -it --rm --device /dev/isgx sbseg-alo-mundo-scone-fspf
 17 | File "/app/programa.py", line 1
 18 | SyntaxError: Non-UTF-8 code starting with '\xc1' in file /app/programa.py on
line
```

```
      1, but no encoding declared; see http://python.org/dev/peps/pep-0263/ for
details
 19 |
 20 | $ docker run -it --rm --device /dev/isgx \
 21 |          -e SCONE_CAS_ADDR=$SCONE_CAS_ADDR \
 22 |          -e SCONE_LAS_ADDR=$SCONE_LAS_ADDR \
 23 |          -e SCONE_CONFIG_ID=sessao-exemplo-fspf/alo-mundo \
 24 |          sbseg-alo-mundo-scone-fspf
 25 | Alo, mundo!
 26 | UM_SEGREDO: Nrj05qjgqHjDlYJd
 27 | isto é um segredo!!!
```

**Figura 1.36. Construção e execução da aplicação Alô, mundo! com código criptografado por SCONE FSPF**

Volumes[25] por sua vez, permitem definir regiões criptografadas do sistema de arquivos que podem ser montadas no sistema de arquivos de aplicações SCONE. Um volume (*volume*, no arquivo de sessão) pode ser compartilhado entre várias aplicações, ou mesmo importado por aplicações definidas em sessões diferentes. Neste caso, é possível controlar que tipo de modificações uma outra sessão pode realizar no sistema de arquivos. Volumes têm uma única região criptografada, e também podem ter uma chave e uma *tag* de FSPF definida pelo usuário. Caso uma chave e *tag* não sejam definidas, o volume é inicializado vazio, e o CAS gerencia as novas chave e *tag* de forma transparente. Para utilizar volumes, é necessário associá-los a uma imagem (*image*, no arquivo de sessão), que pode então ser referenciada pelas aplicações.

## 1.5. Computação Confidencial nativa da Nuvem com Intel SGX

Nesta seção, discutimos como combinar computação confidencial com computação nativa da nuvem para nosso caso de uso. Para a implementação dos componentes, detalharemos a criação de um produtor baseado no SGX SDK e de um processador (consumidor e produtor) baseado no SCONE. Outros exemplos, como um consumidor SDK, além de produtores e consumidores SCONE, estão disponíveis no repositório do mini-curso [LSD, 2020].

### 1.5.1. Construindo clientes Kafka confidenciais utilizando SGX SDK

Conforme descrito na Seção 1.4, o desenvolvimento de aplicações que utilizam SGX SDK deve ser feito utilizando a linguagem C/C++, particionadas em uma parte confiável e uma parte não confiável. Além disso, uma vez que o enclave não consegue executar chamadas de sistema o desenvolvedor deve gerenciar a execução de atividades de persistência e comunicação que circularam pela porção não confiável, de modo que não sejam vazadas informações potencialmente sensíveis.

Também consequência da impossibilidade da execução de chamadas de sistema pela parte confiável, é a impossibilidade de ligação de várias bibliotecas populares de C/C++ contra enclaves, como ASIO, Boost, librdkafka, RapidJSON, entre outras. Para utilizar tais bibliotecas, uma alternativa é ligá-las contra a parte insegura e utilizá-las através de *ocalls*, porém, não há garantias em relação à segurança das informações que são transacionadas através de *ocalls*.

Neste exemplo utilizaremos a biblioteca *librdkafka*[26] para implementação de um cliente produtor Kafka. O gerenciamento de chaves criptográficas bem como os

[25] https://sconedocs.github.io/CAS_session_lang_0_3
[26] https://github.com/edenhill/librdkafka

processos de criptografia e descriptografia de mensagens são feitos dentro do enclave, de forma que nenhuma informação confidencial é vazada em texto plano. Uma alternativa à esta biblioteca é a implementação do suporte ao protocolo do Kafka dentro do enclave, que é uma tarefa complexa. Outra biblioteca utilizada é a JSMN, para fazer a análise de strings JSON. Esta, por sua vez, é ligada diretamente contra o enclave, uma vez que ela não depende de chamadas de sistema.

Para a garantir segurança dos dados, a gerência do processo de atestação é feita pelo enclave, assim como a descriptografia dos dados simulados e a nova criptografia com a chave do Kafka. A porção insegura recebe então apenas dados opacos que ela encapsula e envia para a rede. Por simplicidade, utilizamos somente uma chave de criptografia e um tópico, mas a solução poderia ser facilmente estendida para sistemas mais complexos (considerando também os pontos a serem discutidos na Seção 1.6).

### 1.5.1.1. Gerenciador de Segredos

O gerenciador de segredos é o componente responsável por armazenar e entregar chaves criptográficas para os demais componentes da aplicação. Escrito aqui em Python, assumimos que este componente executa em uma máquina de confiança do usuário.

A entrega de segredos é realizada mediante atestação remota, ou seja, as aplicações são submetidas ao processo de atestação remota para verificação de suas identidades, conforme discutido na Seção 1.4.2, para que então os segredos sejam entregues. No exemplo desenvolvido aqui, o gerenciador de segredos identifica as aplicações através dos seus respectivos valores de MRENCLAVE e para cada um estão associados segredos diferentes. Esses segredos são carregados através de um arquivo de configuração que utiliza o formato JSON, conforme exposto na Figura 1.37.

```
01 | {
02 |  "app-list": [
03 |   {
04 |    "app-name": "consumer",
05 |    "mrenclave": "164f40a42ab6908cde5183e9f6e662a408d0c31a684afedf81b2506b62b32979",
06 |    "secrets": {
07 |     "KAFKA_SECRET": "4145533132384b65792d313643686172"
08 |    }
09 |   },
10 |   {
11 |    "app-name": "producer",
12 |    "mrenclave": "0c0ccf1e24287cfa50d6a85f7652afbfd482761c7d89c11124540a528a690984",
13 |    "secrets": {
14 |     "KAFKA_SECRET": "4145533132384b65792d313643686172",
15 |     "PAYLOAD_SECRET": "b588fbd73e704331df95764ed85faf78"
16 |    }
17 |   }
18 |  ]
19 | }
```

**Figura 1.37 - Exemplo de arquivo de configuração com segredos**

Na Figura 1.37 podemos identificar duas entradas de aplicações em *app-list*: *consumer* e *producer*. Para cada entrada nós temos um nome (*app-name*), utilizado apenas para identificar a aplicação; um MRENCLAVE (*mrenclave*) que é utilizado para checagem da identidade da aplicação; e por fim, os segredos (*secrets*), que são as chaves criptográficas utilizadas para criptografar e descriptografar os conteúdos das mensagens dentro dos enclaves.

A comunicação entre o sistema gerenciador de segredos e as aplicações utilizam *socket* em sua comunicação, protegida através do uso de primitivas criptográficas como CMAC (do inglês, *Cipher-Based Message Authentication Code*) para autenticação, e criptografia utilizando chaves simétricas derivadas a partir de um processo de troca de chaves de Diffie–Hellman.

Durante a troca de mensagens, verificações são realizadas para identificar possíveis adulterações que possam ser realizadas por atacantes no processo de atestação remota. A Figura 1.38 apresenta o processo de verificação do *quote*, recebido pelo sistema gerenciador de segredos, ao longo do processo de atestação.

```
085 |     gid = quote[0x4:0x8]
086 |     if gid != acontext.get_sp_gid():
087 |         print("FAIL: GID on quote is different
                        of initial GID. Aborting...")
088 |         return False
089 |
090 |     derived_key_smk = acontext.get_derived_key("SMK")
091 |     calc_mac = aes128_cmac(derived_key_smk, g_a +
                                          ps_sec_prop_desc + quote)
092 |
093 |     if calc_mac != mac:
094 |         print("FAIL: MAC on MSG3 is different of
                      computed locally. Aborting...")
095 |         return False
096 |
097 |     quote_report_data = quote[QUOTE_REPORT_DATA_OFFSET:
098 |                               QUOTE_REPORT_DATA_OFFSET +
                                  QUOTE_REPORT_DATA_LENGTH]
099 |
100 |     derived_key_vk = acontext.get_derived_key("VK")
101 |     calc_report_data = sha256_hash(acontext.get_sp_public_numbers()['x']+
                                        acontext.get_sp_public_numbers()['y']+
102 |                                     acontext.get_public_numbers()['x']+
                                        acontext.get_public_numbers()['y']+
103 |                                     derived_key_vk, False)+
                                        binascii.unhexlify(32 * b'00')
104 |
105 |     if quote_report_data != calc_report_data:
106 |         print("FAIL: [#%d] Quote report data on MSG3 is different of the
                     computed locally. Aborting...", acontext.get_id())
107 |         return False
```

**Figura 1.38 - Processo de verificação do *quote* recebido da aplicação**

Na Figura 1.38, campos do *quote* são verificados com o objetivo de verificar sua integridade, uma vez que a computação utilizada para calcular o CMAC e *hash* desses campos utilizam chaves derivadas do processo de troca de chaves de Diffie-Hellman. Ao final do processo de atestação, os segredos são criptografados utilizando uma das chaves simétricas derivadas, e enviados para a aplicação cliente, conforme visto na função *send_secrets()* na Figura 1.39[27].

```
056 | def send_secrets(client_socket, address, acontext, mrenclave):
057 |     global secrets
058 |
059 |     derived_key_sk = acontext.get_derived_key("SK")
060 |
061 |     msg_type = struct.pack('<B', MSG_TYPE_SECRETS)
062 |     payload = aes_encrypt(json.dumps(secrets[mrenclave]), derived_key_sk)
063 |     payload_len = struct.pack('<I', len(payload))
064 |
```

[27] A implementação completa do sistema de gerenciamento de segredos está disponível em: https://git.lsd.ufcg.edu.br/lsd-sbseg-2020/kafka-sample-sgxsdk/tree/master/attestor

```
065 |     msg_secrets = msg_type + payload_len + payload
066 |
067 |     try:
068 |         client_socket.send(msg_secrets)
069 |         print("INFO: [#%d] Encrypted secrets sent!" % acontext.get_id())
070 |     except:
071 |         print("FAIL: [#%d] Unable to send secrets!" % acontext.get_id())
```

**Figura 1.39 - Código de criptografia e envio de segredos**

### 1.5.1.2. Produtor Kafka

Para implementar o produtor Kafka utilizando o SGX SDK, foi utilizada para a comunicação com o Apache Kafka a biblioteca *librdkafka*. Essa aplicação simula um gateway Kafka para medições de energia produzidos por um *smart meter*. Essas medições se encontram criptografadas no arquivo *smartmeter-data.enc*, e a chave para descriptografar as informações é recebida do sistema gerenciador de segredos durante o processo de atestação.

```
095 | static int initialize_kafka_producer() {
096 |     char errstr[512];
097 |
098 |     run = 1;
099 |     conf = rd_kafka_conf_new();
100 |
101 |     if (rd_kafka_conf_set(conf, "bootstrap.servers", broker,
102 |                           errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
103 |         fprintf(stderr, "%s\n", errstr);
104 |         rd_kafka_conf_destroy(conf);
105 |         return 0;
106 |     }
107 |
108 |     rd_kafka_conf_set_dr_msg_cb(conf, dr_msg_cb);
109 |
110 |     rk = rd_kafka_new(RD_KAFKA_PRODUCER, conf, errstr, sizeof(errstr));
111 |     if (!rk) {
112 |         fprintf(stderr,
113 |                 "%% Failed to create new producer: %s\n", errstr);
114 |         return 0;
115 |     }
116 |
117 |     signal(SIGINT, stop);
118 |
119 |     return 1;
120 | }
```

**Figura 1.40 - Código de inicialização do produtor Kafka**

A Figura 1.40 apresenta o código utilizado para inicializar um produtor Kafka utilizando a *librdkafka*. Esse produtor é inicializado na parte não confiável da aplicação, consequentemente informações relacionadas aos metadados das mensagens e detalhes sobre o cluster podem ser obtidos por alguém com acesso privilegiado ao sistema. A recuperação das chaves criptográficas pelo lado seguro é disparada a partir de uma chamada de *ecall* para a parte confiável, como ilustrado na Figura 1.41 e as chaves recuperadas nunca deixam o enclave.

```
283 |         fprintf(stderr, "INFO: Getting secrets from attestation service
                    '%s:%d'...\n", attestation_host, attestation_port);
284 |         if (SGX_SUCCESS != ecall_load_keys_from_server(global_eid,
                &ecall_ret, attestation_port, attestation_host,
                strlen(attestation_host) + 1) || ecall_ret < 0) {
285 |             fprintf(stderr, "FAIL: Unable to get keys from attestation
                                service! Exiting...\n");
286 |
287 |             sgx_destroy_enclave(global_eid);
288 |
```

```
289 |             return EXIT_FAILURE;
```

**Figura 1.41 - Chamada de *ecall* para obtenção das chaves a partir do gerenciador de segredos**

A função *ecall_load_keys_from_server()*, por sua vez, executa a função *do_remote_attestation()* dentro do enclave, que realiza o processo de atestação remota, e consequentemente obtenção das chaves criptográficas.

```
320 | int do_remote_attestation(int ra_port, char *ra_host,
                                uint8_t p_payload_key[SGX_AESGCM_KEY_SIZE],
                                uint8_t p_kafka_key[SGX_AESGCM_KEY_SIZE]) {
321 |     int sock_fd, ret, ra_ret = 0;
322 |     ssize_t recv_ret;
323 |     sgx_status_t ocall_ret;
324 |
325 |     sgx_ra_context_t ra_ctx;
326 |     sgx_ec256_public_t pubkey;
327 |
328 |     uint8_t recv_msg_type;
329 |
330 |     ocall_ret = ocall_create_connection(&sock_fd, ra_port, ra_host);
331 |     if (ocall_ret != SGX_SUCCESS) {
332 |         print_string("ENCL: FAIL: Unable to call
                           'ocall_create_connection'...\n");
333 |
334 |         return RA_ERROR;
335 |     }
336 |
337 |     if (sock_fd < 0) {
338 |         print_string("ENCL: FAIL: Cannot connect to host: [%s:%d]\n", ra_host,
                            ra_port);
339 |
340 |         return RA_ERROR;
341 |     }
342 |
343 |     if (send_init_ra(sock_fd) < 0) {
344 |         print_string("ENCL: FAIL: Unable to send INIT_RA message.
Exiting...\n");
345 |
346 |         ocall_close_connection(&ret, sock_fd);
347 |         return RA_ERROR;
348 |     }
349 |
350 |     do {
351 |         ocall_ret = ocall_recv(&recv_ret, sock_fd,
                                   (char *)&recv_msg_type,
                                   sizeof(uint8_t));
352 |
353 |         if (SGX_SUCCESS != ocall_ret || recv_ret != sizeof(uint8_t)) {
354 |             print_string("ENCL: FAIL: Unable to receive message.
Exiting...\n");
355 |             ra_ret = RA_ERROR;
356 |             break;
357 |         }
358 |
359 |         switch (recv_msg_type) {
360 |             case MSG_TYPE_MSG0:
361 |                 ra_ret = handle_ra_msg0(sock_fd, &ra_ctx, &pubkey);
362 |                 break;
363 |
364 |             case MSG_TYPE_MSG2:
365 |                 ra_ret = handle_ra_msg2(sock_fd, &ra_ctx);
366 |                 break;
367 |
368 |             case MSG_TYPE_SECRETS:
369 |                 ra_ret = handle_ra_secrets(sock_fd, &ra_ctx,
                                                p_payload_key, p_kafka_key);
370 |                 break;
371 |         }
372 |     } while (ra_ret == RA_CONTINUE);
373 |
```

```
374 |     ocall_close_connection(&ret, sock_fd);
375 |
376 |     return ra_ret;
377 | }
```

**Figura 1.42 - Função que realiza o processo de atestação remota**

No código apresentado na Figura 1.42 podemos observar que o código realiza o processo de atestação remota através do uso de funções que processam as mensagens recebidas de acordo com o seu tipo (*MSG0*, *MSG2* e *MSG_SECRETS*). As mensagens do tipo *MSG0* e *MSG2* representam as mensagens trocadas no processo de atestação remota, conforme apresentado na Seção 1.4.2, enquanto a *MSG_SECRETS* é o tipo de mensagem que transporta os segredos, e só é enviada pelo gerenciador de segredos em caso de sucesso no processo de atestação.

Após o processo de atestação remota, a aplicação recebe duas chaves: *PAYLOAD_SECRET* e *KAFKA_SECRET*. A chave *PAYLOAD_SECRET* é utilizada para descriptografar dentro do enclave a informação lida do arquivo de medições criptografadas, enquanto a chave *KAFKA_SECRET* é utilizada para criptografar as mensagens que vão ser enviadas para o tópico no barramento de dados.

Para cada medição criptografada, é executada a *ecall ecall_reencrypt_message()* pela parte não confiável, que recebe o conteúdo criptografado e, dentro do enclave, o descriptografa com a chave *PAYLOAD_SECRET* e criptografa utilizando a chave *KAFKA_SECRET*, fazendo a publicação no Kafka utilizando a *ocall ocall_produce_message()*[28].

### 1.5.1.3. Execução e testes com Kubernetes

Para executar a aplicações *producer* com Kubernetes precisamos construir as imagens, utilizando os *Dockerfiles* dentro dos diretórios correspondentes a cada aplicação. Para facilitar o processo de criação das imagens, disponibilizamos dois *scripts* chamados *build-images-hw.sh* e *build-images-sim.sh*[29]. Quando executado, o primeiro *script* gerará imagens prontas para executar as aplicações utilizando SGX, já o segundo gerará imagens que executam em modo simulado, dispensando o uso do *driver* SGX, mas sem nenhuma garantia de proteção de memória. Com as images *Docker* construídas, podemos criar os arquivos YAML que descrevem como será a implantação de cada aplicação no Kubernetes, para cada aplicação.

Primeiramente, é importante notar que, o attester tipicamente executaria fora da nuvem, em uma máquina de controle e confiança do usuário que implanta a aplicação. Além disso, em um modo de testes, ele poderia ser configurado para não consultar a Intel para validação (definindo a variável de ambiente *NO_IAS*). Caso contrário, para validar os quotes com o IAS, é preciso realizar a inscrição no serviço de atestação da Intel e obter um SPID e uma *subscription key*. Isso pode ser feito através da página do

[28] A implementação completa do produtor Kafka está disponível em https://git.lsd.ufcg.edu.br/lsd-sbseg-2020/kafka-sample-sgxsdk/tree/master/producer
[29] https://git.lsd.ufcg.edu.br/lsd-sbseg-2020/kafka-sample-sgxsdk

serviço de atestação[30], seguindo as instruções de cadastro. Além disso, há uma documentação da API disponível[31].

A Figura 1.43 mostra o arquivo YAML para a implantação do *producer*.

```
01 | apiVersion: apps/v1
02 | kind: Deployment
03 | metadata:
04 |   name: producer-sdk
05 | spec:
06 |   selector:
07 |     matchLabels:
08 |       run: producer-sdk
09 |   replicas: 1
10 |   template:
11 |     metadata:
12 |       labels:
13 |         run: producer-sdk
14 |     spec:
15 |       containers:
16 |       - name: producer-sdk
17 |         args: ["./producer", "10.11.19.115:9092", "sbsegtopic",
                    "./input/smartmeter-data.enc"]
18 |         image: lsd-sbseg-2020/producer:hw
19 |         imagePullPolicy: Always
20 |         env:
21 |           - name: "ATTESTATION_SERVICE"
22 |             value: "attestor:10719"
23 |         volumeMounts:
24 |         - mountPath: /dev/isgx
25 |           name: dev-isgx
26 |         securityContext:
27 |           privileged: true
28 |       volumes:
29 |       - name: dev-isgx
30 |         hostPath:
31 |           path: /dev/isgx
```

**Figura 1.43 - Implantação do *producer* no Kubernetes: arquivo YAML de implantação**

O exemplo mostra a implantação do *producer* com SGX habilitado. Na linha 17, temos os argumentos que são o endereço do Kafka, o nome do tópico onde serão inseridas as mensagens produzidas e o arquivo que contém o conteúdo que será transformado em mensagens Kafka. Além desses argumentos, a variável de ambiente *ATTESTATION_SERVICE* indica o endereço do *attester*, onde o *producer* buscará pelas chaves de criptografia necessárias para ler os dados criptografados, e para produzir as mensagens. As chaves são entregues após o processo de atestação bem sucedido.

De posse dos arquivos de implementação das aplicações e dos segredos do *attestor*, podemos realizar a implantação de fato, usando o comando *kubectl* do Kubernetes (para a versão MicroK8s, a mais simples de ser instalada, o comando seria *microk8s kubectl apply -f producer-deployment.yaml*). Depois da execução dos comandos de implantação, podemos verificar o estado das aplicações que acabamos de implantar com o comando *kubectl get pods*.

### 1.5.2. Construindo clientes Kafka confidenciais em Python com SCONE

Como discutido anteriormente, com aplicações confidenciais construídas com SCONE as porções inseguras são geridas pelo SCONE e o código do usuário é todo inserido na parte segura. Desta maneira, nessa seção mostraremos a implementação de uma

[30] https://api.portal.trustedservices.intel.com/EPID-attestation
[31] https://api.trustedservices.intel.com/documents/sgx-attestation-api-spec.pdf

aplicação simples que se comunica com um servidor Apache Kafka, consumindo e produzindo eventos, e que tira proveito da flexibilidade oferecida pelos contêineres seguros SCONE para se tornarem confidenciais.

#### 1.5.2.1. Construindo um Processador de dados Kafka em Python

Apresentamos agora um código para processar os dados consumidos de um tópico e tomar uma decisão com base no conteúdo desses dados[32]. No exemplo, o processador será responsável por detectar se existem anomalias nas medições coletadas por sensores inteligentes de energia e retroalimentar o Kafka em um tópico chamado *alarms* com o tempo em que a anomalia ocorreu.

```
01 | import sys, getopt, os
02 | from json import loads
03 | from time import sleep
04 | from kafka import KafkaConsumer, KafkaProducer
05 | from kafka.errors import KafkaError
06 | from cryptography.hazmat.backends import default_backend
07 | from cryptography.hazmat.primitives.ciphers import (
08 |     Cipher, algorithms, modes
09 | )
10 |
11 | KEY = os.getenv('KAFKA_KEY').encode('ascii')
12 | NONCE_BYTE_SIZE = 12
```

**Figura 1.44. Trecho de Código - Carregamento de módulos (Processador)**

O processador possui características tanto de consumidor quanto de produtor, assim como mostrado na Figura 1.44. A função *decrypt* (Figura 1.45) recebe como argumentos a chave para decifragem, os dados de autenticação de mensagem, o *nonce*, o texto cifrado e a *tag*, todas essas variáveis são necessárias no algoritmo AES-GCM. Como resultado da função temos os dados originais de energia que foram enviados ao servidor (linha 22). A função *consume*, descrita na Figura 1.46, se conecta com o servidor Kafka informado no parâmetro *server*, a conexão com o servidor passa por uma etapa de autenticação SSL. Observe ainda que o consumidor é configurado sem *enable_auto_commit*, ou seja, receber mensagens através desse consumidor não reflete na marcação mantida no Zookeeper. Assim, toda vez que esse consumidor é iniciado ele começa a consumir mensagens a partir do mesmo ponto. Esse consumo se dá de acordo com o valor escolhido no parâmetro *auto_offset_reset* que definimos como *latest*, ou seja, o evento mais recente.

```
14 | def decrypt(key, associated_data, nonce, ciphertext, tag):
15 |     decryptor = Cipher(
16 |         algorithms.AES(key),
17 |         modes.GCM(nonce, tag),
18 |     ).decryptor()
19 |
20 |    if associated_data is not None:
20 |         decryptor.authenticate_additional_data(associated_data)
21 |
22 |     return decryptor.update(ciphertext) + decryptor.finalize()
```

**Figura 1.45. Trecho de Código - Função para decifrar**

```
24 | def consume(server, topic):
25 |     consumer = KafkaConsumer(topic,
26 |                              security_protocol='SSL',
```

[32] Código completo do processador disponível em: https://git.lsd.ufcg.edu.br/lsd-sbseg-2020/kafka-sample-scone/blob/master/src/feedback.py

```
27 |                              ssl_check_hostname=False,
28 |                              ssl_cafile='/certs/CARoot.pem',
29 |                              bootstrap_servers=[server],
30 |                              auto_offset_reset='latest',
31 |                              enable_auto_commit=False)
32 |     for message in consumer:
33 |         encoded_message = message.value
34 |
35 |          nonce = encoded_message[:12]
36 |          tag = encoded_message[12:28]
37 |          cipher = encoded_message[28:]
38 |
39 |          text = decrypt(KEY, None, nonce, cipher, tag).decode('ascii')
40 |          return text
```

**Figura 1.46. Trecho de Código - Função de consumo com retorno**

Para produção de avisos de alerta, criamos a função *produce_warning* (Figura 1.47), que é similar à função *produce* utilizada no produtor, mas que recebe como parâmetro, além do endereço do servidor Kafka, a variável *message*, que representa o conteúdo da mensagem a ser publicada no tópico *alarm*, linhas 42 e 45.

```
42 | def produce_warning(server, message):
43 |     producer = KafkaProducer(bootstrap_servers=[server],
                                 security_protocol='SSL',
                                 ssl_check_hostname=False,
                                 ssl_cafile='/certs/CARoot.pem')
44 |
45 |     future = producer.send("alarm", value=b'[alarm] - ' +
message.encode('ascii'))
46 |     try:
47 |         record_metadata = future.get(timeout=5)
48 |     except KafkaError as e:
49 |         print(e)
50 |         sys.exit(1)
```

**Figura 1.47. Trecho de Código - Função de produção de alertas**

A lógica de detecção de anomalias nos dados de energia fica então no código executado como módulo principal (Figura 1.48). Note que entre as linhas 69 e 75 temos um laço, que consome uma métrica e a codifica como JSON (linha 70), e faz uma avaliação da coluna *V*, que representa a tensão da rede. Definimos para este exemplo que uma anomalia ocorre quando o valor de *V* é inferior a 210. A linha 71 é responsável por verificar se uma anomalia ocorreu e, caso ocorra, a função *produce_warning* é chamada e o tempo em que a métrica foi coletada, coluna *timestamp*, é enviado como conteúdo da mensagem de alarme.

```
52 | if __name__ == "__main__":
53 |     argv = sys.argv[1:]
54 |
55 |     try:
56 |         opts, args = getopt.getopt(argv,"hs:t:",["server=","topic="])
57 |     except getopt.GetoptError:
58 |         print('feedback.py -s <server:port> -t topic')
59 |         sys.exit(2)
60 |     for opt, arg in opts:
61 |         if opt == '-h':
62 |             print('feedback.py -s <server:port> -t topic')
63 |             sys.exit()
64 |         elif opt in ("-s", "--server"):
65 |             server = arg
66 |         elif opt in ("-t", "--topic"):
67 |             topic = arg
68 |
69 |     while True:
70 |         metric = loads(consume(server, topic))
```

```
71 |         if float(metric['V']) <= 210.00:
72 |             produce_warning(server, str(metric['timestamp']))
73 |             print("alarm sent")
74 |
75 |         sleep(5)
```

**Figura 1.48. Trecho de Código - Checagem de alertas**

### 1.5.2.2. Adicionando confiabilidade às soluções utilizando SCONE

As aplicações Python apresentadas nas seções anteriores ainda não possuem as garantias de confiabilidade que ambientes de execução confiável proporcionam. Para que possamos executá-las dentro de enclaves protegidos Intel SGX, podemos utilizar contêineres que contenham uma versão protegida do interpretador Python (ou seja, o interpretador foi compilado com o próprio SCONE).

Como base de construção, utilizamos a imagem curada que contém um interpretador Python-3.7 seguro[33], que está disponível para acesso público. Esse interpretador utiliza da versão modificada da *musl-libc* para executar de forma transparente dentro do enclave SGX. Como detalhado no *Dockerfile* representado na Figura 1.49, adicionamos a essa imagem base as bibliotecas *libffi* e *openssl*, que são necessárias para funções de criptografia (linha 2). Copiamos então o código-fonte da aplicação para uma pasta no contêiner chamada */app* e também o arquivo *requirements.txt*, que descreve quais módulos Python a aplicação utiliza (linha 4). Na linha 6 os módulos são instalados através da ferramenta *pip*. A imagem desse contêiner pode então ser gerada com o comando: *docker build . -t sbseg:latest*

```
01 | FROM sconecuratedimages/kubernetes:python-3.7.3-alpine3.10-scone4.2
02 | RUN apk update && apk add gcc g++ libffi-dev openssl-dev
03 | COPY ./feedback.py /app
04 | COPY requirements.txt /
05 | WORKDIR /app
06 | RUN pip install -r /requirements.txt
07 | ENTRYPOINT python3
```

**Figura 1.47. Trecho de Código - Arquivo *Dockerfile***

Além da imagem Python segura, precisamos garantir a integridade (ou até a confidencialidade do código). Como discutimos na Seção 1.4.6, Python é uma linguagem interpretada, sendo assim é o interpretador que está sendo verificado quando há uma execução. Ou seja, isso significa que o *hash* ou MRENCLAVE das aplicações é o do próprio interpretador, e ele então lê o código depois de carregado. Assim, as aplicações Python estão sujeitas a ataques contra sua integridade, mesmo executando dentro de enclaves protegidos. Para remediar este problema precisamos proteger o sistema de arquivos da imagem, que contém o código-fonte e as dependências. Como visto na Seção 1.4.6, isso pode ser feito de forma transparente. Usamos então o mesmo script da Figura 1.32, que cria, utilizando FSPF, uma região criptografada no diretório */app*, onde o código-fonte estará armazenado.

Utilizando do script *fspf.sh* e se beneficiando da solução de *multi-stage building* do Docker, cria-se então um novo contêiner, que consiste no encapsulamento do criado na seção anterior e seu Dockerfile é mostrado na Figura 1.48. Usando como base a imagem *sbseg:latest* executamos o *fspf.sh* (linha 7), assim criando o diretório protegido

[33] *Tag* no Docker Hub: sconecuratedimages/kubernetes:python-3.7.3-alpine3.10-scone4.2

*/app* na imagem de apelido *fspf*, dessa imagem copiamos o arquivo *fspf.pb* e o conteúdo de */app* para um diretório chamado *app* para a imagem final (linhas 9 e 10) e definimos o valor da variável *SCONE_CONFIG_ID*, que será utilizada pelo SCONE na fase de atestação remota.

```
01 | FROM sbseg:latest as fspf
02 | COPY fspf.sh /
03 | RUN mkdir /app
04 | WORKDIR /
05 | RUN SCONE_NO_FS_SHIELD=1 SCONE_MODE=sim /fspf.sh
06 | FROM sbseg:latest
07 | COPY --from=fspf /fspf.pb /
08 | COPY --from=fspf /app /app
09 | ENV SCONE_CONFIG_ID=python_kafka/python_kafka_producer
```

**Figura 1.48. Trecho de Código - Arquivo *Dockerfile.fspf***

A imagem desse contêiner pode então ser gerada com o comando: *docker build . -t sbseg:fspf -f Dockerfile.fspf* e no arquivo *keytag* estarão os valores da chave e *tag* do arquivo *fspf.pb*, necessários para decifrá-lo.

O último passo antes de podermos executar o contêiner *sbseg:fspf* é submeter o arquivo de sessão da nossa aplicação para o CAS. A Figura 1.49 mostra o arquivo de sessão que é utilizado para registrar o serviço Kafka criado na seção anterior. Note que na linha 4 é definido o nome do serviço. O CAS é responsável por guardar os segredos das aplicações, no caso descrito os segredos são a chave de cifragem, definida na linha 10 como variável de ambiente e o certificado para autenticação do servidor Kafka contido nas linhas 17-39 como arquivo injetado. Note também que a chave e a *tag* do arquivo *fspf.pb* obtidas durante a criação do FSPF são informadas nas linhas 11-13.

```
 01 | name: python_kafka
 02 | version: "0.3"
 03 | services:
 04 |   - name: python_kafka_feedback
 05 |     image_name: python_kafka
 06 |     command: python3 -u feedback.py -s 127.0.0.1:9092 -t sbsegtopic
 07 |     mrenclaves:
[67b8017f7083435cb614b87c8daa14303f741a10a2a0bbf5dfabec777cf629b9]
 08 |     pwd: /src
 09 |     environment:
 10 |       KAFKA_KEY: "AES128Key-16Char"
 11 |     fspf_path: /fspf.pb
 12 |     fspf_key: {{key}}
 13 |     fspf_tag: {{tag}}
 14 | images:
 15 |   - name: python_kafka
 16 |     injection_files:
 17 |        - path: /certs/CARoot.pem
 18 |          content: |
 19 |             -----BEGIN CERTIFICATE-----
 20 |             MIIDazCCAlOgAwIBAgIUIj/cO1QmJhmzVDiGI07lMEErwXEwDQYJKoZIhvcNAQEL
 .. |             ...
 38 |             3jlCpqlCl/KzUVYzh5wH
 39 |             -----END CERTIFICATE-----
 40 | security:
 41 |   attestation:
 42 |     tolerate: [debug-mode, hyperthreading, outdated-tcb]
 43 |     ignore_advisories: "*"
```

**Figura 1.49. Trecho de Código - Arquivo *session.yml***

#### 1.5.2.5. Execução e testes com Kubernetes

Podemos testar o consumo dos dados publicados usando também o consumidor simples provido pelo próprio Kafka, dessa forma é possível ver que os dados que foram publicados estão cifrados. Para esse teste simples executaremos um *Pod* consumidor em um *cluster* Kubernetes. O arquivo mostrado na Figura 1.50 mostra um exemplo de manifesto para criação de um *Pod* consumidor. A imagem utilizada é definida na linha 10, e nas linhas 11 e 12 definimos o comando de execução do consumidor e seus parâmetros: endereço do servidor Kafka e tópico. As linhas 13-23 contém as variáveis de ambiente SCONE.

```
01 | apiVersion: v1
02 | kind: Pod
03 | metadata:
04 |   name: consume-demo
05 |   labels:
06 |     purpose: demonstrate-consumer
07 | spec:
08 |   containers:
09 |   - name: console-consumer
10 |     image: lucasmc/sbseg:fspf
11 |     command: ["python3"]
  12  |          args:  ["/src/kafkaconsumer.py",  "-s",  "10.30.0.51:9092",  "-t",
"sbsegtopic"]
13 |     env:
14 |       - name: "SCONE_CAS_ADDR"
15 |         value: "4-2-1.scone-cas.cf"
16 |       - name: "SCONE_LAS_ADDR"
17 |         value: "10.30.0.51"
18 |       - name: "SCONE_VERSION"
19 |         value: "1"
20 |       - name: "SCONE_LOG"
21 |         value: "7"
22 |       - name: "SCONE_CONFIG_ID"
23 |         value: "kafka_python/python_kafka_consumer"
24 |     volumeMounts:
25 |       - mountPath: /dev/isgx
26 |         name: dev-isgx
27 |     securityContext:
28 |       privileged: true
29 |   volumes:
30 |   - name: dev-isgx
31 |     hostPath:
32 |       path: /dev/isgx
33 |   restartPolicy: OnFailure
```

**Figura 1.50. Trecho de Código - Arquivo *consumer.yaml***

Este manifesto pode então ser executado no Kubernetes (no caso do MicroK8s: *microk8s kubectl apply -f consumer.yaml)* e para observar os valores de saída basta olhar o registro de saída do *Pod* em questão (no caso do MicroK8s: *microk8s kubectl logs -f console-consumer*). O registro de saída do *Pod* mostrará as mensagens criptografadas que foram enviadas ao Kafka.

## 1.6. Boas práticas e direções promissoras

Esta seção complementa o material apresentado com sugestões de material complementar e direções de pesquisa e de refinamento dos exemplos expostos aqui.

### 1.6.1. Implantação de *software* nativo da nuvem

O extenso ecossistema de computação nativa da nuvem oferece uma série de soluções e ferramentas que podem ser facilmente integradas a novas aplicações nativas da nuvem,

promovendo a portabilidade e o desacoplamento. Helm[34], por exemplo, é um instalador de aplicações nativas da nuvem. Atuando de forma similar à de um gerenciador de pacotes de sistema operacional, *Helm* permite implantar aplicações em *clusters* Kubernetes de forma rápida. As aplicações são definidas em artefatos chamados *charts*, que são agrupados em repositórios. Para um operador de nuvem, basta adicionar determinado repositório e utilizar a interface de linha de comando do *Helm* para instalar, atualizar (com versionamento e possibilidade de recuperação em caso de falha) e gerenciar aplicações nativas da nuvem. É possível personalizar determinados parâmetros através de um arquivo YAML, o que facilita o gerenciamento de diferentes cenários de implantação (por exemplo, cenários como produção, desenvolvimento e teste).

Helm provê um repositório canônico de *charts* de aplicações populares. SCONE também oferece um repositório com versões seguras de algumas aplicações populares, dentre elas MariaDB, Kafka e PySpark, que podem ser instaladas em *clusters* Kubernetes que suportam computação confidencial (como Azure Kubernetes Services [Microsoft, 2020]). A utilização de serviços Kubernetes é uma alternativa mais prática em relação à configuração de máquinas virtuais, informações sobre a configuração de clusters podem ser encontrados nos apêndices disponíveis em [LSD, 2020].

### 1.6.2. Abordagens complementares de segurança

No caso de uso desenvolvido ao longo do capítulo, a utilização de mecanismos de computação confidencial permite que o desenvolvedor não precise confiar na integridade da infraestrutura (ex., *firmware*, sistema operacional, hipervisor). Isso é possível uma vez que apenas o seu código conseguirá acesso às chaves de criptografia dos dados e que uma vez obtidas, essas chaves não podem ser inspecionados pelo software privilegiado na infraestrutura. No entanto, a possibilidade de que algum componente de segurança apresente vulnerabilidades não deve ser descartada e é conhecido como segurança em camadas: caso uma das camadas se mostre mais frágil que o esperado, as outras ainda oferecerão proteção.

No caso em mãos, existem algumas alternativas que podem ser consideradas: (1) assumir um modelo de zero confiança (do inglês, *zero trust*); (2) atestação do carregamento do sistema operacional onde os enclaves estão executando; (3) configuração de mecanismos adicionais de controle de acesso.

A adoção de um modelo de zero confiança tem dois aspectos principais. Em primeiro lugar, pode-se assumir zero confiança na rede. Esta abordagem contrasta com o modelo de perímetro de segurança, onde as máquinas na rede local são confiáveis e as máquinas não confiáveis estão separadas das primeiras por um *firewall*. A confiança zero na rede então assume que a comunicação entre quaisquer dois componentes deve ser autenticada e criptografada. Para o nosso caso de uso, isso poderia ser implementado usando certificados X.509 para o estabelecimento de canais TLS mutuamente autenticados, através dos quais o cliente verifica o servidor, como fizemos nas seções anteriores, mas também o servidor autentica o cliente. Este modelo é suportado pela maioria de sistemas maduros, a exemplo do Apache Kafka e MariaDB, está disponível

[34] https://helm.sh/

em bibliotecas das linguagens de programação populares e é suportado por ferramentas de gerência de aplicações nativas da nuvem através de padrões como SPIFFE[35].

Além de não confiar na rede, podemos assumir também que humanos não devem ser confiados com segredos. Esta estratégia foi exemplificada na Seção 1.4, onde um segredo, que poderia ser uma chave criptográfica, foi escolhida internamente pelo sistema (através do SCONE CAS) e entregue diretamente ao código da aplicação depois da atestação. Sem nunca ter saído dos enclaves, o segredo nunca poderia ter sido vazado por um humano. Tal estratégia poderia ser usada também para geração de certificados de autenticação de servidores e clientes X.509 para conexões TLS, isso permite que código seja aprovado e assinado por um comitê (de modo que nenhum indivíduo possa gerar código que exporte os segredos) e depois de implantado nunca exponha os dados sensíveis do sistema em execução.

A atestação do sistema operacional pode ser feita a partir do uso de componentes como o TPM (do inglês, *Trusted Platform Module*) [Arthur e Challener, 2015]. Enquanto o carregamento de um enclave registra o carregamento de uma aplicação, o TPM pode ser usado para registrar o carregamento do sistema operacional, assim como serviços e seus arquivos de configuração. Embora o TPM considere uma base de confiança muito maior (todo o ambiente da máquina), e por isso seja menos confiável, ele serve como camada adicional de segurança, por exemplo, provendo evidências que uma versão atualizada do sistema operacional foi usado e que serviços desnecessários não foram carregados.

Finalmente, o Apache Kafka poderia ser configurado para limitar quais tópicos cada cliente pode acessar. Isto é feito usando listas de controle de acesso (do inglês, Access Control List - ACL[36]). O controle de acesso pode então ser baseado nas identidades embutidas em certificados do cliente, como vistos acima. Assim, como estes certificados podem ser específico do cliente, mesmo que um deles seja comprometido, o domínio de impacto da falha é limitado.

### 1.5.4. Alternativas para computação confidencial, privacidade e leis de proteção de dados

Computação confidencial pode ser alcançada através de recursos de hardware, como o Intel SGX discutido aqui, ou de técnicas algorítmicas, como computação homomórfica [Gentry, 2009] (HE, do inglês, *Homomorphic Encryption*) e computação multi-parte [Cramer, Damgard, e Nielsen, 2015] (MPC, do inglês, *Multi-Party Computation*).

HE e MPC são técnicas que realizam processamento sobre dados numéricos criptografados. O dado só é entregue a entidades não-confiáveis após ser criptografado. Embora não possam visualizar os dados originais, essas entidades conseguem realizar operações aritméticas sobre os dados criptografados. Com HE, a cifragem transforma cada valor numérico em um polinômio e o valor correto do dado pode ser decifrado por quem tiver a chave de entrada correta para a função polinomial. A desvantagem é que simples operações aritméticas se tornam custosas operações polinomiais [Fan e Vercauteren, 2012]. A técnica de MPC é semelhante à HE, pois também opera sobre

[35] https://spiffe.io/
[36] https://kafka.apache.org/documentation/#security

dados criptografados, porém, é ainda mais complexa pois as operações são realizadas de forma distribuída e síncrona, o que adiciona latência de rede ao processo [Orlandi, 2011]. Uma consequência positiva que se aplica à HE e MPC é que o fato de adições e multiplicações serem suficientes para replicar as operações lógicas AND, OR, XOR e NOT, permite que HE e MPC possam ser generalizadas para realizar computações arbitrárias, embora esta generalização seja um processo complexo [Gentry, 2010].

A HE pode ser categorizada em três tipos: computação parcialmente homomórfica (PHE, do Inglês *Partially Homomorphic Encryption*), computação um tanto homomórfica (SWHE, do Inglês *Somewhat Homomorphic Encryption*), e computação completamente homomórfica (FHE, do Inglês *Fully Homomorphic Encryption*). O PHE permite a realização apenas de um conjunto limitado de operações sobre dados criptografados, como por exemplo, apenas adições ou apenas multiplicações, como é o caso do algoritmo de criptografia RSA que é homomorficamente multiplicativo mas não é homomorficamente aditivo. No SWHE, adições e um número limitado de multiplicações são permitidas sobre um mesmo conjunto de dados criptografados. No FHE, adições e multiplicações podem ser realizadas de forma indiscriminada, mas em compensação a quantidade de operações adicionais requeridas para reduzir o impacto do ruído gerado durante a adição e multiplicação dos polinômios tornam a computação (quase) impraticável [Naehrig, Lauter, e Vaikuntanathan, 2011]. Para se ter uma ideia desse custo, Hayward e Chiang (2015) sorteou 20 números inteiros de 8 bits e computou a soma, produto vetorial, e variância, sobre esses números, que custaram, respectivamente, 34.9 segundos, 952.17 segundos, e 2496.62 segundos [Hayward e Chiang, 2015], operações que durariam apenas algumas dezenas de ciclos em um enclave.

Uma vantagem clara de enclaves sobre HE e MPC é o custo computacional. Quando se trata da técnica de HE, existe um custo-benefício entre eficiência do processamento confidencial (PHE e SWHE) e a capacidade de executar uma quantidade arbitrária de operações de adição e multiplicação (FHE). PHE e SWHE são mais eficientes em tempo, porém mais limitadas em termos de capacidades de processamento confidencial, pois cada valor criptografado acumula ruído à medida que é adicionado ou multiplicado, até um dado ponto em que finalmente o ruído torna o resultado criptografado indecifrável [Naehrig, Lauter, e Vaikuntanathan, 2011]. O FHE, por outro lado, possui alta capacidade de processamento confidencial, sendo generalizável para computar muito além de operações aritméticas, mas são pouco viáveis em termos de tempo computacional para a grande maioria dos problemas [Hayward e Chiang, 2015]. Outra vantagem dos enclaves SGX é a capacidade de atestação de código, já que mesmo com a confidencialidade garantida, um código baseado em HE e MPC poderia ser adulterado para realizar computações diferentes da esperada.

Enquanto computação confidencial tem como diferencial a proteção de dados em uso, estando associada a mecanismos de comunicação e armazenamento seguros, privacidade tem como objetivo garantir a capacidade de agir seletivamente sobre quais de suas informações são compartilhadas. Computação confidencial pode então ser considerada uma ferramenta auxiliar na tarefa de garantir privacidade, já que sem mecanismos de confidencialidade, não haveria privacidade.

Por outro lado, existem estratégias alternativas que operam sobre os dados com o objetivo de reduzir informação, preservando a privacidade nos dados, mas podendo deteriorar a qualidade do dado original - anonimização [Zouinina et al., 2020] e privacidade diferencial [Nguyen, Kim e Kim, 2013]. Em termos de privacidade, anonimização pode simplesmente diminuir a informação disponível, eliminar os nomes de pessoas ou substituindo valores numéricos exatos por intervalos. Além de poder reduzir a utilidade dos dados, o uso de anonimização é frequentemente criticado pois pode deixar informações suficientes para que a anonimização seja desfeita, como no notório caso da Netflix [Narayanan e Shmatikov, 2008], onde o processo de anonimização foi desfeito para alguns usuários contidos no banco de dados.

Além dos aspectos de privacidade, outro conceito relacionado é a rastreabilidade. Computação confidencial por si só também pode não ser suficiente para garantir que indivíduos tenham controle sobre seus dados: quem os está usando, de que forma, e que como pode revogar o uso. Estes são alguns poderes exigidos nas leis mais recentes de proteção de dados, como a Lei Geral de Proteção de Dados (LGPD, lei nº 13.709, com vigência a partir de setembro de 2020).

Nesse sentido, empresas que lidam com dados de terceiros precisam se adequar aos requisitos da LGPD. As principais mudanças envolvem uma remodelagem da governança e gestão de processos, buscando promover transparência e facilidade de acesso às informações de utilização dos dados pessoais. Embora esse movimento por parte das empresas tenha se tornado evidente, existe uma distinção entre política e mecanismo. Mesmo que a lei estipule políticas, é possível que uma determinada empresa que em algum momento fez uso inadequado de dados pessoais altere informações emitidas em relatórios de uso de dados de clientes.

O processamento confidencial posto em prática por enclaves provêem duas interessantes propriedades que podem elevar o grau de auditabilidade e transparência de empresas que desejam estar aderentes à LGPD: (1) proteção de dados em memória e (2) atestação de um ambiente de execução. Enquanto a proteção de dados em uso na memória, discutida anteriormente, protege os dados de cópia por operadores e invasores, os procedimentos de atestação garantem que bibliotecas e binários em um ambiente de execução não foram adulterados. Protegidas contra alterações, as aplicações se comportarão da forma esperada, o que pode ser usado para implementar mecanismos de criação de relatórios na utilização de dados. Assim, os sistemas de controle e os relatórios de acesso serão confiáveis. Tal abordagem pode ser complementada pela adição de informações desses relatórios em um registro público imutável, como por exemplo, uma *blockchain*.

Uma implementação das capacidades mencionadas acima pode ser encontrada na plataforma DNAT [Nascimento et al., 2020] (*Data and Application Tracking*). Esta plataforma emprega computação confidencial via TEEs para impedir o acesso de agentes indevidos a dados sensíveis, bem como prover o rastreamento do uso dos dados pertencentes a uma entidade, e conceder a esta entidade o poder de revogação do uso de seus dados. No DNAT, registros públicos imutáveis são empregados para guardar o histórico de uso de um determinado dado assim como a aquisição de direito de uso e possíveis revogações. O uso de um dado é intermediado pela própria plataforma, a qual

emprega os registros públicos imutáveis e o Intel SGX para garantir que só agentes devidamente autorizados façam uso dos dados.

## 1.6. Considerações finais

Este trabalho apresentou a construção de uma aplicação prova-de-conceito para a disseminação e processamento de dados de sensores de forma confidencial. O uso de um barramento de mensagens Kafka e de um ambiente de execução SCONE facilitam a adaptação da aplicação para outros fins.

Alguns trechos de código foram omitidos neste trabalho. Os códigos completos podem ser obtidos no repositório Gitlab do minicurso [LSD, 2020]. Além do código, o repositório apresenta exemplos complementares e instruções para a criação de máquinas virtuais com suporte a SGX no provedor Microsoft Azure. Além da Azure, usuários do sistema RNP poderão contratar recursos de nuvem com suporte a SGX a partir do portal NasNuvens[37]. De forma semelhante, tais usuários poderão ter acesso ao sistema LiteCampus de processamento de dados de sensores com suporte a processamento confidencial, sistema que serviu de referência para o caso de uso discutido aqui.

## 1.7. Agradecimentos

Este trabalho foi apoiado pelo Laboratório de Sistemas Distribuídos (LSD) da Universidade Federal de Campina Grande, pelo GT LiteCampus da Rede Nacional de Pesquisa (RNP) e pela Smartiks Ltda. Agradecemos a José Benardi Nunes e Eduardo Falcão pelas sugestões e ao time da Scontain pelo apoio na utilização do SCONE.

## Referências

Armel, K. C., Gupta, A., Shrimali, G., Albert, A. (2013) “Is disaggregation the holy grail of energy efficiency? The case of electricity”, Energy Policy.

Arnautov, S., Trach, B., Gregor, F., Knauth, T., Martin, A., Priebe, C., Lind, J., Muthukumaran, D., O'Keeffe, D., Stillwell, M. L., Goltzsche, D., Eyers, D., Kapitza, R., Pietzuch, P., Fetzer, C. (2016) “SCONE: Secure Linux Containers with Intel SGX”. Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation.

Arthur, W., Challener, D. A practical guide to TPM 2.0: Using the new trusted platform module in the new age of security. Springer Nature, 2015.

Barbosa, P., Brito, A., Almeida, H. (2016) “A Technique to provide differential privacy for appliance usage in smart metering”, Information Sciences.

Cramer, R., Damgard, I. B., Nielsen, J. B., Secure multiparty computation. Cambridge University Press (2015).

Costan, V., Devadas, S. (2016) “Intel SGX Explained”. IACR Cryptol. ePrint Arch., v. 2016, n. 86.

Eugster, P. T., Felber, P., Guerraoui, R., Kermarrec, A.-M. (2003) “The many faces of publish/subscribe”. ACM computing surveys (Volume 35).

[37] https://www.nasnuvens.rnp.br/cms/

Fan. J., Vercauteren, F., “Somewhat practical fully homomorphic encryption”, IACR Cryptology ePrint Archive, vol. 2012, p. 144, 2012.

Gentry, C. A fully homomorphic encryption scheme. Tese de doutorado, Stanford University Stanford (2009).

Gentry, C. (2010) Computing arbitrary functions of encrypted data. Communication of the ACM 53, 3 (2010).

Hayward, R., Chiang, C. (2015) Parallelizing fully homomorphic encryption for a cloud environment, Journal of Applied Research and Technology, Volume 13, Issue 2.

IBM Security. Cost of a Data Breach Report 2020. (Online, acessado em 15/09/2020) https://www.ibm.com/security/data-breach

LSD. Processamento confidencial de dados de sensores na nuvem (Repositório). Laboratório de Sistemas Distribuídos, Universidade Federal de Campina Grande. (Online, acessado em 25/09/2020). https://git.lsd.ufcg.edu.br/lsd-sbseg-2020

Microsoft. Azure Confidential Computing Documentation. (Online, acessado em 24/09/2020). https://docs.microsoft.com/en-us/azure/confidential-computing/

Naehrig, M., Lauter, K., Vaikuntanathan, V. (2011) “Can homomorphic encryption be practical?”, Proceedings of the 3rd ACM Workshop on Cloud Computing Security.

Nguyen, H., Kim, J., Kim, Y. (2013) “Differential privacy in practice”, Journal of Computing Science and Engineering.

Narayanan, A., Shmatikov, V. (2008) “Robust De-anonymization of Large Sparse Datasets”, Proceedings of the IEEE Symposium on Security and Privacy.

Nascimento, J. R., Nunes, J. B. S., Falcão, E. L., Sampaio, L., Brito, A. (2020) “On the tracking of sensitive data and confidential executions”, Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems.

Oleksenko, O., Trach, B., Krahn, R., Martin, A., Silberstein, M., Fetzer, C. (2018) “Varys: Protecting SGX enclaves from practical side-channel attacks”, Proceedings of the 2018 USENIX Annual Technical Conference.

Orlandi, C., “Is multiparty computation any good in practice?”, Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, 2011.

Silva, F., Silva, M., Brito, A. (2020) “KafkaProxy: data-at-rest encryption and confidentiality support for Kafka cluster”, Anais do 20o Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais.

Verma, A., Pedrosa, L., Korupolu, M. R., Oppenheimer, D., Tune, E., Wilkes, J. (2015), “Large-scale cluster management at Google with Borg”, Proceedings of the ACM European Conference on Computer Systems.

Zouinina S., Bennani Y., Rogovschi N., Lyhyaoui A. (2020) “A Two-Levels Data Anonymization Approach”, Artificial Intelligence Applications and Innovations. Springer.

# Capítulo 2

# Processamento de Linguagem Natural para Identificação de Notícias Falsas em Redes Sociais: Ferramentas, Tendências e Desafios

Nicollas R. de Oliveira (UFF), Pedro Silveira Pisa (Solvimm),
Bernardo Costa (Solvimm), Martin Andreoni Lopez (Samsung Research),
Igor Monteiro Moraes (UFF), Diogo M. F. Mattos (UFF)

***Abstract***

*The epidemic spread of fake news is a side effect of the expansion of social networks to circulate news, in contrast to traditional mass media such as newspapers, magazines, radio, and television. Human inefficiency to distinguish between true and false facts exposes fake news as a threat to logical truth, democracy, journalism, and credibility in government institutions. In this chapter, we present methods for preprocessing data in natural language, vectoring, dimensionality reduction, machine learning, and quality assessment of information retrieval. We also present a practical demonstration of identifying fake news will, from the collection and textual processing of social media news to the application of detection algorithms.*

***Resumo***

*A disseminação epidêmica de notícias falsas (fake news) é um efeito colateral da expansão do uso de redes sociais como meio de circulação de notícias, em contraste às mídias tradicionais de comunicação massiva, como jornal, revista, rádio e televisão. A ineficiência humana para distinção entre fatos verídicos e falsos expõe as notícias falsas como uma ameaça à verdade lógica, à democracia, ao jornalismo e à credibilidade nas instituições de governo. Este capítulo apresenta métodos de pré-processamento de dados em linguagem natural, vetorização, redução de dimensionalidade, aprendizado de máquina e avaliação da qualidade de recuperação de informação. Ao final do capítulo é apresentada uma demonstração prática do processo de identificação de notícias falsas, desde a coleta e processamento textual de notícias de mídias sociais até a aplicação algoritmos de classificação.*

Este capítulo foi realizado com recursos do CNPq, CAPES, RNP, FAPERJ, FAPESP (2018/23062-5) e Prefeitura de Niterói/FEC/UFF (Edital PDPA 2020).

## 2.1. Introdução

A veracidade da informação é parte da essencial da sua integridade. O combate às notícias falsas torna indissociáveis os problemas de integridade e veracidade da informação em rede social e do consumo de dados na camada de aplicação. A divulgação de conteúdo falso implica desperdícios de recursos da rede e de processamento e, também, consiste em grave ameaça à integridade das informações e à credibilidade do serviço prestado [de Oliveira et al., 2020]. Dessa forma, o compartilhamento de informações inverídicas diz respeito à qualidade da confiança (*Quality of Trust* - QoT) aplicada à distribuição de notícias [Liu et al., 2010], referindo-se a quanto um usuário confia em um conteúdo de uma determinada fonte.

Em diferentes países, observam-se baixos níveis de confiança nas mídias de massa, *e.g.* apenas 40% nos Estados Unidos[2], enquanto há altas porcentagens de compartilhamento de *links* nunca lidos (*blindshares*), *e.g.* 59% no Reino Unido [Gabielkov et al., 2016]. Em 2016, durante as eleições presidenciais dos Estados Unidos, a sociedade americana testemunhou uma epidemia alarmante de notícias falsas, cujos efeitos foram sentidos multilateralmente. Efeito semelhante foi sentido nas eleições de 2018 no Brasil. Devido ao seu potencial de disseminação, aceitação e destruição [Vosoughi et al., 2018], as notícias falsas são atualmente uma das grandes ameaças ao conceito de verdade lógica, deteriorando a democracia, o jornalismo, a justiça e até a economia [Zhou e Zafarani, 2018, Wang, 2017]. Esta última, em especial, teve de lidar com flutuações de 130 bilhões na bolsa de valores, como consequência de uma declaração falsa afirmando que Barack Obama havia se ferido em uma explosão[3]. Nesse sentido, há um crescente esforço conjunto da comunidade acadêmica para desenvolver abordagens capazes de analisar, detectar e intervir na atuação desses conteúdos enganosos. Comprovações científicas já revelaram a vulnerabilidade dos humanos em distinguir verdade e falsidade, sendo reduzida a quase uma probabilidade aleatória, em média 54% de acerto [Zhou e Zafarani, 2018, Wang, 2017, Rubin, 2010, Rubin et al., 2016].

O objetivo deste capítulo é apresentar os principais algoritmos e técnicas que auxiliam na caracterização linguística e detecção de notícias falsas em redes sociais para a garantia da integridade da informação. Este capítulo caracteriza o fenômeno [Rubin et al., 2015a, Rubin et al., 2015b], investiga a propagação em mídias sociais e apresenta as ferramentas e os algoritmos para a detecção de notícias falsas.

O fator chave que impulsiona a criação de notícias falsas é que estas são criadas e publicadas *online*, de maneira mais rápida e barata quando comparadas a veículos tradicionais de mídia como jornais e televisão. Assim, o capítulo evidencia que embora a identificação de notícias falsas possa ser realizada manualmente por profissionais em jornalismo, o foco deste capitulo está na identificação automática através de aparato computacional. A identificação automática pode seguir abordagens distintas, como a prova automática de afirmações lógicas através de fatos já conhecidos, através da análise de propagação das notícias nas redes sociais, através da análise do perfil dos usuários que

[2]Disponível em https://news.gallup.com/poll/185927/americans-trust-media-remains-historical-low.aspx.

[3]Disponível em https://www.forbes.com/sites/kenrapoza/2017/02/26/can-fake-news-impact-the-stock-market/#559102f12fac.

compartilham as notícias ou através do processamento de linguagem natural para a extração de conhecimento em uma abordagem estilístico-computacional [Zhou e Zafarani, 2018]. O escopo do capítulo se limita à abordagem estilístico-computacional baseada em processamento de linguagem natural e justifica-se no fato de que o consumo de dados em redes sociais, por parte de usuários, está restrito à informação que chega ao usuário final. O usuário não tem acesso aos modelos de disseminação de conteúdo ou a modelos de reputação dos usuários que compartilharam os conteúdos consumidos. No capítulo são apresentas ainda as métricas de qualidade na extração das informações, assim como plataformas computacionais, disponíveis no mercado, que já executam o processamento de linguagem natural como serviço na nuvem.

O restante do capítulo está organizado da seguinte forma. A Seção 2.2 define o fenômeno da propagação de notícias falsas. Os métodos tradicionais de identificação de notícias falsas são discutidos na Seção 2.3. A criação de uma base de dados para identificação correta de notícias falsas é apresentada na Seção 2.4. A Seção 2.5 descreve o processamento de dados em linguagem natural. A Seção 2.6 explica os processos para a transformação de textos em matrizes operáveis computacionalmente, enquanto a Seção 2.7 apresenta as principais ferramentas de aprendizado de máquina usadas sobre dados em linguagem natural. A Seção 2.8 elenca soluções comerciais em nuvens computacionais para tratamento de dados em linguagem natural e a Seção 2.9 descreve iniciativas de pesquisas para a identificação de notícias falsas. Os desafios e oportunidades são discutidos na Seção 2.10. A Seção 2.11 apresenta uma atividade prática para a identificação de notícias falsas. A Seção 2.12 realiza as considerações finais do trabalho.

## 2.2. A Definição de Notícias Falsas

O termo notícias falsas (*fake news*) originalmente faz referência a informações falsas e muitas vezes sensacionais divulgadas sob o disfarce de reportagem. Contudo, o uso desse termo evoluiu e, atualmente, é considerado sinônimo de propagação de informações falsas em mídias sociais [Sharma et al., 2019]. Ressalta-se que, segundo o *Google Trends*, o termo "fake news" alcançou grande popularidade entre os anos de 2017 e 2018, tendo o pico de popularidade em outubro de 2018, quando houve a eleição presidencial no Brasil [4].

Notícias falsas são definidas como notícias que são intencionalmente e comprovadamente falsas [Zhou e Zafarani, 2018], ou, como quaisquer informações apresentadas como notícias que são factualmente incorretas e projetadas para enganar o consumidor, fazendo-o acreditar que são verdade [Golbeck et al., 2018]. Sharma *et al.* argumentam que essas definições, no entanto, são restritas pelo tipo de informação ou pela intenção de engano e, portanto, não capturaram o escopo amplo do uso atual. Assim, Sharma *et al.* definem o termo como uma notícia ou mensagem publicada e propagada pela mídia, contendo informações falsas, independentemente dos meios e motivos por trás dela [Sharma et al., 2019]. Apesar da inexistência de um consenso claro sobre o conceito de notícias falsas, a definição formal mais aceita interpreta como notícias intencionalmente e verificavelmente falsas. Com relação essa definição, destacam-se dois aspectos: a intenção e a autenticidade. O primeiro aspecto diz respeito à intenção desonesta usada com o intuito

[4]Disponível em https://trends.google.com.br/trends/explore?date=all&geo=BR&q=fake%20news.

de enganar o leitor. Já o segundo se relaciona com a possibilidade de essas informações falsas terem sua veracidade verificadas.

As notícias falsas podem ser diferenciadas pelos meios empregados para falsificar informações. O conteúdo das notícias pode ser completamente falso, totalmente fabricado para ludibriar o consumidor ou pode ser um conteúdo ardiloso que se utiliza de informações enganosas para abordar um determinado problema. Há também a possibilidade de serem usados conteúdos impostores que simulam fontes genuínas, mas, quando na verdade, as fontes são falsas. Outras características fraudulentas dos conteúdos de notícias falsas são também o uso de conteúdos manipulados, como manchetes e imagens que não estão de acordo com o conteúdo veiculado, ou também a contextualização da notícia com elementos falsos, assim como com conteúdo legítimo, porém em um contexto falso.

As notícias falsas também apresentam motivações ou intensões diversas. São identificadas como motivações para a criação e divulgação de notícias falsas as intensões: de prejudicar ou desacreditar pessoas ou instituições; intensões de lucro para gerar ganhos financeiros aumentando a veiculação e a visualização de publicações *online*; intensões de influenciar e manipular a opinião pública; assim como intensões de promover a discórdia ou, simplesmente, por diversão.

Diversos conceitos concorrem e se sobrepõem ao conceito de notícias falsas. Uma síntese desses múltiplos conceitos, não considerados notícias falsas, pode ser elencada como [Rubin et al., 2015a, Shu et al., 2017, Zhou e Zafarani, 2018, Chen et al., 2015]:

1. **sátiras e paródias**, que pelo conteúdo humorístico embutido, usando sarcasmos e ironias, é factível de ter seu caráter enganoso identificado;
2. **rumores e boatos**, que não se originaram de eventos de notícias, porém são aceitos publicamente;
3. **teorias de conspiração**, por não serem facilmente verificáveis como verdadeiras ou falsas;
4. ***spams***, comumente associados a *e-mails* não desejados, os *spams* constituem qualquer campanha publicitária que chega aos leitores por mídia sociais sem que sejam desejadas;
5. **trotes e embustes (*hoaxes*)** que são motivados apenas por diversão ou para enganar indivíduos direcionados;
6. **caça-cliques (*clickbait*)** que pelo fato de empregarem imagens em miniaturas, ou manchetes sensacionalistas, no processo convencimento de usuários a acessarem e compartilharem conteúdos duvidosos, mais se assemelham a um tipo de propaganda falsa;
7. **desinformação (*misinformation*)** que é criada involuntariamente, sem uma origem ou intenção específica de desorientar o leitor;
8. **contra-informação (*disinformation*)** que são informações criadas com intenção específica de confundir o leitor.

As características de cada um desses tipos de conteúdo fraudulento são comparadas às notícias falsas na Tabela 2.1

**Tabela 2.1. Termos e conceitos relacionados à notícias falsas.**

| | **Autenticidade** | **Intenção** | **Forma de Notícia** |
|---|---|---|---|
| Sátira e Paródias | Falsa | Não Ruim | Não |
| Rumores e Boatos | Desconhecida | Desconhecida | Desconhecido |
| Teorias de Conspiração | Desconhecida | Desconhecida | Não |
| Spam | Possivelmente Verdadeira | Ruim / Publicitária | Não |
| Trotes e Embustes | Falsa | Não Ruim | Não |
| Caça-cliques | Possivelmente Verdadeira | Publicitária | Não |
| Desinformação | Falsa | Desconhecida | Desconhecido |
| Contra-informação | Falsa | Ruim | Desconhecido |

### 2.2.1. As Características das Notícias Falsas

O crescimento das comunicações mediadas por mídia social é um dos principais fatores que fomentam a mudança de características nas notícias falsas atuais [Sharma et al., 2019]. A incapacidade de um indivíduo de discernir com precisão as notícias falsas das verdadeiras leva ao compartilhamento contínuo e à crença em informações falsas nas redes sociais [Zhou e Zafarani, 2018, Wang, 2017, Rubin, 2010, Rubin et al., 2016]. É difícil para um indivíduo diferenciar entre o que é verdadeiro e o que é falso enquanto é sobrecarregado com informações enganosas que são recebidas por repetidas vezes. Ademais, os indivíduos tendem a confiar em notícias falsas porque há atualmente uma descrença do público em relação às mídias tradicionais e, porque, muitas vezes tais notícias são compartilhadas por amigos ou confirmam um conhecimento prévio. Isso torna a identificação de notícias falsas mais crítica em comparação a outros tipos de informações, já que geralmente são apresentadas com elementos que lhe conferem autenticidade e objetividade, sendo relativamente mais fácil de obter a confiança do público.

A mídia social e o compartilhamento colaborativo de informações em plataformas *online* fomentam também a propagação de notícias falsas, efeito chamado de câmara de eco (*echo chamber effect*) [Shu et al., 2020]. O realismo ingênuo, em que os indivíduos tendem a acreditar mais facilmente nas informações que estão alinhadas a seus pontos de vista, o viés de confirmação, no qual os indivíduos procuram e preferem receber informações que confirmam seus pontos de vista existentes, e teoria da influência normativa, em que os indivíduos escolhem compartilhar e consumir opções socialmente seguras como uma preferência para aceitação e afirmação em um grupo social, são fatores importantes na percepção e compartilhamento de notícias falsas que fomentam o efeito da câmara de eco [Shu et al., 2020]. Esses conceitos implicam a necessidade de os indivíduos buscarem, consumirem e compartilharem informações que estejam alinhadas com suas visões e ideologias. Como consequência, os indivíduos tendem a formar conexões com indivíduos ideologicamente semelhantes e, de forma complementar, os algoritmos de recomendação de redes sociais tendem a personalizar recomendações de conteúdos que atendam às preferências de um indivíduo ou grupo. Esses comportamentos levam à formação de câmaras de eco e bolhas de filtro, em que os indivíduos ficam menos expostos a pontos

de vista conflitantes e ficam isolados em sua própria bolha de informação [Fuller et al., 2009, Sharma et al., 2019]. O confinamento das notícias falsas em câmaras de eco ou bolhas de informação tendem a aumentar as suas sobrevida e divulgação, pois incorrem no fenômeno da credibilidade social, que sugere que a percepção das pessoas sobre a credibilidade de uma informação aumenta se os outros também a percebem como verdadeira, já que há a tendência de que indivíduos considerem como verídica uma informação a que são submetidos repetidas vezes [Rubin et al., 2016].

Os padrões de difusão de notícias falsas nas redes sociais têm sido frequentemente estudados para identificar as características das notícias falsas que auxiliam na discriminação entre notícias falsas e verdadeiras. O problema de identificação de notícias falsas pode ser definido de diversas formas. A classificação pode ser vista como a execução de uma classificação binária entre falsa ou verdadeira, boato ou não boato, farsa ou não farsa. Outra forma de definir o problema é como a execução de uma classificação de várias classes, verdadeira, quase verdadeira, parcialmente verdadeira, principalmente falsa ou falsa, ou ainda como rumor não verificado, rumor verdadeiro, rumor falso ou não é rumor [Sharma e Sharma, 2019]. A principal diferença entre a definição dos problemas de classificação é em função dos diferentes esquemas de anotação ou contextos de aplicativos em conjuntos de dados diferentes. Normalmente, os conjuntos de dados são coletados de declarações anotadas em sítios *web* de verificação de fatos, como o "Fato ou Fake"[5] ou a "Agência Lupa"[6]. Esses sítios refletem o esquema de rotulagem usado pela organização de verificação de fatos específica.

Sharma *et al.* identificam três características relevantes para a identificação de notícias falsas: as fontes ou promotores da notícia; o conteúdo da informação; e a resposta do usuário ao receber a notícia em redes sociais [Sharma et al., 2019]. A fonte ou os promotores da notícia têm grande influência na classificação da veracidade da notícia. Contudo, Sharma *et al.* ressaltam que as listas de fontes possíveis de notícias falsas não são exaustivas e que os domínios usados para a divulgação de uma notícia podem ser falsificados [Sharma et al., 2019]. Outro ponto a ser ressaltado é que *bots*, contas falsas ou comprometidas controladas por humanos ou programas para apresentar e promover informações nas redes sociais, são responsáveis por acelerar a velocidade de propagação de informações verdadeiras e falsas de forma quase igual, a fim de alavancar a credibilidade e a reputação das contas de *bots* [Davis et al., 2016]. A segunda característica importante é o conteúdo da informação propagada. O conteúdo da informação é uma das principais características a ser analisada para classificar a notícia como verdadeira ou falsa. Oliveira *et al.* identificam que notícias falsas e notícias reais veiculadas no Brasil têm um comportamento estatisticamente diferente no somatório da frequência relativa das palavras usadas no conteúdo. Notícias falsas tendem a usar menos palavras relevantes do que notícias reais [de Oliveira et al., 2020]. Outras características textuais incluem o uso de palavras sociais, auto-referências, declarações de negação, reclamações e itens generalizantes, além de que há uma tendência de que notícias falsas apresentem menor complexidade cognitiva, menos palavras exclusivas, mais palavras de emoção negativa e mais palavras de ação [Sharma et al., 2019]. Por fim, as respostas do usuário nas redes sociais fornecem informações auxiliares para a detecção de notícias falsas. A resposta

[5]Disponível em https://g1.globo.com/fato-ou-fake/.

[6]Disponível em https://piaui.folha.uol.com.br/lupa/.

dos usuários é importante para a identificação, pois, somada aos padrões de propagação, são mais difíceis de serem manipuladas do que o conteúdo da informação e, por vezes, as respostas dos usuários contêm informações óbvias sobre a veracidade [Zhou e Zafarani, 2018]. O engajamento dos usuários, nas formas de curtidas, compartilhamentos, respostas ou comentários, contém informações capturadas na estrutura de árvores de propagação que indicam o caminho do fluxo de informações, informações temporais em carimbos de data e hora, informações textuais em comentários do usuário e informações de perfil do usuário envolvido no engajamento [Sharma et al., 2019].

A caracterização da fonte ou da propagação, do conteúdo e da reposta do usuário permitem definir diferentes técnicas de identificação de notícias falsas, tais como, identificação baseada em retroalimentação pelo padrão de propagação, identificação no processamento de linguagem natural aplicado ao conteúdo de mensagens e aplicação de mecanismos de aprendizado de máquina e, por fim, identificação baseada em intervenção dos usuários. Este capítulo foca nas soluções baseadas na análise do conteúdo das notícias.

### 2.2.2. O Processo de Disseminação de Notícias Falsas

Diversas entidades, indivíduos e organizações interagem na divulgação, moderação e consumo de notícias falsas nas redes sociais. Devido à pluralidade de atores envolvidos, o problema de identificação e mitigação da disseminação de notícias falsas torna-se ainda mais complexo. A divulgação das notícias falsas é fortemente baseada em mídias sociais em detrimento de mídias tradicionais de jornalismo, devido à grande escala, ao alcance das mídias sociais e à capacidade de compartilhar colaborativamente conteúdo. Os sítios *web* de mídias sociais têm se tornado a forma mais popular de disseminação, devido à crescente facilidade de acesso e popularização da comunicação mediada por computador e do acesso à Internet [Mattos et al., 2019]. Paralelamente, enquanto nas mídias tradicionais de jornalismo a responsabilidade pela criação do conteúdo cabe ao jornalista e à organização redatora, a moderação nas redes sociais varia bastante. Cada mídia social está sujeita a diferentes regras de moderação e regulamentação de conteúdo. A informação é consumida principalmente pelo público em geral ou pela sociedade, que constituem número crescente de usuários de mídia social. O crescimento no consumo de informação por meio de mídia social aumenta o risco de notícias falsas causarem danos generalizados [Sharma et al., 2019].

Sharma *et al.* destacam três atores distintos na propagação das notícias falsas: o adversário, o verificador de fatos e o usuário susceptível [Sharma et al., 2019]. Os adversários são indivíduos ou organizações mal-intencionados que muitas vezes se passam por usuários comuns de redes sociais usando *bots* [Davis et al., 2016] ou contas reais. Os adversários podem tanto agir como fonte ou como promotores de notícias falsas. Essas contas também agem em grupo propagando conjuntos de notícias falsas. O verificador de fatos consiste em um conjunto de várias organizações de verificação de fatos, como "Fato ou Fake" e a "Agência Lupa", que buscam expor ou confirmar notícias que gerem dúvidas sobre a sua veracidade. Muitas das vezes, as verificações se baseiam no jornalismo de checagem de fatos que depende da verificação humana. Contudo, há soluções tecnológicas automatizadas que visam a detecção de notícias falsas para empresas e consumidores. Essas soluções atribuem pontuações de credibilidade a conteúdo da *web* usando inteligên-

cia artificial. Por fim, o usuário susceptível consiste no usuário de rede social que recebe o conteúdo duvidoso, porém não é capaz de distinguir entre uma notícia falsa ou verídica e, assim, acaba propagando a notícia falsa em sua rede social, mesmo que não tenha a intensão de contribuir para a proliferação de conteúdo fraudulento.

## 2.3. Os Métodos Tradicionais de Detecção de Notícias Falsas

A identificação de notícias falsas pode ser realizada por meios manuais, através de profissionais em jornalismo, sendo a abordagem mais utilizada normalmente. Contudo, tal abordagem não é compatível com o volume atual de criação e disseminação de conteúdo nas redes sociais. Para contrapor esse problema de escalabilidade, métodos automáticos geralmente integram técnicas de Recuperação de Informação, Processamento de Linguagem Natural (PLN) e Aprendizado de Máquina no processo de verificação da veracidade de notícias veiculadas na Internet.

A respeito de métodos automáticos de detecção de notícias falsas, distinções podem ser observadas ao discretizar as formas de detecção por foco de atuação. Na literatura são vislumbradas três teorias analíticas preponderantes e potencialmente úteis na contenção de notícias falsas. A primeira teoria é fundamentada em uma **análise baseada na propagação**, cujo foco está no mapeamento qualitativo ou quantitativo do espalhamento das notícias falsas em rede sociais, a partir de padrões empíricos ou modelagem matemática, respectivamente. A base de ambos os mapeamentos é a cascata de notícias falsas, uma estrutura em árvore que representa todo o processo de disseminação de notícias falsas, podendo ser pautada tanto em uma perspectiva por saltos ou por tempo. A Figura 2.1 retrata as perspectivas de representação da propagação citadas.

Um desses padrões de propagação foi mapeado por Kwon *et al.*, cujo estudo revelou uma tendência das notícias não-confirmadas exibirem múltiplos e periódicos picos de discussão ao longo do dia no *Twitter*, enquanto que notícias confirmadas apresentavam apenas um pico proeminente [Kwon et al., 2013]. Adicionalmente, os estudos de Zhou *et al.* e Vosoughi *et al.* alertaram sobre a capacidade das notícias falsas, sobretudo do âmbito político, se espalharem de maneira mais rápida, mais abrangente e com mais abrangência do que notícias verdadeiras. Tal conclusão foi embasada no comportamento da representação em cascata das notícias falsas, marcado por uma maior largura máxima, profundidade e tamanho, alcançados em menos tempo que a representação em cascata de notícias legítimas [Zhou et al., 2015, Vosoughi et al., 2018].

Embora útil, a descoberta de padrões empíricos de propagação característicos de cada tipo de notícia é uma estratégia com resultados temporários, pois a alta dinamicidade e variabilidade de comportamento das notícias falsas. É conveniente a aplicação em conjunto com uma modelagem matemática. Em geral, essa modelagem recorre a uma análise regressiva usando modelos clássicos como o epidêmico e o econômico.

A construção matemática da difusão de notícias falsas através de uma modelagem epidêmica visa principalmente a predição do número de disseminadores (temperatura geral). Essa estratégia de modelagem inicia com uma etapa que associa cada usuário a um dentre três estados: (i) disseminadores; (ii) potenciais disseminadores; e (iii) disseminadores arrependidos, aqueles que após encaminharem ou publicarem uma notícia falsa a apagam. Nesta etapa, há também a definição inicial das taxas de transição entre es-

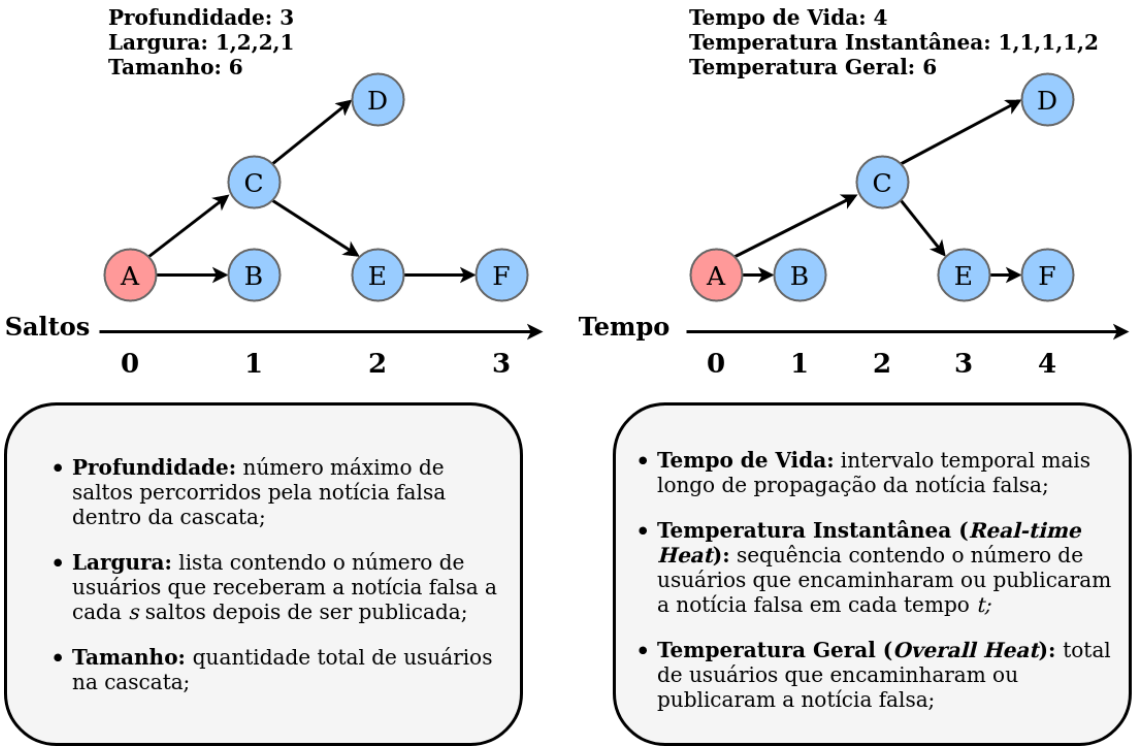



**Figura 2.1. Ilustração das cascatas de notícias falsas, tanto numa perspectiva baseada em saltos (à esquerda) quanto baseada em tempo (à direita). O nó-raiz $A$, em ambas as perspectivas, representa o primeiro usuário a publicar ou criar a notícia falsa e os demais nós representam usuários atuantes no encaminhamento ou compartilhamento do conteúdo falso.**

ses estados. A próxima etapa consiste na construção do modelo, a qual pode considerar fenômenos como o efeito *backfire* [7] e o reflexo de Semmelweis [8], que revelam a rejeição de indivíduos às ideias contrárias as suas. A terceira etapa consiste na determinação das taxas reais de transição entre estados [Zhou e Zafarani, 2018].

A modelagem econômica introduz uma abordagem racional sobre interações de notícias falsas, que tenta capturar e predizer o comportamento dos indivíduos ao serem expostos a uma notícia falsa. Neste tipo de modelagem, o ciclo de geração e consumo de notícias é visto como um jogo de estratégia entre dois jogadores, os publicadores e os consumidores. A cada jogador, a decisão de encaminhar ou deletar uma notícia falsa implica pares de vantagens específicas e excludentes entre si. Aos publicadores, cabe a escolha entre obter uma vantagem de curto prazo ($g_p$), que maximiza o lucro relacionado ao número de consumidores alcançados, ou uma vantagem de longo prazo ($b_p$), que privilegia sua reputação, tornando-os uma fonte autêntica de notícias. Já para os consumidores, as consequências dessa decisão dual é dividida entre uma vantagem de informação ($g_c$), que permite a obtenção de informação verdadeira e não enviesada, ou uma vantagem psicológica ($b_c$), ligada à teoria de viés confirmatório que reflete sua preferência por receber notícias que satisfazem opiniões prévias e necessidades sociais. Dessa forma,

[7] Relacionado ao fato de indivíduos rejeitarem mais fortemente evidências opostas as suas crenças.

[8] Remete a tendência dos indivíduos rejeitarem novas evidências por estas contradizerem suas normas e crenças estabelecidas.

quando $g_p > b_p$ e $g_c > b_c$ constrói-se uma cadeia propícia para o espalhamento de notícias falsas [Shu et al., 2017].

Paralelamente, existe a **análise baseada no usuário**, que considera o papel deste na disseminação das notícias, consequentemente distinguindo um usuário malicioso daqueles sem má intenção, os ingênuos. Sejam motivados por benefícios monetários ou não monetários, a atuação de usuários maliciosos nas redes sociais se dá através de contas que escondem a real identidade do gerenciador. Ao analisar o nível de participação de humanos no processo de gerenciamento dessas contas, pode-se dividi-las em três categorias: *social bots*, *cyborgs* e *trolls*. Todas essas contas maliciosas altamente ativas e partidárias têm um único propósito de tornarem-se fontes poderosas de proliferação de notícias falsas. Em um nível baixo de dependência humana, os *social bots* são contas controladas por um algoritmo de computador, cujo objetivo é produzir conteúdo automaticamente e interagir com humanos ou outros *bots*. Já em nível intermediário, os *cyborgs* são contas que alternam entre atividades automatizadas e humanas. Normalmente, este tipo de conta maliciosa é registrada por um usuário humano, fornecendo assim uma camuflagem para definir programas automatizados para realizar atividades nas redes sociais. No nível mais elevado de dependência, os *trolls* são contas totalmente mantidas por usuários humanos reais que visam perturbar comunidades *online* e provocar uma resposta emocional dos consumidores [Shu et al., 2017].

Outros trabalhos, como Barreto *et al.*, propõem uma metodologia capaz de distinguir usuários legítimos e *spammers* considerando a 2-vizinhança no *Twitter*. A proposta é subdivida em três etapas, cuja primeira é a pré-seleção manual de possíveis usuários. Como critério de pré-seleção de um usuário malicioso utiliza-se o fato do usuário enviar mensagens contendo pelo menos um tópico popular. A segunda etapa inclui a coleta dos dados da rede no entorno dos usuários pré-selecionados. Como última etapa, é feita uma análise desses dados avaliando métricas como distribuição de grau, centralidade de grau, coeficiente de agrupamento e *PageRank*. Ao final, os autores relatam um comportamento diferenciado da distribuição de grau dos *spammers*, contrariando a lei de potência esperada para os usuários legítimos [Barreto et al., 2014].

Mesmo não intencionalmente, usuários comuns são igualmente susceptíveis a se tornarem propagadores de notícias falsas. Além da baixa capacidade de detecção de notícias falsas, usuários normais são influenciados por fatores psicológicos e sociais. Na psicologia, esses fatores são identificados como vulnerabilidades individuais cujo um dos exemplos conhecidos é o realismo ingênuo. Esta vulnerabilidade formula uma tendência dos usuários em acreditar que suas percepções da realidade são os únicos pontos de vista, enquanto as demais são consideradas desinformadas, irracionais ou tendenciosas. Considerando o campo social, a disseminação das notícias falsas está intimamente conectada à dinâmica social dos indivíduos, estando correlacionada à três teorias: (i) a Teoria da Prospecção, que descreve a tomada de decisão como um processo pelo qual os indivíduos fazem escolhas com base nos ganhos e perdas relativas em comparação com seu estado atual; (ii) a Teoria da Identidade Social, que associa o autoconceito dos indivíduos é derivado a partir da percepção de pertencimento a um grupo social relevante; (iii) a Teoria da Influência Normativa, na qual enfatiza que aceitação e afirmação social são essenciais para a identidade e autoestima de um indivíduo, fazendo com que os usuários escolham ser "socialmente seguros" [Shu et al., 2017].

Embora a existência de notícias falsas preceda o surgimento das mídias sociais, seu advento alterou e ampliou a dinâmica de propagação desse tipo de informação, inclusive adicionando novos atores. Outro fator atual que facilita a disseminação desse tipo de notícia é o fenômeno de bolha social ou câmara de eco (*echo chamber*) em que usuários tendem a se relacionar virtualmente com seus *like-minders*, ou seja, pessoas que pensam como eles. Nessas bolhas sociais estão presentes duas ideias principais, sendo a primeira conhecida como credibilidade social. Tal ideia é explicada pelo fato de as pessoas serem mais propensas a considerar uma fonte como credível se os outros também a considerarem, especialmente quando não há como se comprovar. A segunda ideia remete a uma heurística de frequência, segundo a qual consumidores naturalmente preferem notícias que são ouvidas mais constantemente, mesmo sendo falsas [Shu et al., 2017].

Uma terceira teoria analítica remete à **análise baseada no estilo** da escrita, cujo foco de atuação principal está no conteúdo da notícia, ou seja, no texto propriamente dito. Essa análise parte da premissa de que notícias falsas detêm perfis de escrita únicos, diferentes dos seus pares legítimos. Cabe então aos métodos de detecção alinhados com essa teoria aplicar técnicas para extração de características linguísticas.

Dentre os estudos relacionados a essa abordagem estilística, destaca-se o apresentado por Rashkin *et al.*, que trabalha sob a hipótese de que as notícias falsas tendem a conter uma narrativa mais interessante a fim atrair leitores [Rashkin et al., 2017]. Assim, utilizando um *corpus* composto por artigos de notícias de diversas intenções, fontes e graus discretos de veracidade, o método empregado prevê a extração de características léxicas latentes. A análise dessas características permitiu formular perfis distintos de notícias dependendo da sua fonte de veiculação. Assim, constata-se que notícias oriundas de fontes confiáveis normalmente apresentam alguma forma de embasamento concreto, como comparações numéricas e expressões relativas a dinheiro. Em um sentido oposto, notícias de fontes menos confiáveis detinham uma incidência maior de pronomes de primeira e segunda pessoa, superlativos, advérbios de modo e palavras que expressam hesitação (*hedging words*). A análise baseada no estilo é a explorada a seguir.

## 2.4. A Construção da Base de Dados

A caracterização do problema de identificação de notícias como um problema de classificação implica a construção de uma base de dados adequada. A construção de uma base de dados com qualidade e disponibilidade é o pilar de qualquer mecanismo automático de detecção de notícias falsas. Sua importância está atrelada à necessidade de armazenar a máxima quantidade de exemplos contrastantes, notícias falsas e verdadeiras, para então serem absorvidos por algoritmos de aprendizagem de máquina [Oshikawa et al., 2018]. A Tabela 2.4 traz uma compilação de bases de dados de notícias falsas disponíveis, tanto na língua inglesa quando em na língua portuguesa.

Nesse contexto, uma eventual coleta errônea de dados tem o potencial de causar inúmeras consequências negativas, que variam desde a particularização da análise até a obtenção de resultados dissonantes. Logo, é prudente adotar algumas diretrizes sugeridas por Rubin *et al.* para a formação de um *corpus* de notícias falsas [Rubin et al., 2015a]. Rubin *et al.* defendem que qualquer construção de uma base de dados, *corpus*, de notícias falsas deve se ater a nove condições importantes, elencadas a seguir. (i) Considerar

**Tabela 2.2. Base de dados de notícias falsas disponíveis**

| | Conteúdo | Quant. | Rotulagem | Anotador |
|---|---|---|---|---|
| **Buzzface** [Santia e Williams, 2018] | Postagens e comentários de redes sociais (Facebook) | 2263 | Granular em quatro níveis (predominantemente verdade, predominantemente falso, mistura de verdadeiro e falso e nenhum conteúdo factual) | Previamente checado por agências de notícias (Buzzfeed) |
| **FAKENEWSNET** [Shu et al., 2020] | Artigos inteiros | 23921 | Binária (verdadeiro ou falso) | Previamente checado por agências de notícias (PolitiFact e GossipCop) |
| **Fake.Br Corpus** [Monteiro et al., 2018] | Artigos inteiros | 7200 | Binária (verdadeiro ou falso) | Considera a credibilidade da fonte |
| **LIAR** [Wang, 2017] | Declarações curtas (políticas) | 12,8k | Granular em seis níveis (verdade, predominantemente verdade, meia-verdade, quase verdade, falso, *pants-fire*) | Previamente checado por agências de notícias (PolitiFact) |
| **Emergent** [Ferreira e Vlachos, 2016] | Declarações e títulos relacionados | 300 | Binária (verdadeiro ou falso) | Equipe jornalística |
| **FEVER** [Thorne et al., 2018] | Declarações curtas (Wikipedia) | 185k | Granular em três níveis (suportada, refutada e sem informação suficiente) | Anotadores humanos treinados |
| **CREDBANK** [Mitra e Gilbert, 2015] | Postagens de redes sociais (Twitter) | 60M | Vetor com 30 dimensões contendo pontuações variáveis em cinco níveis de veracidade | Crowdsourcing |
| **BuzzfeedNews** | Postagens de redes sociais (Facebook) | 2282 | Granular em quatro níveis | Equipe jornalística |
| **BuzzFeed-Webis** [Potthast et al., 2017] | Postagens de redes sociais (Facebook) | 1687 | Granular em quatro níveis | Previamente checado por agências de notícias (Buzzfeed) |
| **PHEME** [Zubiaga et al., 2016] | Postagens de redes sociais (Twitter) | 330 | Binária (verdadeiro ou falso) | Jornalistas e crowdsourcing |

tanto as instâncias falsas como as verdadeiras permite que eventuais métodos preditivos aplicados à base considerem padrões característicos de cada tipo de notícia. (ii) A informação deve estar preferencialmente em formato textual, em vez de ser apresentada como mídia em formato de áudio ou vídeo. Informações nesses formatos devem ser transcritas, tornando-se manipuláveis por ferramentas de processamento de linguagem natural. (iii) A homogeneidade das notícias quanto ao tamanho e (iv) quanto a maneira da escrita, são outras duas condições a serem consideradas, evitando sempre que possível instâncias muito díspares. Igualmente, existe uma preocupação com (v) a forma de distribuição das notícias, visto que há suspeitas de que ao saber como e em qual contexto estas foram fornecidas, *e.g.* humorístico, sensacionalista, pode-se influenciar os leitores. Além disso,

(vi) a aquisição de notícias de um mesmo intervalo temporal é um fator primordial, pois os assuntos podem variar drasticamente em um curto intervalo de tempo. Adicionalmente, (vii) é aconselhável atender alguns aspectos pragmáticos, tais como custos com direito autoral, disponibilidade, facilidade de obtenção e privacidade dos escritores. Não se deve negligenciar o (viii) idioma e a (ix) cultura a que pertencem os dados coletados, pois a tradução pode implicar ambiguidades ou más interpretações, afetando negativamente a eficiência de processos de detecção [Rubin et al., 2015a, Rubin, 2014].

## 2.5. Processamento de Linguagem Natural

O Processamento de Linguagem Natural (PLN), também conhecido como linguística computacional, consolida-se como um campo de pesquisa que envolve modelos e processos computacionais para a solução de problemas práticos de compreensão e manipulação de linguagens humanas. Independentemente de sua forma de manifestação, textual ou fala, a linguagem natural é entendida como qualquer forma de comunicação diária entre humanos. Tal definição exclui linguagens de programação e notações matemáticas, consideradas linguagens artificiais. As linguagens naturais estão em constante mudança, dificultando o estabelecimento de regras explícitas para computadores [Clark et al., 2012, Otter et al., 2020, Bird et al., 2009].

Em uma decomposição refinada, o PLN pode ser dividido em cinco estágios primários de análise, que, quando realizados, permitem que o significado pretendido pelo autor seja extraído computacionalmente de um documento textual. Embora seja mais condizente com um estágio de pré-processamento, o primeiro estágio é a segmentação por *tokenização*. A *tokenização* é uma técnica obrigatória dado que os documentos textuais em linguagem natural geralmente são compostos de frases longas, complicadas e mal formadas. A etapa seguinte é a análise léxica, que visa relacionar as variantes morfológicas aos seus *lemas*, ou seja, a forma primitiva das palavras do dicionário. A análise sintática foca no relacionamento das palavras entre si, cada uma assumindo seu papel estrutural nas frases, e de como as frases podem ser partes de outras, constituindo sentenças. Linguisticamente, a análise semântica tenta destilar o significado de palavras, expressões fixas, sentenças inteiras, sendo assim frequentemente aplicada na resolução de ambiguidades. Por fim, a análise pragmática busca compreender uma determinada frase, observando referências pronominais e a coerência textual da estrutura das frases adjacentes. Embora o PLN possa introduzir outros estágios de análise, como reconhecimento de emoção, esses cinco estágios básicos são suficientes para extrair a informação semântica contextualizada de um documento de linguagem natural [Indurkhya e Damerau, 2010].

Limitando o processamento até o estágio de análise morfológica, é possível compor uma sequência básica de técnicas de PLN para garantir a identificação, e posterior remoção, de qualquer ruído textual que possa comprometer a extração e interpretação inteligente das informações contidas em cada sentença. Nesta sequência, ilustrada na Figura 2.2, são aplicadas técnicas de limpeza e conformação dos dados incluindo *tokenização*, remoção de pontuação e caracteres especiais, eliminação de *stopwords*, correção ortográfica, reconhecimento de entidades nomeadas e *stemização* ou *lematização*. Guiada pela ordem acima mencionada, cada *sentença* do texto original é primeiramente submetida a um procedimento de discretização visto na Etapa 1, conhecido como *tokenização*. Usando neste caso o caractere de espaço como critério delimitador, a *tokenização* trans-

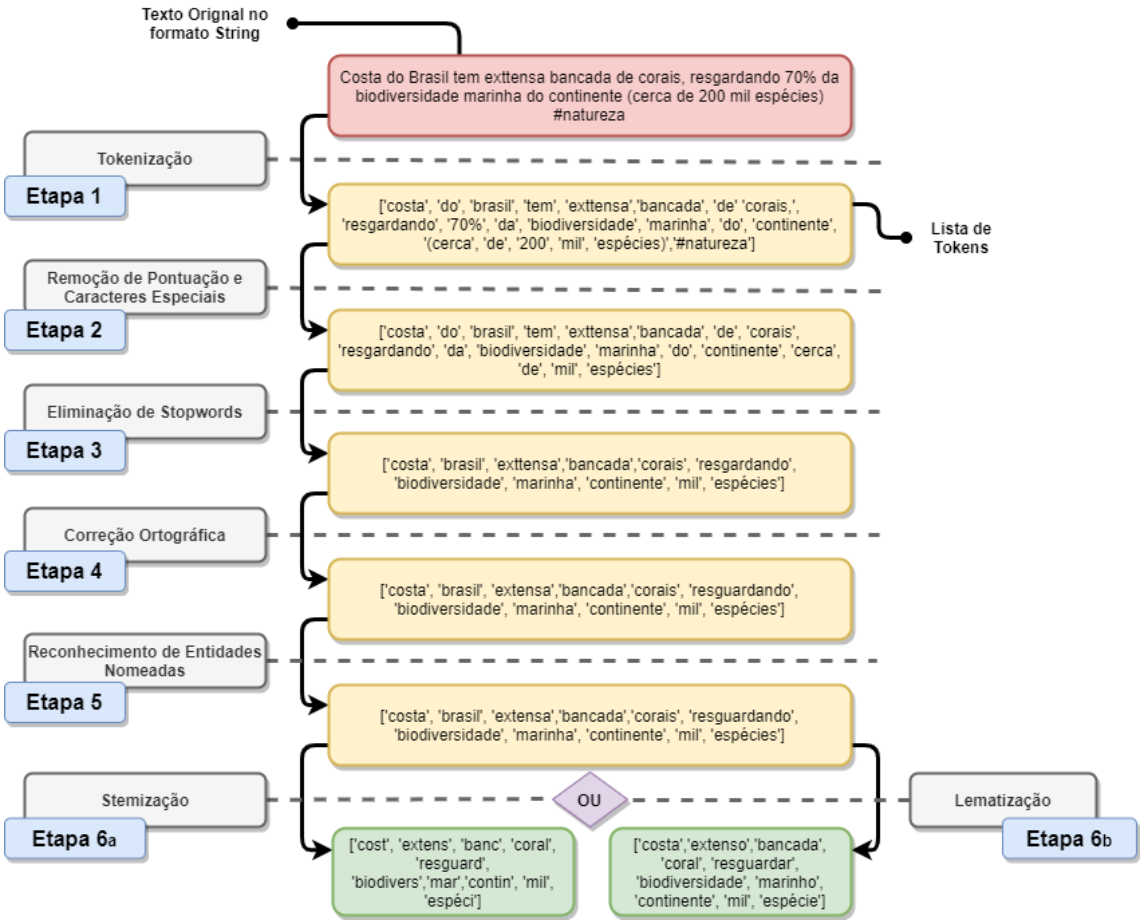



**Figura 2.2. Aplicação do processamento de linguagem natural em um texto bruto. A tokenização segmenta o texto contíguo em um conjunto de *tokens*. Elementos de pouca relevância semântica são removidos, assim como pontuações, caracteres especiais e *stopwords*. Entidades nomeadas são identificadas e removidos. *Stemização* ou *lematização* reduzem a diversidade de *tokens*.**

forma cada sentença contígua em uma lista de *tokens*, permitindo o manuseio individualizado dos *tokens*. Basicamente cada *token* é visto como uma instância de uma sequência de caracteres. Posteriormente na Etapa 2, recursos ortográficos como pontuação, *e.g.* pontos final, de exclamação e de interrogação, e caracteres especiais, *e.g.* números, cifrão e asterisco, são removidos de cada *token*.

Na Etapa 3, eliminam-se as *stopwords*, ou palavras mais frequentes, como conectivos, artigos e pronomes. Essa tarefa em especial tem como base o princípio de que quanto maior a frequência de uma palavra no *corpus*, menos informação relevante a palavra possui. Em seguida na Etapa 4, ocorre a correção ortográfica através da comparação do *token* com seu correspondente mais próximo no dicionário. Executa-se tal procedimento calculando a distância Levenshtein, *i.e.*, o número mínimo de operações necessárias para transformar um nome no banco de dados em outro contido em um dicionário de nomes. O reconhecimento de entidades nomeadas, Etapa 5, identifica principalmente nomes próprios, com subsequente remoção dessas palavras. Na *stemização*, as palavras flexionadas ou derivadas são reduzidas ao seu radical, eliminando possíveis variantes ou

**Tabela 2.3. Características usadas em cada abordagem de detecção de notícias falsas baseadas em processamento de linguagem natural.**

| Tipo | Atributos | [Zhou et al., 2004] | [Fuller et al., 2009] | [Afroz et al., 2012] | [Hauch et al., 2015] | [Monteiro et al., 2018] | [Rashkin et al., 2017] | [Rubin et al., 2016] |
|---|---|---|---|---|---|---|---|---|
| Quantidade | Contagem de caracteres ou *tokens* | x | | x | | x | | |
| | Contagem de palavras | | x | x | x | x | | |
| | Contagem de sentenças | x | x | x | x | x | | |
| | Contagem de verbos | x | x | | x | | x | |
| | Contagem de frases nominais[1] | x | | | | | | |
| | Contagem de substantivos | | | | | x | | |
| | Contagem de *stopwords* | | | | | x | | x |
| | Contagem de adjetivos | | | | | x | | x |
| | Contagem de modificadores[2] | x | x | x | | x | x | x |
| Informalidade | Taxa de erros tipográficos | x | | | x | x | | |
| Complexidade | Média de caracteres por palavra | x | x | x | x | | | |
| | Média de palavras por sentença | x | x | x | x | x | | |
| | Média de orações por sentença | x | | | | | | |
| | Média de pontuações por sentença | x | x | x | | x | | |
| Incerteza | % de verbos modais | x | x | x | x | x | x | |
| | % de termos que indicam certeza[3] | x | x | x | x | | x | |
| | % termos que indicam generalização | | x | | x | | x | |
| | % termos que indicam tendência | | x | x | x | | | |
| | % de números e quantificadores[4] | | | x | x | | x | |
| | # de pontos de interrogação | | | x | | | | |
| Não Imediação | % de voz passiva | x | x | | x | x | | |
| | Pronomes na $1^a$ pessoa do singular | x | x | x | x | x | x | x |
| | Pronomes na $1^a$ pessoa do plural | x | x | x | x | x | x | x |
| | Pronomes na $2^a$ ou $3^a$ pessoa do plural | x | x | x | x | x | | x |
| Diversidade | Diversidade Léxica: % palavras únicas | x | x | x | x | | | |
| | Redundância: % de *function words*[5] | x | x | x | x | | | |
| | % de *content words*[6] | x | x | | x | | | |
| | Entidades nomeadas aleatórias[7] | | | | | | | x |
| Sentimento | % de palavras positivas | x | x | x | x | x | | |
| | % de palavras negativas | x | x | x | x | x | | x |
| | # de pontos de exclamação | | | x | | | | |
| | Teor humorístico/sarcástico | | | | | | | x |

[1]frases cujos núcleos são substantivos. [2]adjetivos e advérbios. [3]e.g. "nunca", "sempre". [4]advérbios de intensidade. [5]palavras com pouco significado atrelado, usadas para expressar relações gramaticais entre palavras ou especificar a atitude ou o humor do falante, e.g., preposições, pronomes, verbos auxiliares, conjunções e artigos. [6]palavras que contém um conteúdo semântico, e.g., substantivos, verbos, adjetivos e a maioria dos advérbios. [7]presença de nomes próprios, nunca antes citados no texto, na última sentença.

plurais. Por fim, com o objetivo de reduzir o processamento desnecessário causado por eventuais redundâncias entre as palavras, seja por flexões ou derivações, é comum a adoção da Etapa 6a ou 6b, sendo respetivamente a *lematização* e a *stemização*. Na tarefa de *lematização*, procura-se eliminar as possíveis variantes ou plurais de uma mesma palavra, reduzindo-as ao mesmo lemas, conhecidos como forma de dicionário. Em contrapartida,

na *stemização* esta redução é feita transformando cada palavra no seu radical [de Oliveira et al., 2020, Navigli, 2009, Manning e Schutze, 1999].

Expandindo o processamento textual a outros estágios linguísticos, há técnicas de PLN que desempenham a tarefa de análise sintática em diferentes graus de complexidade. Em um nível básico, a POS (*part-of-speech*) *tagging* caracteriza-se como uma técnica que retorna apenas a camada mais inferior da árvore de análise, ou seja, a marcação gramatical. Assim, cada a palavra de sentença é atribuído um meta-dado, identificando sua classe gramatical e conjugação. Em um nível intermediário, a técnica *chunking*, também chamada de análise superficial, é uma técnica que analisa frases inteiras, primeiro identificando as partes constituintes das frases (substantivos, verbos, adjetivos) e, em seguida, ligando-as a unidades de ordem superior com significado gramatical discreto. Através dessa técnica, é selecionar estruturas sintáticas específicas como frases nominais, verbais [Manning et al., 2014].

A análise de sentimento, ou mineração de opinião, inspeciona o texto fornecido e identifica a atitude ou emoção dominante no texto através de um grau de polaridade, classificando-o como positivo, negativo ou neutro. Outra propriedade comumente associada à análise de sentimento é a subjetividade, que permite diferenciar frases com alta acidência de opinião, julgamento ou emoção das frases com informações factuais. Normalmente a classificação do sentimento de frases funciona considerando as palavras isoladamente, atribuindo pontos positivos para palavras positivas e pontos negativos para palavras negativas e, em seguida, resumindo esses pontos. A simplicidade dessa lógica resulta em um desprezo pela ordem das palavras implicando perdas semânticas relevantes [Socher et al., 2013]. Modelos online atuais consideram a estrutura da sentença e constroẽm a representação de sentenças inteiras. Assim, esses modelos calculam o sentimento baseados em como as palavras da sentença compõem o significado de frases longas.

Atualmente, dentre as ferramentas mais poderosas de extração de conhecimento sobre textos, a Stanford CoreNLP [9] e a NLTK [10] são as mais conhecidas. Outras ferramentas como a Consulta Linguística e Contagem de Palavras, *Linguistic Inquiry and Word Count* - LIWC) [Pennebaker et al., 2001] destaca-se como um software de análise textual capaz de analisar e quantificar os componentes emocionais, cognitivos e estruturais presentes nos textos. A capacidade do LIWC revelar características latentes de um texto é intimamente dependente do idioma do dicionário de palavras associado ao software. Embora originalmente otimizado para a língua inglesa, atualmente o dicionário LIWC foi traduzido para a língua portuguesa [Balage Filho et al., 2013]. Essas ferramentas são igualmente úteis na extração de características como as vistas na Tabela 2.3.

## 2.6. Representação Vetorial de Textos

Mesmo devidamente padronizada, cada sentença não é passível de ser operada matematicamente, visto que ainda é composta por radicais de palavras e não por valores mensuráveis. Destaca-se que até este momento, as operações realizadas sobre os dados são realizadas em cadeias de caracteres. No entanto, para o cálculo de modelos de aprendizado de máquina são necessários dados que possam ser operados matematicamente.

[9] Disponível em https://stanfordnlp.github.io/CoreNLP/.

[10] Disponível em https://www.nltk.org/.

Para obter uma representação numérica, emprega-se o Modelo de Espaço Vetorial. Esse modelo define que textos, sejam sentenças ou documentos, podem ser interpretados como um espaço vetorial de palavras, em que cada palavra pode ser representada em diferentes padrões, tais como: o binário, Saco-de-Palavras, Frequência do Termo – Inverso da Frequência nos Documentos (*Term Frequency–Inverse Document Frequency*, TF-IDF). Para ilustrar as particularidades de cada padrão de vetorização, considera-se o *corpus* [11] da Tabela 2.4 formado por uma coletânea de quatro documentos, cada um contendo apenas uma única sentença. Devido à unicidade na quantidade de sentenças adotada no *corpus* exemplo, as descrições a seguir mostram as possíveis representações vetoriais em nível de documento e não em nível de sentença, embora isto seja igualmente viável.

**Tabela 2.4. *Corpus* exemplo**

| | |
|---|---|
| **Documento 1 (D1)** | Primeira sentença do corpus |
| **Documento 2 (D2)** | A segunda sentença é curta |
| **Documento 3 (D3)** | A terceira é curta |
| **Documento 4 (D4)** | A quarta sentença é a maior do corpus |

### 2.6.1. Modelo de Espaço Vetorial Binário

Consiste no modelo mais intuitivo de vetorização, em que para cada palavra é atribuído um valor 1 ou 0 de acordo com sua presença ou ausência na sentença. Embora simples, é possível constatar pela Tabela 2.4 que este padrão de representação é pobre do ponto de vista semântico, uma vez que não traz qualquer informação sobre a importância de um termo para o conjunto de textos. No entanto, este modelo de representação é bastante útil para técnicas que aplicam filtros sobre os dados em linguagem natural, já que permite a criação de máscaras binárias de comparação. Ademais, esse modelo de representação requer poucos recursos computacionais para a sua implementação.

**Tabela 2.5. Representação vetorial do *corpus* exemplo da Tabela 2.4 no modelo binário.**

| **Termos** | **primeira** | **quarta** | **a** | **corpus** | **curta** | **do** | **maior** | **segunda** | **sentença** | **terceira** | **é** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **D1** | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| **D2** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| **D3** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| **D4** | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |

### 2.6.2. Modelo de Espaço Vetorial de Saco-de-Palavras

O modelo de Saco-de-Palavras, tradução livre para *Bag-of-Words* (BoW), caracteriza-se como um tipo de modelo vetorial que atribui pesos aos termos, correspondentes ao número de ocorrências observadas do termos no texto. Matematicamente, os vetores dessa representação são expressos conforme a equação

$$V_D = [w_1, w_2, ..., w_{n-1}, w_n], \quad (1)$$

[11] Linguisticamente, um *corpus* é uma coletânea de documentos sobre determinado tema. Um conjunto de *corpus* é denominado *corpora*.

em que $V_D$ é o vetor de pesos $w$ para cada sentença do documento $D$ até o n-iésimo termo.

A Tabela 2.6 ressalta a presença de um peso igual a 2 na última linha da coluna referente ao termo "a". Isto de fato está condizente com a quantidade de vezes que esse termo aparece em D4 na Tabela 2.4, entretanto não reflete a importância semântica para o *corpus* considerado.

**Tabela 2.6. Representação vetorial do *corpus* exemplo da Tabela 2.4 no modelo *Bag-of-Words*.**

| Termos | primeira | quarta | a | corpus | curta | do | maior | segunda | sentença | terceira | é |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **D1** | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| **D2** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| **D3** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| **D4** | 0 | 1 | 2 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |

Este modelo de representação, assim como seu antecessor, sofre do mesmo problema crítico, a presunção de uma igualdade de relevância de todos os termos perante ao *corpus*. Tal suposição pode conferir resultados questionáveis, uma vez que, termos com alta ocorrência em um único documento podem eventualmente ser supervalorizados em uma avaliação baseada na soma total de cada termo no *corpus* [Manning et al., 2010]. Embora esse modelo falhe ao identificar a importância semântica de um termo, o custo computacional para a sua implementação é baixo e permite identificar termos mais prevalentes tanto em um documento quanto em todo *corpora* através de operações simples, soma de colunas, com a matriz de pesos. Destaca-se ainda que o Saco de Palavras é um primeiro passo da implementação de modelos mais complexos.

### 2.6.3. Modelo de Espaço Vetorial Frequência do Termo – Inverso da Frequência nos Documentos

Esse modelo clássico de vetorização é definido pela equação:

$$tfidf_t = tf_{t,d} \times idf_t, \tag{2}$$

em que, para um termo $t$, a Frequência do Termo – Inverso da Frequência, $tfidf_t$, é o produto de duas medidas estatísticas, a **frequência do termo** (TF), $tf_{t,d}$, e o **inverso da frequência nos documentos** (IDF), $idf_t$. Embora o cálculo de frequência do termo ($tf$) siga a mesma lógica apresentada na Seção 2.6.2, o diferencial está na sua ponderação por $idf_t$, uma parcela que remete a quanto esse termo é citado nos demais documentos. Em sua fórmula, expressa na equação:

$$idf_t = log\frac{N}{df_t}, \tag{3}$$

define-se $N$ como a contabilização do número de ocorrências do termo $t$ no conjunto de documentos e $df_t$ considera a frequência do termo $t$ no documento em questão.

**Tabela 2.7. Representação vetorial do *corpus* exemplo da Tabela 2.4 no modelo TF-IDF.**

| Termos | primeira | quarta | a | corpus | curta | do | maior | segunda | sentença | terceira | é |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **D1** | 0.614 | 0 | 0 | 0.484 | 0 | 0.484 | 0 | 0 | 0.392 | 0 | 0 |
| **D2** | 0 | 0 | 0.378 | 0 | 0.467 | 0 | 0 | 0.592 | 0.378 | 0 | 0.378 |
| **D3** | 0 | 0 | 0.408 | 0 | 0.505 | 0 | 0 | 0 | 0 | 0.640 | 0.408 |
| **D4** | 0 | 0.419 | 0.535 | 0.330 | 0 | 0.330 | 0.419 | 0 | 0.267 | 0 | 0.267 |

Essa modificação permite mensurar o grau de relevância semântica de um termo de um documento, em relação a toda coletânea. Como esperado, verifica-se que a Tabela 2.7 possui a mesma quantidade de linhas e colunas do modelo Saco-de-Palavras. Uma variante do TF-IDF original, é conhecido como TF-ISF (*Term Frequency – Inverse Sentence Frequency*), sendo largamente empregada na sumarização de textos em nível de sentença e não em nível de documento como o TF-IDF.

A representação pelo modelo TF-IDF, em relação às demais, é a que carrega maior correlação entre a semântica do termo e o seu peso no espaço vetorial. Essa representação é bastante útil em problemas que visam extrair conhecimento das bases de dados de acordo com a semântica dos documentos [de Oliveira et al., 2020]. No entanto, essa representação é sensível ao uso de sinônimos de palavras comuns. Como sinônimos pouco usuais têm baixa frequência de utilização, mesmo que se refiram a significados comuns amplamente representado por outras palavras, o termo sinônimo passa a ter alto peso na representação TF-IDF, embora possa não ser tão significativo para a representação do dado. Tal anomalia é frequentemente abordada em trabalhos que se baseiam em dicionários de sinônimos, tessauro (*thesaurus*), para normalizar o vocabulário do texto [Jarmasz e Szpakowicz, 2003].

Um ponto importante a ser esclarecido é que, independente da representação aplicada, a dimensão do vetor está vinculada à quantidade restante de palavras distintas contidas em todo o banco de dados, já que várias delas foram removidas durante as etapas descritas na Seção 2.5. As palavras mantidas na sentença são as que carregam significado e, portanto, são as mais importantes para o entendimento da ideia central do texto. Ao se considerar a modelagem de problemas de aprendizado de máquina baseados no processamento de linguagem natural, as palavras remanescentes são as características do conjunto de dados sobre o qual deseja-se fazer o aprendizado.

### 2.6.4. Modelo de Espaço Vetorial de *Feature Hashing*

Diferentemente das representações anteriores, a representação por *Feature Hashing* delimita o tamanho do espaço vetorial com base em posições em uma tabela *hash*. Essa representação usa uma função *hash* para geração dos vetores, a qual mapeia dados de tamanho variável em índices de uma tabela de tamanho fixo, denominada tabela *hash*, ou tabela de dispersão. No contexto da vetorização, os índices resultantes correspondem aos termos analisados. Cada documento pode ser representado a partir dos $N$ índices da tabela, de forma que, para um agrupamento de $M$ documentos, a sua representação matemática é verificada por meio de uma matriz $M \times N$, que identifica a coleção de documentos (*corpora*). A determinação de $N$ é arbitrária, podendo ser menor ou igual

a quantidade total de termos (*tokens*). Entretanto, o valor ótimo de posições deve ser avaliado pois, sendo inferior à quantidade de termos observados nos documentos, a representação pode apresentar inconsistência, uma vez em que há a colisão de termos em índices comuns que podem armazenar informações não correlatas.

Para a representação dos *corpus* exemplo conforme o modelo de *feature hashing* foram selecionados 5 índices, arbitrariamente, considerando um vocabulário de 11 palavras distintas. Assim, os vetores são verificados na Tabela 2.8.

**Tabela 2.8. Representação vetorial do *corpus* exemplo da Tabela 2.4 usando o modelo de *feature hashing*. Diferentemente dos demais modelos, são observadas apenas 5 colunas para representação dos documentos, o que corresponde ao número de índices da tabela de dispersão.**

| *Hashes* | Índice 1 | Índice 2 | Índice 3 | Índice 4 | Índice 5 |
|---|---|---|---|---|---|
| **D1** | 1 | 1 | 1 | 0 | 0 |
| **D2** | 0 | 1 | 1 | 1 | 1 |
| **D3** | 0 | 1 | 1 | 0 | 1 |
| **D4** | 1 | 3 | 1 | 1 | 1 |

Esse modelo de espaço vetorial fornece uma representação compacta dos dados, ao custo de uma menor granularidade semântica, já que cada índice da tabela *hash* pode conter dados não correlacionados semanticamente.

### 2.6.5. Incorporações de Palavras (*Word Embeddings*)

A escolha de tratar as palavras como unidades atômicas, isto é, sem uma conexão semântica entre si, traz simplicidade e robustez ao modelo de espaço vetorial. Apesar de possibilitar uma avaliação da similaridade entre frase ou documento, esses modelos inviabilizam uma medição por palavra, tornando palavras com sentidos próximos como "mar" e "oceano" invisíveis à modelagem vetorial. Uma consequência imediata dessa carência semântica é a dificuldade de lidar com sinônimos. Outra desvantagem é a alta dimensionalidade, um reflexo do caráter esparso dos vetores gerados [Camacho-Collados e Pilehvar, 2018, Mikolov et al., 2013].

Como alternativa, as incorporações de palavras (*words embeddings*) surgem como uma forma de representação distribuída de palavras, idealizada segundo a hipótese distribucional. Nesta hipótese, cada palavra é caracterizada pela sua vizinhança, expressando, portanto uma tendência de palavras com significados semelhantes que aparecerem em contextos similares [Firth, 1957]. Tais representações de palavras podem ser obtidas aplicando modelos preditivos baseados em redes neurais que, quando treinados com grandes volumes de dados textuais, incorporam a semântica das palavras em vetores de baixa dimensão, densos e de tamanho fixo. A principal vantagem da representação vetorial individualizada para cada palavra consiste na preservação das relações semânticas e sintáticas entre palavras, permitindo assim que sinônimos ou palavras minimamente relacionadas sejam mapeadas em vetores semelhantes [Li et al., 2015].

A popularização das técnicas de incorporações de palavras ocorreu através da *Word2Vec* [Mikolov et al., 2013], uma ferramenta que computa a representação vetorial de palavras através de dois modelos possíveis, o Saco de Palavras Contínuo (*Continuous*

*Bag-of-Words*, CBOW) e a *Skip-gram*. Ambos os modelos dividem os textos em dois grupos, palavra-alvo e contexto. Em especial, o contexto é interpretado como um conjunto limitado das palavras que circundam a palavra-alvo. O tamanho dessa limitação, conhecida como janela, define o número de palavras a serem consideradas a esquerda e a direita da palavra-alvo.

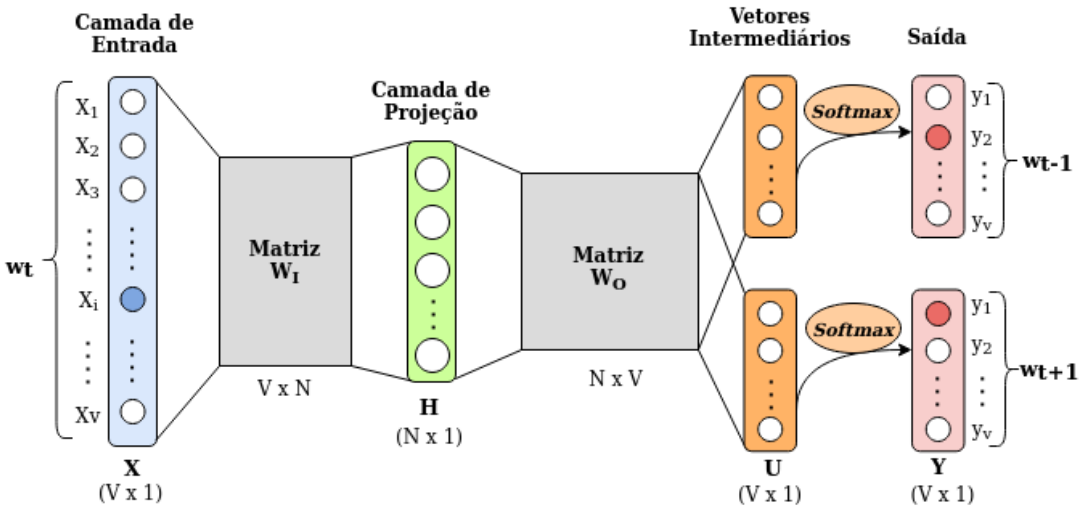



**Figura 2.3. Ilustração da arquitetura do modelo Skip-gram considerando como entrada a palavra-alvo $w_t$ codificada no seu vetor one-hot $X$. Este vetor representa a palavra-alvo como uma sequência de $V$ zeros, exceto por um único valor um na posição $x_i$. Na saída do modelo são obtidos C vetores de distribuição de probabilidade, um para cada palavra do contexto. Com o modelo devidamente treinado, espera-se que as maiores probabilidades de cada vetor $Y$, encontradas nas posições $y_2$ e $y_1$, expressem as palavras de contexto $w_{t-1}$ e $w_{t+1}$.**

A particularidade do modelo Skip-gram está na sua capacidade usar uma palavra-alvo $w_t$ na predição do contexto de palavras $W_t = [w_{t-j}, ..., w_{t+j}]$ que a circunda. Como ilustrado na Figura 2.3, a arquitetura do modelo Skip-gram é composta pelas camadas de entrada e saída, intercaladas por uma camada de projeção. O tamanho da camada de entrada, assim como da camada de saída, está atrelado ao número de palavras $V$ existentes no vocabulário usado no treinamento. Já o tamanho da camada de projeção é determinado com base em um parâmetro $N$ arbitrário, que expressa a dimensão do futuro vetor de palavras gerado $H$ (*word embeddings*). Esta dimensão indica a quantidade de características usadas na representação numérica de cada palavra, sendo portanto inferior à dimensão do vetor original de cada palavra inserido na camada de entrada. A conexão da camada de entrada para a camada de projeção é feita através de uma matriz de pesos $W_I$ de tamanho $V \times N$. Analogamente, a conexão da camada de projeção para a camada de saída é desempenhada pela matriz $W_O$ de tamanho $N \times V$. Como usualmente feito antes do treinamento de redes neurais, ambas as matrizes de peso $W_I$ e $W_O$ são inicializadas com valores aleatórios pequenos. A inserção de uma palavra-alvo na camada de entrada da rede neural inicia com a codificação desta palavra em seu vetor *one-hot*, uma matriz coluna $N \times 1$ usada para distinguir cada palavra em um vocabulário. Este vetor consiste em 0s em todas as posições, com exceção de um único 1 em uma posição usada exclusivamente para identificar a palavra.

No processo de treinamento, a cada iteração são empregados dois algoritmos de aprendizado: de propagação direta (*forward propagation*) e de retropropagação (*back-propagation*). Aplicando primeiramente o algoritmo de propagação direta, o vetor *one-hot* da palavra-alvo de entrada é multiplicado pela matriz de pesos $W_I$ para formar o vetor $H$ da camada oculta. Em seguida, o vetor $H$ é então multiplicado por $W_O$ gerando assim $C$ vetores intermediários idênticos, cada um representando uma palavra de contexto. As saídas do modelo são adquiridas aplicando a cada vetor intermediário a função *softmax*:

$$p(w_{t+j}|w_t) = \frac{exp({v'_{w_o}}^{\top} v_{w_t})}{\sum_{i=1}^{V} exp({v'_{w_i}}^{\top} v_{w_I})}, \tag{4}$$

em que dada a palavra-alvo $w_t$, $v_{w_i}$ é sua linha correspondente na matriz de peso $W_I$ e $v_w$ é sua coluna correspondente na matriz $W_O$. Esta função normaliza o vetor intermediário $U$ composto por $V$ números flutuantes, transformando-o no vetor de distribuição de probabilidade $Y$. Uma vez descoberto o vetor normalizado de probabilidades de cada palavra de contexto, o algoritmo de retropropagação os compara com o vetor *one-hot* da palavra correspondente para assim atualizar as matrizes de peso $W_I$ e $W_O$. Essa atualização ocorre especificamente nos valores da coluna correspondente de $W_O$ e da linha correspondente de $W_I$.

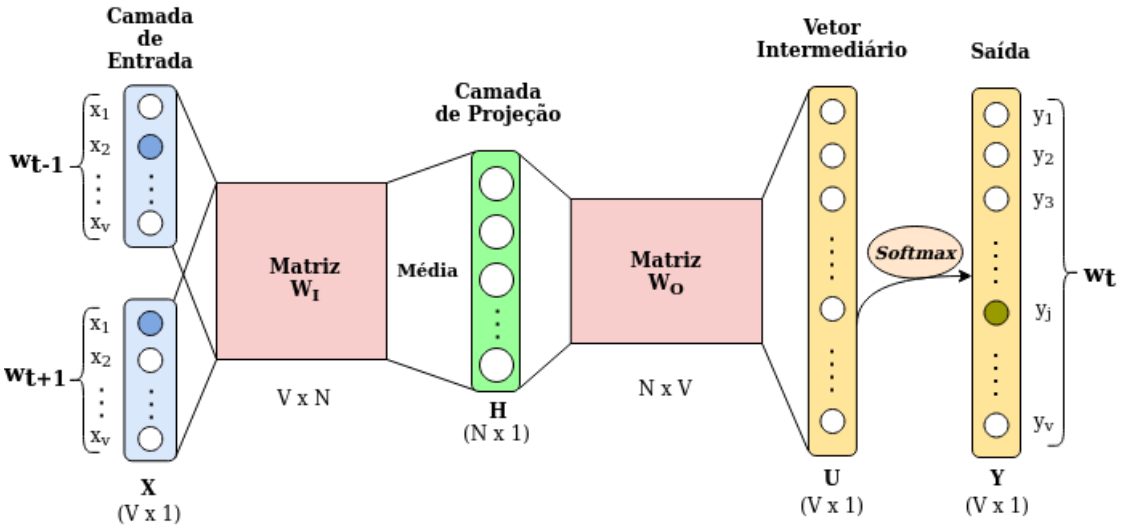



**Figura 2.4. Ilustração da arquitetura do modelo CBOW considerando como entrada as palavras de contexto $w_{t+1}$ e $w_{t-1}$ codificadas em seus vetores *one-hot*. Na saída do modelo é obtido um vetor de distribuição de probabilidade. Com o modelo devidamente treinado, espera-se que a maior probabilidade do vetor $Y$, encontrada na posição $y_i$, expresse a palavra-alvo $w_t$.**

Ao inverter a atuação da palavra-alvo e as palavras de contexto na rede neural, a arquitetura do modelo CBOW torna possível a predição de uma palavra-alvo a partir do contexto de palavras próximas. Como consequência dessa inversão, o modelo admite múltiplas entradas, uma para cada palavra de contexto. Essa multiplicidade de vetores de entrada deriva a necessidade do cálculo da média de seus vetores de palavras correspondentes, estes construídos pela multiplicação dos múltiplos vetor *one-hot* de entrada e

pela matriz W. Uma segunda consequência é a presença uma única função *softmax*, ao contrário das *C* existentes na arquitetura do modelo Skip-gram [Hu et al., 2016]. O modelo CBOW converge de maneira mais rápida em relação ao Skip-gram. Contudo, este apresenta melhores resultados para palavras pouco frequentes em relação àquele.

## 2.7. Aprendizado Sobre Dados de Redes Sociais em Linguagem Natural

Aprendizado de máquina é inerentemente um campo multidisciplinar, focado na construção de programas de computador que melhoram automaticamente com a experiência [Boutaba et al., 2018]. O aprendizado de máquina está relacionado à extração de conhecimento a partir de dados brutos. Os algoritmos de aprendizado de máquina têm como objetivo descobrir como realizar tarefas importantes generalizando as suas operações a partir de exemplos de dados [Domingos, 2012]. Embora existam diferentes definição para o aprendizado de máquina, todas convergem para a ideia de usar algoritmos para obter dados, aprender com eles e então determinar ou prever algum fenômeno. Existem diferentes algoritmos de aprendizado de máquina, cada qual indicado para um tipo de saída desejada. O **aprendizado supervisionado**, também intitulado aprendizagem com exemplos, pressupõe a existência de entradas e saídas marcadas, compondo um conjunto de treinamento, para assim aprender uma regra geral que mapeia as entradas em saídas. Em contraste, o **aprendizado não-supervisionado**, independe de qualquer marcação sobre os dados, forçando o algoritmo a identificar padrões entre as entradas, de modo que as entradas que têm algo em comum sejam agrupadas na mesma categoria. O **aprendizado por reforço** aprende à medida que interage com um ambiente dinâmico e, dessa maneira, qualquer ação que tenha algum impacto no ambiente fornece uma retroalimentação que orienta o algoritmo [Ayodele, 2010].

Esta seção discute as técnicas de redução de dimensionalidade a partir das representações vetoriais extraídas dos dados em linguagem natural, apresenta as métricas de similaridade possíveis de serem usadas sobre os dados, os algoritmos supervisionados e não-supervisionados e, por fim, métricas de avaliação dos algoritmos.

### 2.7.1. Redução Dimensional

Ao utilizar base de dados extensas, ainda mais sendo composta por textos de domínios de conhecimento heterogêneos, é inevitável lidar com vetores de características extremamente longos. Além da elevação da complexidade computacional, o uso de representações vetoriais demasiadamente grandes pode não ser a opção mais adequada. Essa hipótese é confirmada no problema conhecido como "maldição da dimensionalidade", o qual expressa a existência um número ótimo de características que podem ser selecionados em relação ao tamanho da amostra para maximizar o desempenho do aprendizado [Zhai et al., 2014]. Nesse cenário, torna-se conveniente a aplicação de algum procedimento para redução da base de dados, seja pela seleção de características originais ou através de técnicas de redução da dimensionalidade. Esta última alternativa tem o objetivo de encontrar representações vetoriais menos complexas, criando novas características sintéticas a partir das originais.

Redução de dimensionalidade é o processo de derivar um conjunto de graus de liberdade menor que reproduza a maior variabilidade de um conjunto de dados [Zhai et al.,

2014, Andreoni Lopez et al., 2019]. Idealmente, a representação reduzida deve ter uma dimensionalidade que corresponda à dimensionalidade intrínseca dos dados. A dimensionalidade intrínseca dos dados é o número mínimo de parâmetros para contabilizar as propriedades observadas nos dados. Matematicamente, na redução da dimensionalidade, dada a variável aleatória $p$-dimensional x = $(x_1, x_2, \ldots, x_p)$, calcula-se uma representação dimensional inferior a ela, s = $(s_1, s_2, \ldots, s_k)$ com $k \leq p$.

Diferentes abordagens são propostas para reduzir a dimensionalidade, classificadas em lineares ou não lineares. A redução linear da dimensionalidade é uma projeção linear dos dados originais, na qual os dados $p$-dimensionais são reduzidos em dados $k$-dimensionais usando $k$ combinações lineares de $p$ características originais. Dois exemplos importantes de algoritmos de redução de dimensão linear são a análise de componentes principais (*Principal Component Analysis* - PCA) e a análise de componente independente (*Independent Component Analysis* - ICA). O objetivo da PCA é encontrar uma transformação linear ortogonal que maximize a variância das características. O primeiro vetor base do PCA, a componente principal, descreve a direção de maior variabilidade dos dados. O segundo vetor é a segunda melhor descrição e deve ser ortogonal ao primeiro e assim por diante em ordem de importância. De forma semelhante, o objetivo da ICA é encontrar uma transformação linear, na qual os vetores da base sejam estatisticamente independentes e não gaussianos, ou seja, a informação mútua entre duas características no novo espaço vetorial é igual a zero. Ao contrário da PCA, os vetores de base na ICA não são ortogonais nem classificados em ordem. Todos os vetores são igualmente importantes. A PCA é geralmente aplicada para reduzir a representação dos dados. Por outro lado, a ICA normalmente é usada para obter a extração de características, identificando e selecionando as características que melhor se adaptam à aplicação. Métodos não-lineares aplicam transformadas nos dados, levando-os a um novo espaço vetorial, no qual aplicam métodos lineares.

Direcionada especialmente para representações vetoriais derivadas de textos, a **Indexação Semântica Latente** [12] (*Latent Semantic Indexing*, LSI) é uma técnica de redução dimensional baseada na Decomposição em Valores Singulares (*Singular Value Decomposition*, SVD). Sua adaptabilidade a dados de origem textual está atrelada a natureza esparsa dos dados. A LSI propõe construir um espaço "semântico" em que termos e documentos intimamente associados são colocados próximos uns dos outros.

Supondo $A$ como a matriz original $n \times m$, em que termos e documentos são representados em linhas e colunas respectivamente, a aplicação da LSI inicia-se pela adoção de um nível de aproximação $k$. Com isso, $A$ pode ser decomposta da seguinte forma:

$$A \approx A_k = U_k D_k V_k^T, \tag{5}$$

em que $A_k$ é uma aproximação de $A$, composta pelo produto da matriz de termo-conceito $U_k$, a matriz de valores singulares $D_k$ e a matriz de conceito-documento $V_k$. Assim, esta matriz $A_k$ expressa a melhor representação da estrutura semântica do *corpus* original, omitindo todos, exceto os $k$ maiores valores singulares na decomposição. Por tal razão, a LSI

[12] Também referenciada como Análise Semântica Latente (*Latent Semantic Analysis*, LSA) para propósitos para além da área de recuperação da informação.

é também conhecida como SVD truncada [Papadimitriou et al., 2000, Deerwester et al., 1990]. A respeito da escolha de $k$, esta é feita através de testes empíricos, avaliando a taxa de variância dos valores singulares. O valor de $k$ deve ser pequeno o suficiente para permitir uma recuperação rápida da informação e grande o suficiente para capturar adequadamente a estrutura do *corpus*. Para dados textuais, a redução da dimensionalidade é preferível de ser realizada pela técnica LSI em comparação à PCA ou ICA, pois, devido à natureza esparsa dos dados, as técnicas PCA e ICA apresentam resultados menos significativos ou falhos, enquanto a LSI é adequada a dados esparsos.

As técnicas de redução de dimensionalidade carecem de expressividade, pois as características geradas são combinações de outras características originais. Portanto, o significado da nova característica sintética é perdido. Quando há a necessidade de interpretação do modelo, por exemplo, ao criar filtros baseados em textos em linguagem natural, é necessário utilizar outros métodos. As técnicas de **seleção de características** produzem um subconjunto das características originais, que são as melhores representantes dos dados. Assim, não há perda de sentido. Existem três tipos de técnicas de seleção de características [Andreoni Lopez et al., 2019]: *wrapper*, filtro e incorporadas.

Os métodos *wrapper*, também chamados de laço fechado, utilizam diferentes classificadores, como máquina vetor de suporte (SVM), árvore de decisão, entre outros, para medir a qualidade de um subconjunto de características sem incorporar conhecimentos sobre a estrutura específica da função de classificação. Assim, o método avalia subconjuntos com base na precisão do classificador. Esses métodos consideram a seleção de característica como um problema de busca, criando um problema $NP$ -difícil. Uma pesquisa exaustiva no conjunto de dados completo deve ser feita para avaliar a relevância do recurso. Os métodos *wrapper* tendem a ser mais precisos do que os métodos de filtro, mas apresentam um custo computacional mais alto [Andreoni Lopez et al., 2019]. Um método *wrapper* popular por sua simplicidade é o *Sequential Forward Selection* (SFS). O algoritmo começa com um conjunto vazio $S$ e o conjunto completo de todas as características $X$. O algoritmo SFS faz uma pesquisa e gradualmente adiciona características, selecionados $S$ por uma função de avaliação, minimizando o erro quadrático médio (MSE). A cada iteração, o algoritmo seleciona uma característica a ser incluída em $S$ entre as características disponíveis restantes em $X$. A principal desvantagem do SFS é que adicionar uma nova característica ao conjunto $S$ evita que o método remova qualquer característica que tenha o menor erro após adicionar outras. Os métodos de filtro são computacionalmente mais leves do que os métodos de *wrapper* e evitam o sobreajuste. Os métodos de filtro também chamados de métodos de laço aberto, usam heurísticas para avaliar a relevância da característica no conjunto de dados [Chandrashekar e Sahin, 2014]. O algoritmo filtra a característica que preenche o critério heurístico. Um dos algoritmos de filtragem mais populares é o Relief. O algoritmo Relief associa cada característica a uma pontuação, que é calculada como a diferença entre a distância do exemplo mais próximo da mesma classe e o exemplo mais próximo da outra classe. A principal desvantagem desse método é a obrigatoriedade de rotular os registros de dados com antecedência. Relief é limitado a problemas com apenas duas classes, mas ReliefF [Robnik-Šikonja e Kononenko, 2003] é uma melhoria do método Relief que lida com classes múltiplas usando a técnica dos $k$ vizinhos mais próximos. Os métodos incorporados têm um comportamento semelhante aos métodos *wrapper*, usando a precisão de um classificador para avaliar a relevância da

característica. No entanto, os métodos incorporados fazem a seleção de características durante o processo de aprendizagem e usam suas propriedades para orientar a avaliação da característica. Essa modificação reduz o tempo computacional em relação aos métodos *wrapper*. O *Support Vector Machine Recursive Feature Elimination* (SVM-RFE) classifica as características de acordo com um problema de classificação baseado no treinamento de uma máquina vetor de suporte (SVM) com um kernel linear. O elemento com a menor classificação é removido, de acordo com o critério $w$, em forma de eliminação reversa sequencial. O critério $w$ é o valor do hiperplano de decisão no SVM.

### 2.7.2. Métricas de Similaridade e Dissimilaridade

Medidas de similaridade e dissimilaridade desempenham um papel crítico na quantificação da semelhança ou distância semântica, respectivamente, entre textos. Independente dos elementos textuais comparados, caracteres, termos, *strings* ou *corpus*, tais medidas estão constantemente presentes na resolução de problemas de análise de padrões, sejam para fins de sumarização, classificação ou agrupamento de textos. Supondo um um par de vetores $A$ e $B$ não nulos, compostos pela mesma quantidade $n$ de termos, tal que $A = [x_1, x_2, ..., x_n]$ e $B = [y_1, y_2, ..., y_n]$, é possível medir a relação semântica entre eles de diversas formas, tais como Distância Euclidiana, Distância de Manhattan e Similaridade do Cosseno.

A métrica de dissimilaridade conhecido como **Distância de Minkowski** é dada pela equação:

$$Dis(A,B) = \left( \sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}. \tag{6}$$

Tal métrica é uma generalização de outras duas igualmente conhecidas, a **Distância de Manhattan** e a **Distância Euclidiana**, para $p$ igual a 1 ou 2 respectivamente. Obviamente, espera-se que quanto mais próximo de zero for o valor de $Dis$, mais similar $A$ e $B$ serão.

Dentre as métricas de similaridade entre conjunto de termos, destaca-se a **Similaridade do Cosseno** que emprega o conceito de produto interno. Sendo definida entre $[-1, 1]$, valores dessa medida mais próximos ao limite superior simbolizam uma maior proximidade entre os vetores de termos. Matematicamente, a similaridade do cosseno entre $A$ e $B$ é demonstrada pela equação:

$$Sim(A,B) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i-1}^{n} x_i^2} \sqrt{\sum_{i-1}^{n} y_i^2}}. \tag{7}$$

### 2.7.3. Algoritmos supervisionados

A distinção dos algoritmos supervisionados pode ser feita definindo aqueles cujo resultado esperado são variáveis de valor real, intitulados algoritmos de regressão, e aqueles cujo resultado são categorias representadas por valores discretos, conhecidos como algoritmos de classificação. Algoritmos de classificação são o foco do trabalho devido à natureza classificatória das aplicações de processamento de linguagem natural abordadas.

#### 2.7.3.1. Máquina de Vetor de Suporte

A Máquina de Vetor de Suporte, *Support Vector Machine* (SVM), consiste em um tipo de algoritmo classificador linear, baseado no conceito de um plano de decisão que define os limites de decisão. O processo decisório acontece através da geração de um hiperplano multidimensional ótimo que separa as amostras em classes, maximizando a distância entre as classes ou a margem de separação. Tal hiperplano é traçado por um subconjunto de amostras, denominados vetores suporte. O caráter ótimo da separação é assegurado pela definição de uma função *kernel* que minimiza a função erro. Embora seja essencialmente um classificador binário, a SVM é igualmente adaptável a um problema multiclasses, em que divide-se o problema original em subproblemas de classificação binária.

Ao lidar com conjunto de amostras não linear, uma estratégia é adotar o artifício de utilizar uma função de *kernel*, em que uma função encontra um novo espaço dimensional, obrigatoriamente maior que o original, que viabilize a separação usando o hiperplano. Dentre as funções *kernels* mais utilizados estão a Linear, Polinomial, *Radial Basis Function* (RBF) e Sigmoid. A capacidade da SVM ser menos propensa ao sobreajuste (*overfitting*), ou seja, obter uma função de separação de complexidade superior à necessária, está intimamente relacionada ao grau de relevância atribuído a amostras longe do limite de separação. Basicamente, uma vez encontrado o hiperplano, a maioria dos dados que não sejam os vetores de suporte são vistos como redundantes.

O uso de algoritmos supervisionados para a detecção de notícias falsas depende de uma grande base de dados contendo tanto notícias verdadeiras, como falsas. Contudo, isso impõe a limitação de haver uma base rotulada com notícias reais e falsas. As notícias falsas, embora sejam cada vez mais numerosas, são difusas nas redes sociais e tendem a ser voláteis, já que algum período após a disseminação perdem a credibilidade. Uma estratégia para contrapor a limitação no número de notícias falsas para o treinamento dos classificadores é o aprendizado de uma única classe, como o baseado no algoritmo **Máquina de Vetor de Suporte de Classe Única** (*One-class Suport Vector Machine*). A SVM de classe única é um algoritmo de aprendizado supervisionado que deriva um hiperplano de decisão para detecção de anomalias. Novos dados são classificados como semelhantes ou diferentes do conjunto de treinamento. Em contraste com as implementações típicas da SVM, a classe única leva em consideração um conjunto de amostras de treinamento de uma única classe. Qualquer nova amostra que não se encaixe na superfície de decisão definida pelo conjunto de treinamento é considerada uma instância de uma nova classe e, portanto, uma notícia falsa [Perdisci et al., 2006, Gaonkar et al., 2019].

#### 2.7.3.2. Floresta Aleatória

A Floresta Aleatória (*Random Forest*, RF) é um algoritmo popular de classificação ou regressão, que opera construindo múltiplas árvores de decisão durante o processo de treinamento. Durante o treinamento, a RF possibiliza a aplicação do método de *bagging*, que permite treinar repetidamente o algoritmo com o mesmo conjunto de dados, entretanto, selecionando as características aleatoriamente. Ilustrativamente, para um conjunto

de treinamento com $X = x_1, x_2, ..., x_n$ amostras de entrada e respectivos $Y = y_1, y_2, ..., y_n$ amostras de saída o método *bagging* implica a seleção aleatória e com repetição dessa base de dados $K$ vezes. Assim, as árvores são treinadas com a mesma informação, de maneira em que o resultado final é formado pelas predições individuais $m_i$ de cada árvore do conjunto, conforme a equação:

$$\hat{m} = \frac{1}{K} \sum_{k}^{i=1} m_i. \tag{8}$$

Uma vantagem relevante da RF para ao modelo tradicional de árvores de decisão é o fato de não ser considerado todo o conjunto de dados, mas apenas um subconjunto dele. Isto implica uma maior aleatoriedade no modelo, auxiliando na correção do sobreajuste. No mesmo sentido, ao incrementar o número de árvores de decisão na RF, a taxa de erro do conjunto de testes converge para um limite, significando que RF mais povoadas são menos suscetíveis ao sobreajuste [Verikas et al., 2011].

#### 2.7.3.3. k-Vizinhos Mais Próximos

O algoritmo k-Vizinhos Mais Próximos (*k-Nearest Neighbors*, kNN) depende da escolha prévia de um parâmetro $k$, que condiciona o número de amostras vizinhas mais próximas usadas no critério de classificação. A partir de uma amostra ainda não classificada, o algoritmo aplica uma métrica de distância, ou similaridade, entre essa amostra e todas as demais já classificadas. Filtrando as $k$ amostras vizinhas que tiveram as menores distâncias. O algoritmo verifica e contabiliza a quantidade de amostras integrantes em cada classe. Finalmente, a amostra é alocada na classe majoritária dos $k$ vizinhos mais próximos. Essa dependência sobre o valor do parâmetro inicial faz com que o resultado do algoritmo apresente diversas classificações, se $k$ for muito alto, ou apresente amostras ruidosas, se $k$ for muito pequeno. Ao ser obrigado a calcular a distância de cada amostra nova com todas as demais já classificadas, o algoritmo requer um consumo computacional maior, sendo assim não indicado para *corpus* muito grandes [Kadhim, 2019]. Ressalta-se também o alto consumo de memória do algoritmo, já que é necessário carregar todo o conjunto de dados em memória para a comparação com as novas amostras.

### 2.7.4. Algoritmos não-supervisionados

Algoritmos de agrupamento são a forma mais comum de aprendizado não-supervisionado. Apesar de possuírem lógica operacional, caso de uso, escalabilidade e desempenhos distintos, o propósito genérico de usar esses algoritmos é a segregação de termos em grupos (*clusters*) de acordo com suas características semânticas. Esse procedimento de separação em grupos é conhecido como agrupamento.

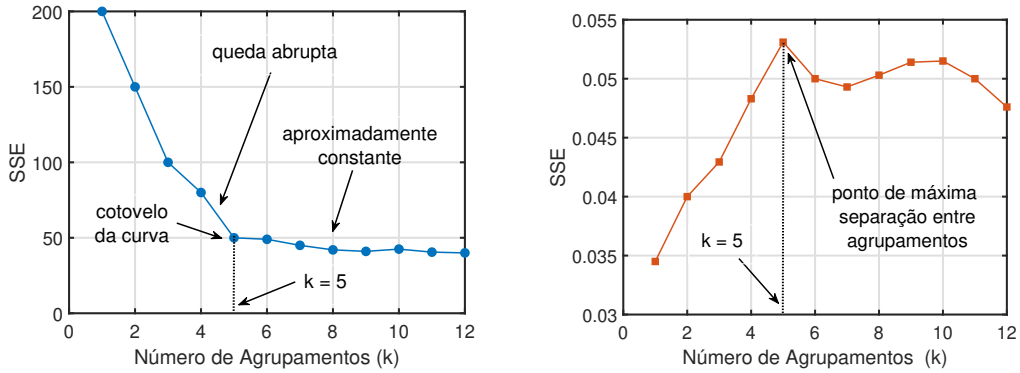
#### 2.7.4.1. Algoritmos Baseados no Particionamento

Essa classificação é dada àqueles algoritmos que são semelhantes no sentido de que cumprem simultaneamente dois critérios no processo de agrupamento de dados. O

primeiro critério expressa a obrigação de ter pelo menos uma amostra em cada agrupamento criado. O segundo refere-se a uma exclusividade de pertencimento, em que cada amostra deve pertencer a somente um agrupamento [Xu e Wunsch, 2005, Fahad et al., 2014].

Um exemplar clássico desse tipo de algoritmo é o ***k-means***, uma heurística capaz de particionar dados em $k$ agrupamentos pela minimização da soma dos quadrados das distâncias em cada agrupamento. Sua lógica de execução, parte da escolha aleatória dos centroides de cada agrupamento seguida do cálculo de distância entre cada amostra e os centroides, segundo uma das métricas de dissimilaridade, ou similaridade, vistas na Seção 2.7.2. Posteriormente cada amostra é alocada no agrupamento cujo centroide está mais próximo. A cada nova amostra alocada a um agrupamento, o centroide é recalculado podendo ocorrer eventuais redistribuições de amostras para outros grupos. O algoritmo finaliza quando cessam essas alterações na alocação das amostras aos agrupamentos.

Outro exemplo é o algoritmo ***k-medoids***, indicado para pequenos conjuntos de dados, e que também particiona os dados em $k$ grupos adotando o critério de minimizar a soma dos quadrados das distâncias em cada grupo. Embora lembre o *k-means*, sua diferença está no fato de escolher efetivamente uma das amostras de entrada como centro dos agrupamentos, não pontos médios como o *k-means*. Essa característica de tomada de decisão se traduz em maior robustez a dados ruidosos e *outliers*, além de uma capacidade de lidar com alta dimensionalidade, útil em representações vetoriais de dados textuais [Xu e Wunsch, 2005,Fahad et al., 2014]. Outra vantagem do *k-medoids* em relação ao *k-means* está no fato de as saídas do *k-medoids* serem mais facilmente interpretadas, dado que os centros dos agrupamentos são amostras reais, ao contrário do *k-means* que fornece um ponto que pode representar uma amostra de dados inviável.




(a) Método *Elbow* (b) Método da Silhueta

**Figura 2.5. Métodos complementares para determinar o número ótimo de agrupamentos. Ambos os métodos idealmente tendem a convergir para um mesmo k, verificado neste exemplo como k=5.**

Contudo ambos os algoritmos, assim como outros, estão sujeitos à uma desvantagem singular: a indeterminação quanto ao número adequado de grupos $k$. A fim de contornar essa indeterminação, são usados dois métodos, *Elbow* e da Silhueta, para analisar previamente a conformidade dos dados a quantidades diferentes de grupos e, assim,

obter um resultado adequado aos dados. Em particular, o *Elbow* mede a compactação dos agrupamentos estabelecendo uma relação entre o número de agrupamentos e sua influência na variação total dos dados dentro do grupo. Graficamente, o melhor valor de $k$ é encontrado identificando o ponto em que o ganho da curva diminui drasticamente, permanecendo aproximadamente constante depois disso. De forma análoga, o método da Silhueta mede a qualidade de um agrupamento. O número ideal de agrupamentos $k$ é aquele que maximiza a silhueta média em uma faixa de valores possíveis para $k$ [Ketchen e Shook, 1996, Rousseeuw e Kaufman, 1990]. A Figura 2.5 mostra um exemplo hipotético de uso dos métodos Cotovelo (*Elbow*) e da Silhueta. Nesse exemplo hipotético, é visto que para o valor $k = 5$ há uma mudança brusca no erro médio quadrático (SSE) interno aos agrupamentos no método Elbow e, para $k = 5$, também há um ponto máximo do erro médio quadrático entre os agrupamentos no método da Silhueta, indicando maior separação entre agrupamentos.

Ressalta-se ainda que há variações dos algoritmos *k-means* e *k-metoids* que consideram graus de pertinência de uma amostra a diversos grupos. Nesses casos, chamados *fuzzy k-means* e *fuzzy k-metoids*, o centro dos agrupamentos são calculados considerando a pertinência parcial de cada amostra aos agrupamentos.

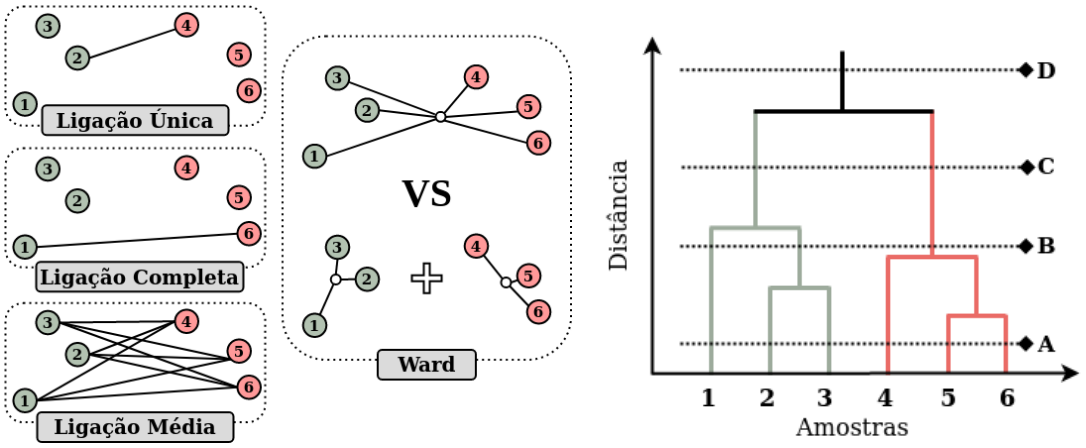
#### 2.7.4.2. Algoritmos Baseados em Densidade

Algoritmos de agrupamento baseados em densidade compartilham uma relação próxima com a abordagem do vizinho mais próximo (*nearest neighbour*). Nesse sentido, um agrupamento, definido como um componente denso conectado, cresce em qualquer direção que a densidade o conduza. Essa lógica de formação dos agrupamentos está diretamente relacionada à principal vantagem desses algoritmos em relação ao grupo dos algoritmos de particionamento, a possibilidade de descobrir agrupamentos com formas arbitrárias, diferente dos agrupamentos tipicamente esféricos retornados pelo algoritmo *k-means*, por exemplo.

Dentre os algoritmos baseados em densidade, o algoritmo de **Clusterização Espacial Baseada em Densidade de Aplicações com Ruído** (*Density Based Spatial Clustering of Application with Noise* – DBSCAN) é o mais popular. Seu intuito é encontrar regiões que satisfaçam uma densidade de pontos mínima estabelecida e que sejam separadas por regiões de menor densidade. Para isso, o algoritmo realiza uma estimativa simples do nível de densidade mínimo, definindo um limite para o número de vizinhos, $minPts$, dentro de um raio $\varepsilon$. Assim, uma amostra com mais de $minPts$ vizinhos dentro desse raio, é considerada um ponto central. Analogamente, uma amostra é considerada como de borda, se dentro de sua vizinhança concentram-se menos amostras que o mínimo definido, porém ainda pertencem à vizinhança de um ponto central qualquer. Por último, amostras que não são alcançáveis por densidade a partir de qualquer ponto central, ou seja, não se configuram nem como pontos centrais nem de borda, são rotulados como *outliers*. Uma desvantagem associada ao seu uso consiste na sua complexidade fortemente polinomial, que requer $\Omega(n^{\frac{4}{3}})$ tempo para convergir, em que $n$ é o tamanho do conjunto de dados [Fahad et al., 2014, Gan e Tao, 2015, Schubert et al., 2017].

### 2.7.4.3. Algoritmos Hierárquicos

Os algoritmos hierárquicos não apenas criam agrupamentos, mas consideram uma lógica multinível e calculam uma representação hierárquica dos dados de entrada. Esta representação é um tipo particular de árvore, em que os nós-folhas expressam dados individuais, e pode ser construída seguindo um método aglomerativo ou divisivo. O método aglomerativo, conhecido também como abordagem *bottom-up*, começa considerando cada amostra como um agrupamento unitário e mescla recursivamente duas ou mais em um novo agrupamento seguindo uma função de ligação escolhida. Tais funções, quando associados à métricas de distância ou de similaridade, definem critérios únicos que elegem os agrupamentos mesclados de cada iteração. A função de ligação única (*single-linkage*), por exemplo, estabelece a união considerando a distância entre as amostras mais próximos de cada agrupamento. De forma oposta, a função de ligação completa (*complete-linkage*) considera a distância das amostras mais distantes entre si cada agrupamento. Paralelamente, a função de ligação média (*average linkage*) calcula a média das distâncias de todas as amostras de um agrupamento em relação a todas as amostras de outro agrupamento. Em especial, o critério de Ward emprega a distância euclidiana na descoberta do par de agrupamentos que minimizam o aumento na variância total interna após a união.




(a) Critérios de Ligação do Método Aglomerativo (b) Dendrograma

**Figura 2.6. (a) Representação bidimensional de diferentes critérios de ligação da considerando 6 amostras já alocadas em dois agrupamentos. b) Dendrograma resultante da aplicação do algoritmo de agrupamento hierárquico sobre as amostras 1-6. O algoritmo emprega o método aglomerativo usando critério de ligação única.**

Por outro lado, o método divisivo, e.g. abordagem *top-down*, inicia com uma estrutura plana em que todas as amostras pertencem ao mesmo agrupamento, ou seja, nível hierárquico. Portanto, a cada iteração, o algoritmo divide um ramo-pai em dois subconjuntos menores, os ramos-filhos. O processo termina quando um critério de parada é atingido, frequentemente, o número $k$ de agrupamentos. No final do algoritmo, é criado um dendrograma de agrupamentos, uma hierarquia de árvore binária [Benavent et al., 2019, de Oliveira et al., 2020, Fahad et al., 2014, Govender e Sivakumar, 2020]. Um possível agrupamento hierárquico considerando a disposição espacial entre as amostras 1-6

da Figura 2.6(a) é ilustrada na Figura 2.6(b). Traçando as retas pontilhadas A-D perpendiculares aos ramos verticais do dendrograma é possível identificar diferentes momentos do processo de agrupamento. Em *A*, nota-se a existência de 6 agrupamentos unitários, ou seja, cada um contendo as amostras. Em *B*, constata-se a existência de 3 agrupamentos: o agrupamento unitário da amostra 1, o agrupamento das amostras 2 e 3, além do agrupamento formado pelas amostras 4, 5 e 6. Em *C*, já é possível identificar o mesmo par de agrupamentos retratados na Figura 2.6(a). Por fim, em *D* verificamos a presença de um único agrupamento superpopuloso, contendo todas as amostras iniciais.

### 2.7.5. Métricas de Avaliação

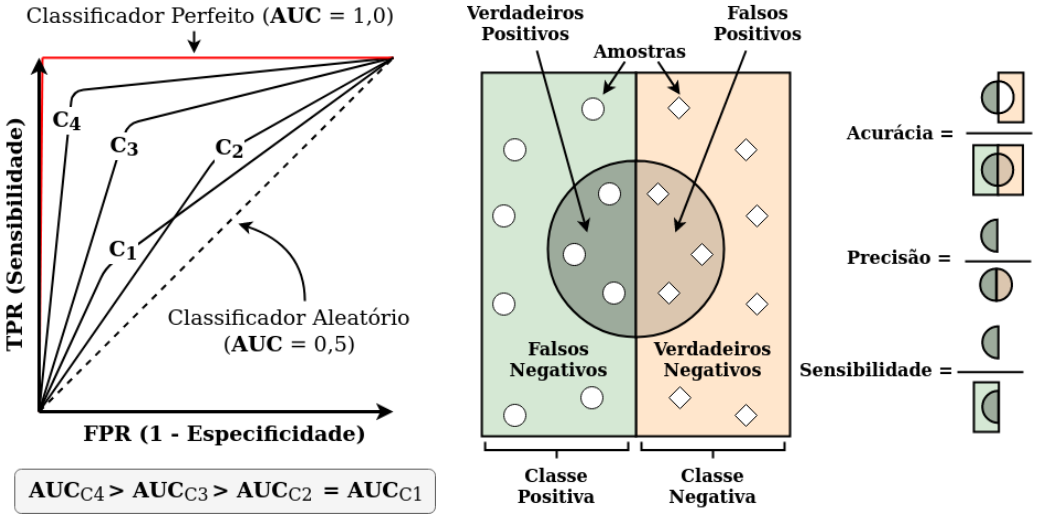
Independente do algoritmo, supervisionado ou não-supervisionado, caso haja o conhecimento prévio sobre dados rotulados com base em uma verdade básica (*ground truth*), torna-se plausível a clara identificação de quantidade de Verdadeiros Positivos (VP), Falsos Positivos (FP), Verdadeiros Negativos (VN) e Falsos Negativos (FN). Tais classificações compõe o cálculo de várias métricas de recuperação de informação, resumidas na Figura 2.7, como:

- **Acurácia** ($A_c$) é definida pela razão do total de amostras classificadas corretamente (VP + VN), pelo número total de amostras (P+N). Para conjunto de dados não-balanceados, uma avaliação de desempenho baseada exclusivamente nesta métrica pode gerar conclusões erradas;
- **Precisão** ($P_r$) é a razão entre, dada uma classe alvo, a quantidade de amostras corretamente classificadas para a classe em questão (VP), pelo conjunto total de predições atribuídas a essa classe, isto é, corretas e incorretas (VP + FP);
- **Sensibilidade** ($S_s$) também conhecida como revocação (*recall*) ou **taxa de verdadeiros positivos** é definida pela razão entre a quantidade de amostras corretamente preditas (VP) para um classe positiva e o total de amostras que pertencem a esta classe, incluindo assim tanto predições corretas quanto as que deveriam ter indicado esta classe (VP + FN). O análogo para a classe negativa é chamado de **especificidade** ou **taxa de verdadeiros negativos**;
- **Medida-$F_1$** ($F_1$-*Score*) relaciona a precisão e a sensibilidade por uma média harmônica expressa por

$$Medida - F_1 = \frac{2}{\frac{1}{P_r} + \frac{1}{S_s}}; \tag{9}$$

  Geralmente, quando maior o valor da medida-$F_1$, melhor a classificação sendo um reflexo do compromisso mútuo entre a precisão ($P_r$) a sensibilidade ($S_s$):

- **Área abaixo da curva ROC** é medida através da curva Característica de Operação do Receptor (ROC), mostrada na Figura 2.7(a), uma representação da razão entre a taxa de verdadeiros positivos (TPR) e a taxa de falsos positivos (FPR), para vários limiares de corte. Essa curva descreve graficamente o desempenho de um modelo de classificação. Sucintamente, quanto maior a área abaixo da curva (mais próxima ao valor unitário), melhor o desempenho do modelo, independentemente do ponto de corte da probabilidade de pertencimento de à classe de cada amostra.




(a) Curvas ROC de Classificadores (b) Métricas de Recuperação da Informação

**Figura 2.7. (a) Curvas ROC de classificadores e comparação da área abaixo da curva ROC (*Area Under the Curve*, AUC). b) Ilustração das métricas acurácia, precisão e sensibilidade em um problema de classificação binária.**

## 2.8. As Soluções para Processamento de Dados em Linguagem Natural em Nuvens Comerciais

Para implementar as diferentes técnicas e algoritmos apresentados em plataformas de computação em nuvem é preciso levar em consideração o tipo de atividade que se espera que o provedor de serviço na nuvem entregue. Os principais provedores de serviço na nuvem públicos são a Amazon Web Service (AWS), a Microsoft Azure e o Google Cloud Platform (GCP). Cada um desses provedores de nuvem oferecem infraestrutura mais simples, como entidades de recursos computacionais análogos a máquinas virtuais, plataformas completas de desenvolvimento, como plataformas que gerenciam a criação de recursos computacionais análogos a máquinas virtuais, até serviços computacionais específicos que não dependem que os usuários gerenciem a infraestrutura de forma alguma, funcionando através de requisições *web* através do protocolo HTTP.

Serviços de inteligência artificial seguem padrão semelhante aos recursos de nuvem em geral, nos quais cada tipo desses recursos tem como público alvo um segmento de profissionais. As plataformas de nuvem dividem a abordagem na qual irão disponibilizar suas ofertas em três camadas: Plataformas de Inteligência Artificial (*Artificial Inteligence Services*), Serviços de Inteligência Artificial (*Artificial Inteligence Platforms*) e Motores de Inteligência Artificial (*Artificial Inteligence Engines*). As próximas subseções detalham cada uma destas três camadas, suas peculiaridades, precificação e público alvo.

### 2.8.1. Plataformas de Inteligência Artificial

As plataformas de inteligência artificial [13], chamadas de *AI Platforms*, estão relacionados com a utilização de alguma solução que facilita o ciclo de vida de uma aplicação

[13] Disponíveis em https://aws.amazon.com/pt/machine-learning/ai-services/.

de inteligência artificial. As *AI Platforms* oferecem o gerenciamento de infraestrutura necessário para treinar e disponibilizar modelos de inteligência artificial e também para fazer a engenharia de características (*feature engineering*) dos dados e lidar com predições em tempo real utilizando algoritmos implementados em estrutura de computação distribuída, como oferecidas pelo arcabouço Apache Spark em sua biblioteca `MLlib` [14]. Esses serviços exigem conhecimento prévio dos algoritmos de inteligência artificial, visto que eles devem ser implementados e o modelo deve ser treinado com dados fornecidos. As *AI Platforms* apoiam esse processo, oferendo recursos para gerenciar o treinamento e a implantação dos modelos, nos quais a implantação de um modelo treinada é realizada em menos de 5 minutos[15].

No caso especifico de uma solução de extração de entidades de um texto, o usuário deverá escolher o algoritmo adequado para solucionar o problema em questão, usar a plataforma oferecida pelas *AI Platforms* para auxiliar na ingestão de dados de forma segura, transformação dos dados e engenharia de características, utilizar as ferramentas para controle e gerência do treinamento dos dados, que pode ser realizada com uma estrutura de recursos computacionais totalmente apartada, com CPU e GPU dedicadas para aquele treinamento, e, por fim, utilizar o serviço que recebe o modelo treinado e cria a infraestrutura juntamente com o servidor HTTP responsável por receber as requisições e retornar as predições.

Os provedores precificam esses serviços com base na quantidade de recurso computacional utilizadas por hora e pela quantidade de armazenamento utilizada, em que alguns deles, como o Amazon Sagemaker na AWS[16], costumam cobrar uma taxa a mais pelo uso da plataforma em si. Esse modelo de cobrança otimiza o uso de recursos, tendo em vista a possibilidade da utilização de recursos computacionais como GPUs em um modelo de aluguel por hora, não necessitando de investimento inicial.

Serviços de *AI Platforms* são recomendados para cientistas de dados e engenheiro de aprendizado de máquina que necessitam implantar aplicações específicas implementando o ciclo completo de desenvolvimento dos modelos, desde a extração dos dados até o monitoramento do modelo final implantado.

### 2.8.2. Serviços de Inteligência Artificial

Os serviços de inteligência artificial [17], chamados de *AI Services*, estão relacionados com a utilização de alguma solução de inteligência artificial implementada previamente pelo provedor de serviço em nuvem. Essas soluções são baseadas em soluções de problemas específicos, como a transformação de texto em voz ou para realizar processamento de linguagem natural extraindo entidades de textos. Esses serviços não exigem conhecimento prévio dos algoritmos de inteligência artificial. Entre os serviços mais comuns estão a automatização de revisões de códigos, a criação de *chatbots*, prevenção de fraudes, traduções em tempo real, transcrição e sintetização de fala, entre outros.

[14]Disponível em https://spark.apache.org/mllib/.

[15]A implantação do modelo treinado se refere ao processo de construção de uma aplicação que utiliza o modelo treinado, recebendo requisições HTTP com os dados de entrada e retornando o resultado da predição do modelo

[16]Disponível em https://aws.amazon.com/pt/sagemaker/.

[17]Disponíveis em https://aws.amazon.com/pt/machine-learning/ai-services/.

Essa categoria de serviços não exige que os usuários treinem os modelos. A utilização é baseada em requisições através do protocolo HTTP, nas quais o usuário envia os dados específicos para a predição e o provedor de serviços na nuvem executa a predição e retorna o resultado. No caso específico de uma solução de extração de entidades de um texto, o usuário deve enviar o texto para o provedor, o provedor analisará o texto com seu modelo proprietário e retornará as entidades encontradas. Por questão de privacidade dos dados, os provedores não armazenam os dados fornecidos pelos usuários ao realizar as predições.

Os provedores precificam esses serviços com base na quantidade de dados que são utilizados como entrada para as predições e a quantidade de requisições feitas pelos usuários. Essa relação de cobrança acarreta que o custo de execução pode ficar mais elevado do que se o usuário realizasse as predições nos próprios modelos. Contudo, o custo é compensado pelo fato do usuário não precisar criar, treinar e gerenciar esses modelos.

Os *AI Services* são recomendados quando o usuário não tem experiência implementando seus próprios algoritmos de inteligência artificial ou para resolução de problemas clássicos e específicos que o provedor já forneça solução pronta que satisfaz os requisitos para a resolução do problema. Os *AI Services* levam a resultados imediatos mediante as requisições *web* através do protocolo HTTP.

### 2.8.3. Motores de Inteligência Artificial

Os motores de inteligência artificial, chamados de *AI Engines*, são a camada dedicada ao uso direto, sem intermédio de uma solução do provedor de serviço em nuvem, de arcabouços de código aberto, como Apache MXNet, Tensorflow e Torch, provendo flexibilidade total aos cientistas de dados e engenheiros de aprendizado de máquina para testar novas implementações de algoritmos, sistemas mais sofisticados que exigem algum recurso de mais baixo nível, como o uso de C e C++. Em geral, é a escolha de pesquisadores que estão implementando um novo modelo otimizado e precisam de total liberdade do sistema operacional, não dependendo de nenhuma implementação prévia do provedor de serviços em nuvem.

Esses serviços funcionam com o provisionamento da infraestrutura requirida pelo usuário para execução do desenvolvimento, treinamento e implantação do modelo. Recursos de GPU, CPU, chegando até mesmo ao nível mais específico de *Field-Programmable Gate Arrays* (FPGAs). Nessa categoria de serviço os provedores de serviço em nuvem alugam recursos computacionais análogos a máquinas virtuais para os usuários.

Os provedores precificam esses serviços analogamente às *AI Platform*, exceto pelo custo extra de licença para a utilização de algumas plataformas em particular. *AI Platforms* são recomendadas para cientistas de dados e engenheiro de aprendizado de máquina que precisam escrever aplicações do zero, com total liberdade de escolha de qualquer tipo de arcabouço e linguagem de programação, com a desvantagem de ter que gerenciar a infraestrutura total requisitada, enquanto *AI Engines* são recomendados para usuários que possam utilizar os arcabouços específicos já implantados na nuvem.

## 2.9. As Iniciativas de Pesquisa

Diversas atividades de pesquisa estão ativas e buscam caracterizar e mitigar os desafios causados pelas notícias falsas. Uma definição inicial sobre as notícias falsas é feita por Lazer *et al.*. No artigo, é abordada a historia das notícias falsas começando pela difamação na Primeira Guerra Mundial até o impacto das notícias falsas durante a eleição presidencial dos Estados Unidos em 2016 [Lazer et al., 2018]. Grinberg *et al.* aprofundam no impacto das notícias falsas durante as eleições de 2016, analisando as mensagens da rede social *Twitter* [Grinberg et al., 2019]. Os autores coletaram *tweets* enviados por 16.442 contas ativas durante a temporada eleitoral de 2016, de 1º de agosto até 6 de dezembro de 2016. Os resultados mostram que os grupos de maior idade, entre 60 e 80 anos, com afinidade politica de direita ou extrema direita são mais propensos à distribuição e ao compartilhamento de notícias políticas falsas. A recente pandemia da Doença Infecciosa por Corona Vírus de 2019 (COVID-19) também é um evento no qual foram disseminadas grande quantidade de notícias falsas. Estudos recentes mostram a correlação entre o uso de mídia social e a desinformação durante a pandemia [Pennycook et al., 2020, Van Bavel et al., 2020].

A detecção de notícias falsas é estudada sob várias perspectivas como Aprendizado de Máquina, Mineração de Dados e Processamento de Linguagem Natural. O Saco-de-Palavras e as frequências de categorias são utilizadas para o treinamento de classificadores como as Maquinas de Vetores Suportes (*Support Vector Machines* - SVM) e modelos bayesianos ingênuos [Poddar et al., 2019]. Uma vez que o modelo matemático é treinado a partir de exemplos conhecidos das duas categorias, notícia falsa ou não, é possível prever instâncias futuras com base em agrupamento numérico e distâncias. O uso de diferentes métodos de agrupamento e funções de distância entre os pontos de dados é uma das bases do algoritmo do SVM. Por outro lado, o algoritmo bayesiano ingênuo faz classificações com base em evidências acumuladas da correlação entre uma determinada variável, como a sintaxe, e as outras variáveis presentes no modelo.

Shu *et al.* fazem uma revisão da detecção de notícias falsas nas mídias sociais de uma perspectiva de mineração de dados, incluindo caracterização de notícias falsas sobre psicologia e teorias sociais, algoritmos existentes, métricas de avaliação e conjuntos de dados representativos [Shu et al., 2017]. *Fake News Tracker* é uma solução para coleta de dados, visualização interativa e modelagem analítica para detecção de notícias falsas. A solução utiliza técnicas de Processamento de Linguagem Natural [Shu et al., 2019].

Alguns trabalhos apresentam técnicas e desafios sobre a detecção de notícias falsas. Zhou e Zafarani identificam e detalham as teorias fundamentais relacionadas em diferentes disciplinas para a detecção de notícias falsas [Zhou e Zafarani, 2018]. Sharma *et al.* discutem os métodos e técnicas existentes aplicáveis à identificação e à mitigação de notícias falsas, com foco nos avanços significativos em cada método e suas vantagens e limitações [Sharma et al., 2019]. Bondielli e Marcelloni fazem um levantamento da literatura sobre as diferentes abordagens para a detecção automática de notícias falsas e rumores [Bondielli e Marcelloni, 2019]. Os autores destacam várias abordagens adotadas para coletar dados de notícias falsas e rumores.

Oshikawa *et al.* apresentam uma comparação dos métodos usados na detecção de notícias falsas usando Processamento de Linguagem Natural (PLN) [Oshikawa et al.,

2018]. De forma semelhante, Sharma *et al.* analisam a revisão da literatura sobre PLN aplicado em notícias falsas, ressaltando a comparação entre as diferentes técnicas de aprendizado de maquina, aprendizado profundo e outras técnicas [Sharma et al., 2019]. Deepak and Chitturi compararam diferentes tipos de redes neuronais na detecção de notícias falsas [Deepak e Chitturi, 2020]. Feng *et al.* propõem uma rede neural convolucional de dois níveis com gerador de resposta do usuário, em que a rede neural captura informações semânticas do texto, representando-as no nível de frase e de palavra, e o gerador de resposta de usuário aprende um modelo da resposta do usuário ao texto da notícia [Qian et al., 2018].

## 2.10. Desafios e Oportunidades de Pesquisa

Embora as pesquisas na identificação, detecção e mitigação da propagação de notícias falsas estejam em pelo desenvolvimento, alguns dos principais desafios no combate às notícias falsas são listados a seguir [Sharma et al., 2019].

- **Grandes interesses e a pluralidade de atores envolvidos.** Devido ao volume que a propagação de notícias falsas atinge em redes sociais em um período curto, as notícias falsas representam uma ameaça às fontes tradicionais de informações, como a impressa tradicional. O espalhamento de notícias falsas ocorre como um evento distribuído e, então, envolve múltiplas entidades e plataformas tecnológicas. Assim, há uma crescente dificuldade de estudar e projetar estratégias computacionais, tecnológicas e de negócios de combate às notícias falsas sem que haja o comprometimento da rapidez e do acesso colaborativo a informações de alta qualidade.

- **Intensão maliciosa do adversário.** O conteúdo das notícias falsas é projetado para dificultar a identificação por humanos das notícias falsas, explorando suas habilidades cognitivas, emoções e preconceitos ideológicos. Além disso, é desafiador para métodos computacionais detectar notícias falsas, pois a forma como as notícias falsas são apresentadas é semelhante à de notícias verídicas e, por vezes, as notícias falsas usam artifícios para dificultar a identificação da fonte ou falsificam a verdadeira fonte da notícia.

- **Suscetibilidade e falta de conscientização do público.** O usuário de redes sociais está sujeito a uma grande quantidade de informações de origens duvidosas, desde informações com cunho humorístico, como sátiras, até informações com o intuito de enganar o consumidor de informações se passando por notícias verídicas. Contudo, o usuário de redes sociais não é capaz de diferenciar uma notícia falsa de uma verídica apenas pelo conteúdo. O usuário não dispõe de informações sobre a credibilidade da fonte ou padrões de propagação da notícia na rede. Assim, para aumentar a conscientização pública, vários artigos e campanha publicitárias são veiculados para fornecerem dicas sobre como diferenciar notícias verídicas de falsas. Por exemplo, a Universidade de Portland, nos Estados Unidos, disponibiliza um guia para a identificação de desinformação (notícias falsas)[18].

[18]Disponível em https://guides.library.pdx.edu/c.php?g=625347&p=4359724.

- **Dinâmica de propagação.** A propagação de notícias falsas em mídia social complica a detecção e a mitigação, pois as informações falsas podem facilmente alcançar e afetar um grande número de usuários em pouco tempo. A informação é transmitida de maneira rápida e fácil, mesmo quando sua veracidade é duvidosa [Friggeri et al., 2014]. A verificação da veracidade deve ser realizada de forma ágil, mas também deve considerar os padrões de propagação da informação ao longo da rede [Meel e Vishwakarma, 2020].

- **Mudanças constante das características das notícias falsas.** Os desenvolvimentos na identificação automatizada de notícias falsas também impulsionam a adaptação da geração de novos conteúdos de desinformação para evitarem de serem classificados como tal. A detecção de notícias falsas baseada em estilo de escrita, diferenciando notícias falsas e verdadeiras por uma análise baseada no processamento de linguagem natural, é uma das principais alternativas usadas devido aos desafios não resolvidos na automatização da verificação de fatos a partir de bases de conhecimento pré-definidas. Assim, abordagens atuais de identificação de notícias falsas baseadas no conteúdo focam na extração de fatos diretamente do conteúdo da notícia e a posterior verificação dos fatos contra bases de conhecimento [de Oliveira et al., 2020].

- **Ataques ao aprendizado por linguagem natural.** Zhou *et al.* argumentam que o uso de processamento de linguagem natural para a identificação de notícias falsas é vulnerável a ataques ao aprendizado de máquina em si [Zhou et al., 2019]. Zhou *et al.* identificam três ataques: a distorção de fatos, a troca entre sujeito e objeto; e a confusão de causas. A distorção de fato consiste em exagerar ou modificar algumas palavras. Elementos textuais, como personagens e tempo, podem ser distorcidos para levar a uma interpretação falsa. A troca entre sujeito e objeto tem como objetivo confundir o leitor entre quem pratica e quem sofre a ação relatada. O ataque de confusão de causa consiste em criar relações causais inexistentes entre dois eventos independentes ou cortar partes de uma história, deixando apenas as partes que o atacante deseja apresentar para o leitor [Zhou et al., 2019].

As oportunidades de pesquisa na identificação e mitigação de notícias falsas focam na detecção rápida ou em tempo real da fonte, no controle da propagação das informações falsas e na redução do impacto das notícias falsas na sociedade. Conjuntos de dados coletados em tempo real, detecção automática de rumores e localização da fonte original são questões de pesquisa desafiadoras [Meel e Vishwakarma, 2020]. A seguir destacam-se as principais oportunidades de pesquisa e desenvolvimento de soluções para o combate às notícias falsas.

- **Extração de características mais significativas.** Determinar as características mais eficazes para detectar notícias falsas de múltiplas fontes de dados é uma oportunidade de pesquisa em aberto. Fundamentalmente, existem duas fontes de dados principais: o conteúdo das notícias e contexto social [Shu et al., 2017]. Da perspectiva de conteúdo de notícias, técnicas baseadas em processamento de linguagem natural e extração de características podem ser usadas para extrair informações do

texto. Técnicas de incorporação, como incorporação de palavras (*word embedding*) e redes neurais profundas são foco de pesquisas atuais para a extração de características textuais e têm o potencial para aprender melhores representações para os dados. Características visuais extraídas das imagens também são indicadores importantes para notícias falsas. O uso de redes neurais profundas é uma oportunidade de pesquisa na extração de características visuais para a detecção de notícias falsas [Sharma et al., 2019, Meel e Vishwakarma, 2020].

- **Detecção em diferentes plataformas e diferentes domínios.** Devido ao fato dos usuários utilizarem diferentes redes sociais, as notícias falsas e boatos se espalham nas diferentes plataformas, dificultando a localização da origem da notícia ou do boato. O rastreamento da origem da informação falsa entre plataformas distintas de redes sociais é uma oportunidade de pesquisa. Para tanto, devem ser considerados diversos aspectos da informação. Contudo, a maior parte da abordagem existente se concentra apenas em uma das formas de detecção da informação falsa: análise de conteúdo, da propagação, do estilo, entre outras. A análise deve considerar, então, diferentes domínios de atributos, como tópicos, sítios *web*, imagens e URLs [Meel e Vishwakarma, 2020].

- **Identificação de câmaras de eco e ponte entre as câmaras.** A mídia social tende a formar câmaras de eco em comunidades em que usuário têm visões e ideologias semelhantes. Os usuários têm suas visões reforçadas e não estão cientes das crenças opostas. Portanto, pesquisas são necessárias para identificar câmaras de eco conflitantes e ligar as câmaras com posições opostas para que os usuários sejam confrontados com visões distintas. Isso também ajuda na descoberta da verdade, fazendo os usuários pensarem criteriosamente e racionalmente em múltiplas dimensões [Meel e Vishwakarma, 2020].

- **Desenvolvimento de modelos de aprendizado de máquina.** Há a necessidade de pesquisa no desenvolvimento de modelos de aprendizado em tempo real, tais como aprendizado incremental e aprendizado federado, capazes de aprender com artigos verificados manualmente e fornecer detecção em tempo real de novos artigos com informações fraudulentas. Outro ponto importante é o desenvolvimento de modelos não-supervisionados em que os algoritmos aprendem com dados reais e, então, artigos que fogem do comportamento de dados reais são classificados como falsos. Há ainda uma escassez de conjuntos de dados específicos para notícias falsas. A falta de conjuntos de dados de larga escala publicamente disponíveis implica a carência de testes (*benchmarks*) para a comparação de desempenho entre algoritmos diferentes [Meel e Vishwakarma, 2020].

- **Desenvolvimento de estruturas de dados capazes de lidar com a estrutura de rede complexa e dinâmica.** A complexidade e a dinamicidade das estruturas de relacionamento em redes sociais tornam a tarefa de identificação e rastreamento de publicações mais complicadas. Assim, há a necessidade de pesquisa para o desenvolvimento de estruturas de dados complexas que reflitam a dinamicidade das relações em redes sociais para permitir a extração de conhecimento acerca da propagação de informações falsas na rede [Meel e Vishwakarma, 2020].

## 2.11. Atividade Prática

Esta seção consolida a uma pluralidade de conceitos teóricos abordados no capítulo através de uma atividade prática do processo de identificação de notícias falsas em redes sociais, empregando a linguagem `Python`. O processo ocorre na rede social *Twitter* e é representado na Figura 2.11. O processo inclui (i) a coleta de notícias, falsas e verdadeiras, empregando a interface de programação de aplicação (*Application Programming Interface* - API) do *Twitter* para efetuar o *web scraping*[19]; (ii) o processamento textual e vetorização do conteúdo das notícias, empregando PLN juntamente com técnicas de representação vetorial de textos; (iii) a aplicação eficiente de algoritmos de detecção, alcançado pela incorporação de técnicas de redução de dimensionalidade; e (iv) a avaliação da eficiência e qualidade da detecção, tendo como parâmetros as métricas de recuperação de informação.

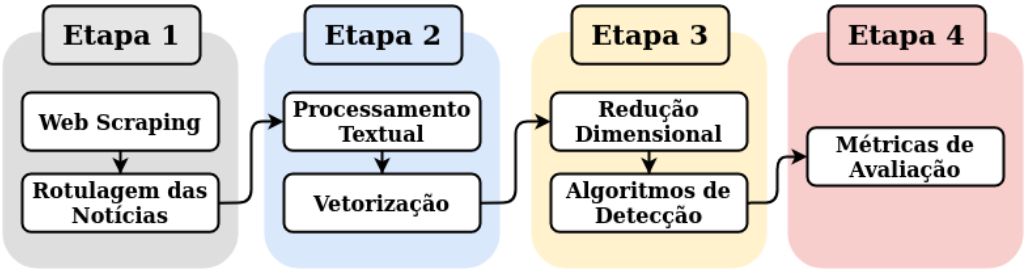



**Figura 2.8. Fluxograma do processo de identificação de notícias falsas desenvolvido na atividade prática. A primeira etapa compreende a formação da base de dados formada por notícias falsas e legítimas. Na segunda etapa são aplicadas técnicas de processamento de linguagem natural e vetorização. Na terceira etapa, após uma redução dimensional, a representação vetorial das notícias é submetida a algoritmos de detecção. A quarta etapa concentra-se em avaliar a qualidade da detecção segundo métricas de recuperação de informação.**

Como primeira etapa da atividade prática, a composição da base de dados inclui tanto notícias verdadeiras quanto falsas, extraídas através do *web scraping* em contas específicas do *Twitter*. Para obtenção do conteúdo dessas contas, é preciso desenvolver um *script*[20] em `Python` que acessa a API do *Twitter* usando credenciais de desenvolvedor. Em posse dessas credenciais, a biblioteca `tweepy`[21] permite a extração contínua do conteúdo textual dos *tweets* de qualquer perfil aberto na rede social. Contudo, além das limitações temporais igualmente enfrentadas por Barreto *et al.*, como o número máximo de requisições por janela de tempo de 15 minutos [Barreto et al., 2014], há também uma limitação da quantidade de *tweets* históricos passíveis de serem coletados. Dessa maneira, a obtenção de *tweets* é restrita a um período de até, no máximo, dois meses passados a contar pela data de execução do *script*. Devido às restrições de acesso à plataforma, uma solução é diversificar as fontes de busca por notícias verdadeiras, coletando *tweets* de outras fontes jornalísticas. A escolha de perfis de veículos jornalísticos como fonte de conteúdo verdadeiro parte da premissa que estes perfis são menos susceptíveis

[19]Também conhecida como coleta, ou raspagem de dados, é uma forma de mineração capaz de extrair o conteúdo de sítios da *web* para uma posterior análise.

[20]Disponível em https://github.com/nicollasro/FakeNewsDetection.

[21]Disponível em https://www.tweepy.org/.

a compartilhar conteúdo de procedência duvidosa do que contas de usuários individuais. Analogamente, a coleta de *tweets* comprovadamente falsos pode ser feita extraindo o conteúdo de perfis dedicados checagem de fatos. Nesses perfis é possível encontrar *tweets* falsos previamente verificados por jornalistas.

Os *tweets* extraídos, uma vez armazenados e rotulados entre falso e verdadeiro, são submetidos à sequência básica de processamento de linguagem natural descrita na Seção 2.5. Em `Python`, várias técnicas de PLN são facilmente implementáveis por funções da biblioteca `NLTK` [22]. Nos procedimentos seguintes, diversas funções e classes de módulos específicos da biblioteca `Scikit-learn` [23] serão empregadas. Em especial, na vetorização usa-se o módulo `FeatureExtration`, que inclui a classe `TfidfVectorizer` capaz de converter a coleção de *tweets* já processados textualmente em uma matriz contendo os valores TF-IDF de cada palavra. Diante da alta dimensionalidade da matriz TF-IDF adquirida, torna-se conveniente empregar o módulo `Decomposition`, que dispõe de diferentes algoritmos de decomposição matricial predominantemente usados na redução dimensional. Devido ao caráter esparso da matriz TF-IDF, a etapa de redução dimensional da atividade prática é desempenhada pelas funções da classe `TruncatedSVD`, que executam a decomposição em valores singulares truncada, (*Singular Value Decomposition* - SVD), também conhecida como Indexação Semântica Latente (*Latent Semantic Indexing* - LSI). Como alertado na Seção 2.7.1, a configuração do nível de aproximação $k$, representado na classe pelo parâmetro `n_components`, precisa ser cuidadosamente escolhido observando o atributo `explained_variance_ratio_`, que expressa o percentual de variância entre as componentes geradas e as originais.

Após obter uma representação vetorial eficientemente reduzida dos *tweets* extraídos, é possível aplicar três exemplos de metodologias diferentes [de Oliveira et al., 2020], capazes de detectar padrões de escrita característicos de notícias falsas. A primeira metodologia, chamada Redução com Treinamento, prevê o treinamento e classificação dos *tweets* coletados usando o algoritmo Máquina de Vetor de Suporte de Classe Única (*One-class Suport Vector Machine*), implementado na classe `OneClassSVM` do módulo `SVM`. A segunda metodologia, chamada Transformação Matricial, introduz uma transformação matricial antes do processo de treinamento com o algoritmo SVM de classe única. Tal transformação é produzida multiplicando a matriz TF-IDF reduzida por uma versão transposta da mesma, porém submetida ao algoritmo de agrupamento *k-means*. A aplicação desse algoritmo no *script* em `Python` depende do uso da classe `KMeans` do módulo `Cluster`. A terceira metodologia, denominada Limite Radial, expande o processo de detecção para um cenário estatístico, partindo da hipótese de que notícias verdadeiras e falsas têm distribuição de probabilidades distintas. A última etapa da atividade prática consiste na avaliação da qualidade da detecção de notícias falsas a partir das métricas de recuperação da informação descritas na Seção 2.7.5. Incorporando ao *script* desenvolvido algumas funções do módulo `Metrics`, tais como `accurary_score`, `precision_score`, `recall_score` e `roc_curve`, é possível obter respectivamente os valores de acurácia, precisão, sensibilidade e da curva ROC. Ao final da atividade prática é esperado que os resultados se assemelhem aos das Figuras 2.9 e 2.10.

[22] Disponível em https://www.nltk.org/.

[23] Disponível em https://scikit-learn.org/stable/.

Os resultados da primeira metodologia, Redução com Treinamento, demonstram um desempenho mais homogêneo entre as métricas, destacando-se principalmente pela alta acurácia e porcentagem de sensibilidade mais expressiva dentre as três metodologias. Já nos resultados referentes à segunda metodologia, Transformação Matricial, percebe-se uma clara predominância na habilidade de classificar qualitativamente notícias como sendo falsas, expressa pela alta precisão. Em contrapartida, detém baixas porcentagens de acurácia e sensibilidade, essas possivelmente fruto de perdas de características, importantes na diferenciação das notícias, impostas por dois níveis de redução dimensional – LSI e *k-means*. Analogamente à anterior, a terceira metodologia, Limite Radial, apresenta o mesmo caráter preciso na identificação de notícias falsas embora constate-se uma depreciação nos seus níveis de sensibilidade.

A Figura 2.10 apresenta uma comparação entre as curvas ROC de cada metodologia de detecção. O bom resultado obtido pela metodologia de transformação matricial, área abaixo da curva de 0,82, baseia-se no fato de que ao realizar o agrupamento com o *k-means*, a dimensionalidade dos dados é substancialmente reduzida, permitindo que a SVM de classe única defina uma hiper-superfície mais ajustada aos dados.

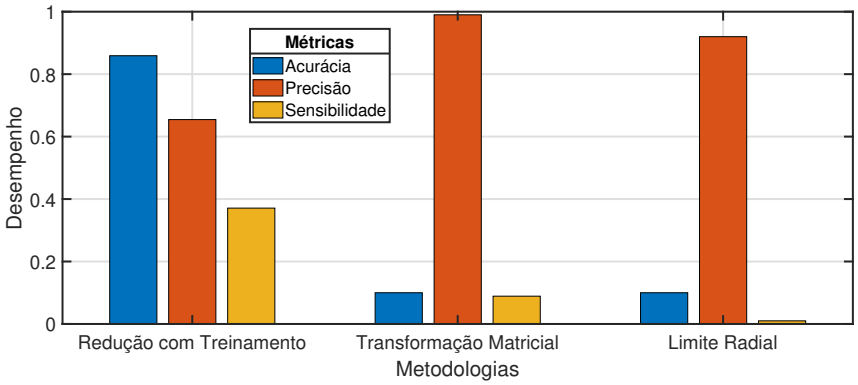



**Figura 2.9. Comparação das metodologias revela um comportamento diverso porém ligeiramente complementar nos níveis de recuperação de informação.**

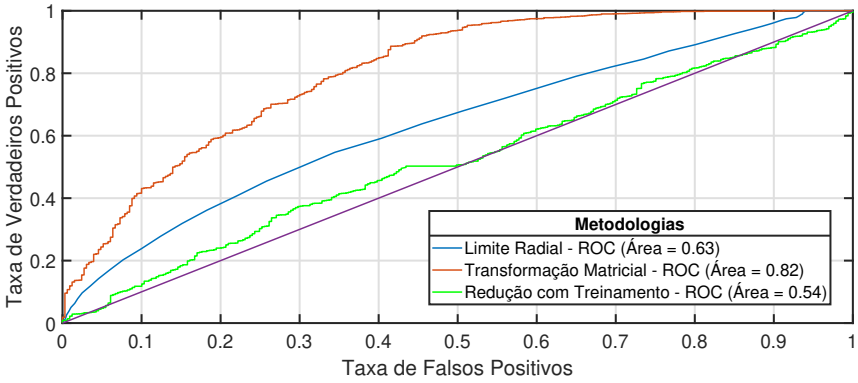



**Figura 2.10. As curvas ROC refletem o desempenho de um sistema classificador binário à medida que o seu limiar de discriminação varia. Dentre as metodologias, a transformação matricial apresenta o melhor desempenho, visto que possui a maior área acima da reta.**

## 2.12. Considerações Finais

Nesse minicurso foram apresentados as definições, características e o processo de disseminação de notícias falsas. Em seguida, foram discutidos os métodos tradicionais para a detecção de notícias falsas. Foram comparados os conjuntos de dados de referência mais recentes. Com base em trabalhos da literatura, foi proposto a utilização do Processamento de Linguagem Natural (PLN) na de detecção de notícias falsas. Foram mostrados como o PLN pode ser usado sobre redes sociais e uma comparação com os diferentes métodos de aprendizado de máquina utilizados. Além disso, questões em aberto e desafios também são destacados para explorar as oportunidades de pesquisa em potencial. Esse trabalho é útil para os pesquisadores compreendam os diferentes componentes da comunicação digital *online* de uma perspectiva social e técnica. Divulgação de notícias falsas em várias plataformas multilíngues, estrutura de rede complexa e dinâmica, grandes volumes de dados em tempo real não rotulados e detecção precoce de boatos são alguns problemas desafiadores que ainda não foram resolvidos e necessitam mais pesquisas. Finalmente, a atividade prática desenvolvida mostrar a viabilidade na detecção de notícias falsas. Melhorar a confiabilidade e o futuro do ecossistema de informações *online* é uma responsabilidade conjunta da comunidade científica, formuladores de políticas digitais, administração e da sociedade em geral.

## Referências

[Afroz et al., 2012] Afroz, S., Brennan, M. e Greenstadt, R. (2012). Detecting hoaxes, frauds, and deception in writing style online. Em *2012 IEEE Symposium on Security and Privacy*, p. 461–475. IEEE.

[Andreoni Lopez et al., 2019] Andreoni Lopez, M., Mattos, D. M. F., Duarte, O. C. M. B. e Pujolle, G. (2019). A fast unsupervised preprocessing method for network monitoring. *Annals of Telecommunications*, 74(3):139–155.

[Ayodele, 2010] Ayodele, T. O. (2010). Types of machine learning algorithms. Em *New advances in machine learning*. IntechOpen.

[Balage Filho et al., 2013] Balage Filho, P., Pardo, T. A. S. e Aluísio, S. (2013). An evaluation of the brazilian portuguese liwc dictionary for sentiment analysis. Em *Proceedings of the 9th Brazilian Symposium in Information and Human Language Technology*.

[Barreto et al., 2014] Barreto, H. F., Campista, M. E. M. e Costa, L. H. M. (2014). Spammers no twitter: Quando contatos deixam de ser bem-vindos. Em *Workshop de Redes P2P, Dinâmicas, Sociais e Orientadas a Conteúdo (Wp2p+ 2014) - SBRC 2014*, volume 1, p. 23–36.

[Benavent et al., 2019] Benavent, X., Castellanos, A., de Ves, E., Garcia-Serrano, A. e Cigarran, J. (2019). Fca-based knowledge representation and local generalized linear models to address relevance and diversity in diverse social images. *Future Generation Computer Systems*, 100:250–265.

[Bird et al., 2009] Bird, S., Klein, E. e Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edição.

[Bondielli e Marcelloni, 2019] Bondielli, A. e Marcelloni, F. (2019). A survey on fake news and rumour detection techniques. *Information Sciences*, 497:38 – 55.

[Boutaba et al., 2018] Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F. e Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):16.

[Camacho-Collados e Pilehvar, 2018] Camacho-Collados, J. e Pilehvar, M. T. (2018). From word to sense embeddings: A survey on vector representations of meaning. *Journal of Artificial Intelligence Research*, 63:743–788.

[Chandrashekar e Sahin, 2014] Chandrashekar, G. e Sahin, F. (2014). A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28.

[Chen et al., 2015] Chen, Y., Conroy, N. J. e Rubin, V. L. (2015). Misleading online content: Recognizing clickbait as false news. Em *Proceedings of the 2015 ACM on Workshop on Multimodal Deception Detection*, p. 15–19. ACM.

[Clark et al., 2012] Clark, M., Kim, Y., Kruschwitz, U., Song, D., Albakour, D., Dignum, S., Beresi, U. C., Fasli, M. e De Roeck, A. (2012). Automatically structuring domain knowledge from text: An overview of current research. *Information Processing & Management*, 48(3):552–568.

[Davis et al., 2016] Davis, C. A., Varol, O., Ferrara, E., Flammini, A. e Menczer, F. (2016). Botornot: A system to evaluate social bots. Em *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, p. 273–274, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.

[de Oliveira et al., 2020] de Oliveira, N. R., de Medeiros, D. S. V. e Mattos, D. M. F. (2020). Syntactic-relationship approach to construct well-informative knowledge graphs representation. Em *4th Cloud and Internet of Things - CIoT'20 (a ser apresentado)*. IEEE.

[de Oliveira et al., 2020] de Oliveira, N. R., Medeiros, D. S. V. e Mattos, D. M. F. (2020). A sensitive stylistic approach to identify fake news on social networking. *IEEE Signal Processing Letters*, 27:1250–1254.

[de Oliveira et al., 2020] de Oliveira, N. R., Reis, L. H., Fernandes, N. C., Bastos, C. A. M., de Medeiros, D. S. V. e Mattos, D. M. F. (2020). Natural language processing characterization of recurring calls in public security services. Em *Proceedings of the 2020 International Conference on Computing, Networking and Communications (ICNC)*, p. 1009–1013. IEEE.

[Deepak e Chitturi, 2020] Deepak, S. e Chitturi, B. (2020). Deep neural approach to fake-news identification. *Procedia Computer Science*, 167:2236 – 2243. International Conference on Computational Intelligence and Data Science.

[Deerwester et al., 1990] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K. e Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407.

[Domingos, 2012] Domingos, P. M. (2012). A few useful things to know about machine learning. *ACM Commun.*, 55(10):78–87.

[Fahad et al., 2014] Fahad, A., Alshatri, N., Tari, Z., Alamri, A., Khalil, I., Zomaya, A. Y., Foufou, S. e Bouras, A. (2014). A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *IEEE transactions on emerging topics in computing*, 2(3):267–279.

[Ferreira e Vlachos, 2016] Ferreira, W. e Vlachos, A. (2016). Emergent: a novel data-set for stance classification. Em *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: Human language technologies*, p. 1163–1168.

[Firth, 1957] Firth, J. R. (1957). A synopsis of linguistic theory, 1930-1955. *Studies in Linguistic Analysis*.

[Friggeri et al., 2014] Friggeri, A., Adamic, L., Eckles, D. e Cheng, J. (2014). Rumor cascades.

[Fuller et al., 2009] Fuller, C. M., Biros, D. P. e Wilson, R. L. (2009). Decision support for determining veracity via linguistic-based cues. *Decision Support Systems*, 46(3):695 – 703. Wireless in the Healthcare.

[Gabielkov et al., 2016] Gabielkov, M., Ramachandran, A., Chaintreau, A. e Legout, A. (2016). Social Clicks: What and Who Gets Read on Twitter? Em *ACM SIGMETRICS / IFIP Performance 2016*, Antibes Juan-les-Pins, France.

[Gan e Tao, 2015] Gan, J. e Tao, Y. (2015). Dbscan revisited: Mis-claim, un-fixability, and approximation. Em *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, p. 519–530.

[Gaonkar et al., 2019] Gaonkar, S., Itagi, S., Chalippatt, R., Gaonkar, A., Aswale, S. e Shetgaonkar, P. (2019). Detection of online fake news : A survey. Em *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)*, p. 1–6.

[Golbeck et al., 2018] Golbeck, J., Mauriello, M., Auxier, B., Bhanushali, K. H., Bonk, C., Bouzaghrane, M. A., Buntain, C., Chanduka, R., Cheakalos, P., Everett, J. B., Falak, W., Gieringer, C., Graney, J., Hoffman, K. M., Huth, L., Ma, Z., Jha, M., Khan, M., Kori, V., Lewis, E., Mirano, G., Mohn IV, W. T., Mussenden, S., Nelson, T. M., Mcwillie, S., Pant, A., Shetye, P., Shrestha, R., Steinheimer, A., Subramanian, A. e Visnansky, G. (2018). Fake news vs satire: A dataset and analysis. WebSci '18, p. 17–21, New York, NY, USA. Association for Computing Machinery.

[Govender e Sivakumar, 2020] Govender, P. e Sivakumar, V. (2020). Application of k-means and hierarchical clustering techniques for analysis of air pollution: A review (1980–2019). *Atmospheric Pollution Research*, 11(1):40–56.

[Grinberg et al., 2019] Grinberg, N., Joseph, K., Friedland, L., Swire-Thompson, B. e Lazer, D. (2019). Fake news on twitter during the 2016 us presidential election. *Science*, 363(6425):374–378.

[Hauch et al., 2015] Hauch, V., Blandón-Gitlin, I., Masip, J. e Sporer, S. L. (2015). Are computers effective lie detectors? a meta-analysis of linguistic cues to deception. *Personality and social psychology Review*, 19(4):307–342.

[Hu et al., 2016] Hu, B., Tang, B., Chen, Q. e Kang, L. (2016). A novel word embedding learning model using the dissociation between nouns and verbs. *Neurocomputing*, 171:1108–1117.

[Indurkhya e Damerau, 2010] Indurkhya, N. e Damerau, F. J. (2010). *Handbook of Natural Language Processing*. Chapman & Hall/CRC, 2nd edição.

[Jarmasz e Szpakowicz, 2003] Jarmasz, M. e Szpakowicz, S. (2003). Not as easy as it seems: Automating the construction of lexical chains using roget's thesaurus. Em *Advances in Artificial Intelligence*, p. 544–549, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Kadhim, 2019] Kadhim, A. I. (2019). Survey on supervised machine learning techniques for automatic text classification. *Artificial Intelligence Review*, 52(1):273–292.

[Ketchen e Shook, 1996] Ketchen, D. J. e Shook, C. L. (1996). The application of cluster analysis in strategic management research: an analysis and critique. *Strategic management journal*, 17(6):441–458.

[Kwon et al., 2013] Kwon, S., Cha, M., Jung, K., Chen, W. e Wang, Y. (2013). Prominent features of rumor propagation in online social media. Em *2013 IEEE 13th International Conference on Data Mining*, p. 1103–1108. IEEE.

[Lazer et al., 2018] Lazer, D. M., Baum, M. A., Benkler, Y., Berinsky, A. J., Greenhill, K. M., Menczer, F., Metzger, M. J., Nyhan, B., Pennycook, G., Rothschild, D. et al. (2018). The science of fake news. *Science*, 359(6380):1094–1096.

[Li et al., 2015] Li, Y., Xu, L., Tian, F., Jiang, L., Zhong, X. e Chen, E. (2015). Word embedding revisited: A new representation learning and explicit matrix factorization perspective. Em *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[Liu et al., 2010] Liu, G., Wang, Y. e Orgun, M. A. (2010). Quality of trust for social trust path selection in complex social networks. Em *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, p. 1575–1576. International Foundation for Autonomous Agents and Multiagent Systems.

[Manning et al., 2010] Manning, C., Raghavan, P. e Schütze, H. (2010). Introduction to information retrieval. *Natural Language Engineering*, 16(1):100–103.

[Manning e Schutze, 1999] Manning, C. e Schutze, H. (1999). *Foundations of statistical natural language processing*. MIT press.

[Manning et al., 2014] Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. e McClosky, D. (2014). The stanford corenlp natural language processing toolkit. Em *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, p. 55–60.

[Mattos et al., 2019] Mattos, D. M. F., Velloso, P. B. e Duarte, O. C. M. B. (2019). An agile and effective network function virtualization infrastructure for the internet of things. *Journal of Internet Services and Applications*, 10(1):6.

[Meel e Vishwakarma, 2020] Meel, P. e Vishwakarma, D. K. (2020). Fake news, rumor, information pollution in social media and web: A contemporary survey of state-of-the-arts, challenges and opportunities. *Expert Systems with Applications*, 153:112986.

[Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G. e Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[Mitra e Gilbert, 2015] Mitra, T. e Gilbert, E. (2015). Credbank: A large-scale social media corpus with associated credibility annotations. Em *ICWSM*, p. 258–267.

[Monteiro et al., 2018] Monteiro, R. A., Santos, R. L., Pardo, T. A., de Almeida, T. A., Ruiz, E. E. e Vale, O. A. (2018). Contributions to the study of fake news in portuguese: New corpus and automatic detection results. Em *International Conference on Computational Processing of the Portuguese Language*, p. 324–334. Springer.

[Navigli, 2009] Navigli, R. (2009). Word sense disambiguation: A survey. *ACM computing surveys (CSUR)*, 41(2):1–69.

[Oshikawa et al., 2018] Oshikawa, R., Qian, J. e Wang, W. Y. (2018). A survey on natural language processing for fake news detection. *arXiv preprint arXiv:1811.00770*.

[Otter et al., 2020] Otter, D. W., Medina, J. R. e Kalita, J. K. (2020). A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*.

[Papadimitriou et al., 2000] Papadimitriou, C. H., Raghavan, P., Tamaki, H. e Vempala, S. (2000). Latent semantic indexing: A probabilistic analysis. *Journal of Computer and System Sciences*, 61(2):217–235.

[Pennebaker et al., 2001] Pennebaker, J. W., Francis, M. E. e Booth, R. J. (2001). Linguistic inquiry and word count: Liwc 2001. *Mahway: Lawrence Erlbaum Associates*, 71(2001):2001.

[Pennycook et al., 2020] Pennycook, G., McPhetres, J., Zhang, Y., Lu, J. G. e Rand, D. G. (2020). Fighting covid-19 misinformation on social media: Experimental evidence for a scalable accuracy-nudge intervention. *Psychological science*, 31(7):770–780.

[Perdisci et al., 2006] Perdisci, R., Gu, G. e Lee, W. (2006). Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems. Em *Sixth International Conference on Data Mining (ICDM'06)*, p. 488–498.

[Poddar et al., 2019] Poddar, K., Umadevi, K. et al. (2019). Comparison of various machine learning models for accurate detection of fake news. Em *2019 Innovations in Power and Advanced Computing Technologies (i-PACT)*, volume 1, p. 1–5. IEEE.

[Potthast et al., 2017] Potthast, M., Kiesel, J., Reinartz, K., Bevendorff, J. e Stein, B. (2017). A stylometric inquiry into hyperpartisan and fake news. *arXiv preprint arXiv:1702.05638*.

[Qian et al., 2018] Qian, F., Gong, C., Sharma, K. e Liu, Y. (2018). Neural user response generator: Fake news detection with collective user intelligence. Em *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, p. 3834–3840. International Joint Conferences on Artificial Intelligence Organization.

[Rashkin et al., 2017] Rashkin, H., Choi, E., Jang, J. Y., Volkova, S. e Choi, Y. (2017). Truth of varying shades: Analyzing language in fake news and political fact-checking. Em *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, p. 2931–2937.

[Robnik-Šikonja e Kononenko, 2003] Robnik-Šikonja, M. e Kononenko, I. (2003). Theoretical and Empirical Analysis of ReliefF and RReliefF. *Machine Learning*, 53(1/2):23–69.

[Rousseeuw e Kaufman, 1990] Rousseeuw, P. J. e Kaufman, L. (1990). Finding groups in data. *Hoboken: Wiley Online Library*.

[Rubin et al., 2016] Rubin, V., Conroy, N., Chen, Y. e Cornwell, S. (2016). Fake news or truth? using satirical cues to detect potentially misleading news. Em *Proceedings of the second workshop on computational approaches to deception detection*, p. 7–17.

[Rubin, 2010] Rubin, V. L. (2010). On deception and deception detection: Content analysis of computer-mediated stated beliefs. Em *Proceedings of the 73rd ASIS&T Annual Meeting on Navigating Streams in an Information Ecosystem-Volume 47*, p. 32. American Society for Information Science.

[Rubin, 2014] Rubin, V. L. (2014). Pragmatic and cultural considerations for deception detection in asian languages. Em *ACM Transactions on Asian Language Information Processing*.

[Rubin et al., 2015a] Rubin, V. L., Chen, Y. e Conroy, N. J. (2015a). Deception detection for news: three types of fakes. Em *Proceedings of the 78th ASIS&T Annual Meeting:*

*Information Science with Impact: Research in and for the Community*, p. 83. American Society for Information Science.

[Rubin et al., 2015b] Rubin, V. L., Conroy, N. J. e Chen, Y. (2015b). Towards news verification: Deception detection methods for news discourse. Em *Hawaii International Conference on System Sciences*.

[Santia e Williams, 2018] Santia, G. C. e Williams, J. R. (2018). Buzzface: A news veracity dataset with facebook user commentary and egos. Em *Twelfth International AAAI Conference on Web and Social Media*, p. 531–540.

[Schubert et al., 2017] Schubert, E., Sander, J., Ester, M., Kriegel, H. P. e Xu, X. (2017). Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–21.

[Sharma et al., 2019] Sharma, K., Qian, F., Jiang, H., Ruchansky, N., Zhang, M. e Liu, Y. (2019). Combating fake news: A survey on identification and mitigation techniques. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(3):1–42.

[Sharma e Sharma, 2019] Sharma, S. e Sharma, D. K. (2019). Fake news detection: A long way to go. Em *2019 4th International Conference on Information Systems and Computer Networks (ISCON)*, p. 816–821. IEEE.

[Shu et al., 2019] Shu, K., Mahudeswaran, D. e Liu, H. (2019). Fakenewstracker: a tool for fake news collection, detection, and visualization. *Computational and Mathematical Organization Theory*, 25(1):60–71.

[Shu et al., 2020] Shu, K., Mahudeswaran, D., Wang, S., Lee, D. e Liu, H. (2020). Fakenewsnet: A data repository with news content, social context, and spatiotemporal information for studying fake news on social media. *Big Data*, 8(3):171–188.

[Shu et al., 2017] Shu, K., Sliva, A., Wang, S., Tang, J. e Liu, H. (2017). Fake news detection on social media: A data mining perspective. *ACM SIGKDD Explorations Newsletter*, 19(1):22–36.

[Socher et al., 2013] Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y. e Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. Em *Proceedings of the 2013 conference on empirical methods in natural language processing*, p. 1631–1642.

[Thorne et al., 2018] Thorne, J., Vlachos, A., Christodoulopoulos, C. e Mittal, A. (2018). FEVER: a large-scale dataset for fact extraction and VERification. Em *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, p. 809–819. Association for Computational Linguistics.

[Van Bavel et al., 2020] Van Bavel, J. J., Baicker, K., Boggio, P. S., Capraro, V., Cichocka, A., Cikara, M., Crockett, M. J., Crum, A. J., Douglas, K. M., Druckman, J. N. et al. (2020). Using social and behavioural science to support covid-19 pandemic response. *Nature Human Behaviour*, p. 1–12.

[Verikas et al., 2011] Verikas, A., Gelzinis, A. e Bacauskiene, M. (2011). Mining data with random forests: A survey and results of new tests. *Pattern recognition*, 44(2):330–349.

[Vosoughi et al., 2018] Vosoughi, S., Roy, D. e Aral, S. (2018). The spread of true and false news online. *Science*, 359(6380):1146–1151.

[Wang, 2017] Wang, W. Y. (2017). "Liar, liar pants on fire": A new benchmark dataset for fake news detection. Em *Annual Meeting of the Association for Computational Linguistics - ACL 2017*.

[Xu e Wunsch, 2005] Xu, R. e Wunsch, D. (2005). Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678.

[Zhai et al., 2014] Zhai, Y., Ong, Y.-S. e Tsang, I. W. (2014). The emerging"big dimensionality". *IEEE Computational Intelligence Magazine*, 9(3):14–26.

[Zhou et al., 2004] Zhou, L., Burgoon, J. K., Nunamaker, J. F. e Twitchell, D. (2004). Automating linguistics-based cues for detecting deception in text-based asynchronous computer-mediated communications. *Group decision and negotiation*, 13(1):81–106.

[Zhou et al., 2015] Zhou, X., Cao, J., Jin, Z., Xie, F., Su, Y., Chu, D., Cao, X. e Zhang, J. (2015). Real-time news cer tification system on sina weibo. Em *Proceedings of the 24th International Conference on World Wide Web*, p. 983–988.

[Zhou e Zafarani, 2018] Zhou, X. e Zafarani, R. (2018). Fake news: A survey of research, detection methods, and opportunities. *arXiv preprint arXiv:1812.00315*.

[Zhou et al., 2019] Zhou, Z., Guan, H., Bhat, M. M. e Hsu, J. (2019). Fake news detection via nlp is vulnerable to adversarial attacks. *arXiv preprint arXiv:1901.09657*.

[Zubiaga et al., 2016] Zubiaga, A., Liakata, M., Procter, R., Wong Sak Hoi, G. e Tolmie, P. (2016). Analysing how people orient to and spread rumours in social media by looking at conversational threads. *PloS one*, 11(3):e0150989.

# Capítulo 3

# Privacidade do Usuário em Aprendizado Colaborativo: *Federated Learning*, da Teoria à Prática

Helio N. C. Neto (UFF)
Diogo M. F. Mattos (UFF)
Natalia C. Fernandes (UFF)

***Abstract***

*The growing number of Internet of Things devices represents an uninterrupted data source that waits for machine learning mechanisms to run. However, the presence of personal data and the difficulty in ensuring users' privacy make it difficult to extract knowledge under recent restrictions such as the General Law on the Protection of Personal Data. In this chapter, we present the Federated Learning paradigm, which allows collaborative execution of learning models in local data and subsequent aggregation into a centralized global model. The chapter focuses on presenting the principles, applications, as well as challenges and attacks on federated learning, through a practical-theoretical approach with a focus on users' privacy.*

***Resumo***

*O número crescente de dispositivos de Internet das Coisas representa uma fonte de dados ininterrupta a espera da execução de mecanismos de aprendizado de máquina. Contudo, a presença de dados pessoais e a dificuldade em garantir a privacidade dos usuários dificultam a extração de conhecimento perante restrições recentes como a Lei Geral de Proteção de Dados Pessoais. Assim, este capítulo apresenta o paradigma de Aprendizado Federado (Federated Learning), que permite a execução colaborativa de modelos de aprendizado em dados locais e posterior agregação em um modelo global centralizado. O capítulo foca em apresentar os princípios, as aplicações, assim como os desafios e os ataques ao aprendizado federado, através de uma abordagem prático-teórica com foco na privacidade dos usuários.*

Este capítulo foi realizado com recursos do CNPq, CAPES, RNP, FAPERJ e FAPESP (2018/23062-5).

## 3.1. Introdução

Previsões apontam que até o final do ano de 2023 o número de dispositivos em rede alcançará um total de 29,3 bilhões de dispositivos, correspondendo a mais de três vezes a total do planeta [2]. No entanto, o número de usuários da Internet deve atingir 5,3 bilhões, o que corresponde a 66% do número de habitantes do planeta. Dessa forma, esses dispositivos apresentam o potencial de implantar operações de sensoriamento coletivo (*crowdsensing*) e fornecer dados para a criação de modelos de aprendizado de máquina para diversos fins [Lim et al., 2020]. Paralelamente, há também um crescente desenvolvimento de técnicas de aprendizado profundo (*Deep Learning* — DL) [Medeiros et al., 2019] e técnicas baseadas em treinamento pelo gradiente descendente estocástico [Lobato et al., 2018, Reis et al., 2020]. Nesse sentido, a coleta de dados das mais diversas fontes permite a realização de diversas pesquisas e aplicações inovadoras [Medeiros et al., 2019]. Na abordagem tradicional [Li et al., 2017], os dados coletados por dispositivos móveis são carregados e processados de forma centralizada em servidores na nuvem, seja um servidor único ou um centro de dados. Contudo, a abordagem centralizada de processamento e fusão de dados implica questões de segurança e privacidade dos dados que são abordadas pelas leis de proteção de dados pessoais em diversos países no mundo.

Políticas de proteção de dados privados, cada vez mais severas, impõem limites para essa abordagem centralizada de extração de conhecimento dos dados. As leis de proteção de dados pessoais estipulam direitos aos titulares dos dados e obrigações às instituições que detêm tais dados. Uma lei de destaque é a GDPR, *General Data Protection Regulation*, vigente em toda a União Europeia (UE) e que estabelece diretrizes quanto ao tratamento, por uma pessoa, empresa ou organização, dos dados pessoais de pessoas na UE [Parlamento Europeu e Conselho da União Européia, 2016]. A legislação enfatiza a preocupação em defender os direitos e as liberdades fundamentais dos indivíduos em relação ao manuseio de seus dados e tem inspirado outros países a assumirem compromissos semelhantes, tais como a Lei Geral de Proteção de Dados Pessoais (LGPD) no Brasil[3], a *California Consumer Privacy Act* (CCPA) nos Estados Unidos[4], a *The Personal Information Protection and Electronic Documents Act* (PIPEDA) no Canadá[5], entre outras. No Brasil, a Lei Geral de Proteção de Dados Pessoais identifica como agentes de tratamento a pessoa natural ou jurídica de direito público ou privado que realiza qualquer operação de tratamento sobre os dados pessoais de outrem [Brasil, 2018]. Dentre os deveres estabelecidos a esses agentes estão a coleta de consentimento explícito do titular do dado e a disponibilização de relatórios que identifiquem as operações de tratamento aplicadas ao dado, incluindo a especificação de seu local de armazenamento, mascaramento do dado e medidas de proteção ao dado.

Outra questão importante relaciona-se ao custo do envio dos dados à nuvem, já que os dispositivos móveis podem estar em uma área com baixa taxa de transferência e altos atrasos. A abordagem centrada na nuvem implica atrasos de propagação que podem incorrer em latências inaceitáveis para determinadas aplicações de decisão em tempo

[2]Disponível em https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[3]Disponível em https://www.lgpdbrasil.com.br/. Último acesso em 25/09/2020.

[4]Disponível em https://www.dlapiperdataprotection.com/?t=law&c=US. Último acesso em 17/09/2020.

[5]Disponível em https://www.dlapiperdataprotection.com/?t=law&c=CA. Último acesso em 17/09/2020.

real [Gupta e Jha, 2015]. A transferência de dados para a nuvem para processamento sobrecarrega tanto as redes de núcleo quanto as redes de acesso. A sobrecarga é ainda mais relevante quando são considerados dados não estruturados, como textos, voz ou vídeo [de Oliveira et al., 2020a, de Oliveira et al., 2020b]. As restrições de envio de dados para nuvem, tanto em banda quanto em atraso, é mais crítica quando o treinamento centrado na nuvem é dependente de redes de acesso sem fio [Mattos et al., 2019, Medeiros et al., 2019]. Assim, propostas atuais consideram o desenvolvimento de aplicações móveis na borda da nuvem (*Mobile Edge Computing* — MEC) [Wang et al., 2019a, Nishio e Yonetani, 2019], em que o aprendizado passa a ser uma tarefa realizada por três atores distintos, dispositivos, borda e nuvem. O aprendizado ancorado no modelo MEC incorre em custos de comunicação significativos e é pouco adequado para aplicações com retreinamento constante [Lim et al., 2020]. Além disso, a terceirização da computação e do processamento de dados em servidores de borda ainda envolvem a transmissão de dados pessoais potencialmente confidenciais, o que pode expor dados sensíveis à privacidade ou mesmo violar leis de proteção de dados pessoais [Parlamento Europeu e Conselho da União Européia, 2016, Brasil, 2018].

O aprendizado federado (*Federated Learning* — FL) é proposto como solução de aprendizado colaborativo com foco na preservação da privacidade e eficiência de comunicação [Brendan McMahan et al., 2017]. O aprendizado federado permite o treinamento com dados reais a partir de dispositivos móveis e a utilização de dados sensíveis à privacidade, sem expor o dono dos dados. O aprendizado federado é uma das principais abordagens para garantir a privacidade dos dados ao passo em que permite a execução de algoritmos de aprendizado de máquina de forma colaborativa sem que os dados privados sejam transferidos para um servidor em nuvem.

O aprendizado federado consiste em um conjunto de tarefas realizadas pelos clientes e servidor do aprendizado federado. Os clientes devem realizar as seguintes tarefas: i) todos os clientes devem recuperar os parâmetros de um modelo treinado do servidor, ii) clientes selecionados aleatoriamente devem atualizar o modelo com seus próprios dados e, após, iii) transferir para o servidor os novos parâmetros do modelo treinado com os dados locais. Ao servidor cabe a tarefa de agregar atualizações providas por diversos clientes, objetivando a melhoraria do modelo global. Após a agregação das contribuições locais, o servidor repassa o novo modelo a todos os clientes, permitindo que todos apliquem o modelo atualizado. As abordagens federadas introduzem a solução de compromisso entre o uso de recursos computacionais (CPU ou GPU) locais para o treinamento dos modelos, ao passo que os clientes podem manter seus dados locais, seguros e privados. Nesse sentido, o aprendizado federado se apresenta como uma abordagem poderosa para manter a privacidade, já que os dados são sempre processados localmente e as operações globais, centralizadas, são voltadas para a agregação de modelos distintos, sem acesso aos dados individuais.

Abordagens ingênuas de aprendizado colaborativo se baseiam nas suposições de que os dados locais dos participantes são independentes e identicamente distribuídos, consistindo um conjunto de dados *iid*. Contudo, a heterogeneidade de dispositivos e a heterogeneidade estatística dos dados contradizem essas suposições, gerando conjuntos de dados que não são independentes e podem ser ter distribuições estatísticas diversas entre os participantes do aprendizado federado, sendo chamados de dados *non-iid*. As

características em um conjunto de dados non-iid são dependentes e não é distribuída de do forma idêntica, ou seja, cada característica não tem a mesma probabilidade de distribuição. Assim, As principais propostas de aprendizado federado se baseiam em modelos de aprendizado profundo e propostas baseadas no treinamento com o algoritmo de gradiente descendente estocástico [Brendan McMahan et al., 2017, Li et al., 2020]. Essas abordagens são prevalentes em detrimento a outros modelos de aprendizado de máquina, pois os principais algoritmos de agregação de modelos federados estão fortemente atrelados as médias ponderadas dos modelos treinados e, portanto, métodos de treinamento que fornecem a direção de otimização do modelo tendem a ser preferidos. Assim, um dos principais algoritmos de aprendizado federado é a média federada (*Federated Average* – FedAvg), que pondera a direção de otimização dos modelos dos clientes de acordo com a quantidade de dados que cada cliente usa para treinar seus modelos.

Embora o aprendizado federado seja proposto como uma solução de privacidade para a realização do aprendizado sobre dados privados, diversos ataques focam no próprio funcionamento do aprendizado para extrair informações dos clientes ou para perverterem o modelo gerado. O primeiro tipo de ataque é o envenenamento do conjunto de dados, em que existem um ou mais participantes maliciosos que tentam enviesar o treinamento utilizando amostras com rótulos falsos. Assim, o atacante pode conduzir o modelo global a falsas inferências. O segundo tipo de ataque é o envenenamento do modelo [Bhagoji et al., 2019], em que o atacante modifica diretamente os parâmetros do modelo local. Esse tipo de ataque é mais efetivo que o anterior, principalmente em ambientes federados com muitos participantes, pois esse ataque não depende do treinamento de um conjunto de dados malicioso. O terceiro é o ataque *Free-Riding* [Weng et al., 2019], no qual um participante visa se beneficiar do modelo global, porém sem contribuir com o processo de treinamento. Nesse caso, o atacante simula ter um conjunto de dados pequeno, realizando pouco ou nenhum esforço computacional e recebe o modelo global treinado colaborativamente pelos outros participantes. O quarto ataque é a inversão do modelo, no qual o atacante possui os modelos de aprendizado de máquina dos usuários e consegue aprender informações sobre os dados. Por fim, há ainda o ataque de inferência dos gradientes, em que o atacante consegue inferir os dados dos usuários através de seus gradientes.

Este capítulo foca em apresentar os princípios, as aplicações e os desafios do aprendizado federado através de uma abordagem prática-teórica com foco na privacidade do usuário. O capítulo aborda os principais conceitos sobre aprendizado federado e os principais tipos de ataques ao ambiente do aprendizado federado, assim como as principais aplicações existentes atualmente. Os principais desafios de pesquisa sobre o aprendizado federado são discutidos em detalhes. Ao final do capítulo, há um roteiro de desenvolvimento de uma aplicação prática visando uma simulação de um ambiente federado em Python. O roteiro da aplicação utiliza a biblioteca TensorFlow[6] e permite que o desenvolvedor crie seus próprios algoritmos de aprendizado federado utilizando os modelos de aprendizado de máquina providos pelo arcabouço Keras[7].

O restante do capítulo está organizado da seguinte forma. A Seção 3.2 apresenta os fundamentos de aprendizado de máquina colaborativo. A Seção 3.3 discute e formaliza

[6]Disponível em https://www.tensorflow.org.

[7]Disponível em https://keras.io/.

o aprendizado federado. A Seção 3.4 descreve as principais aplicações do aprendizado federado. Os ataques ao paradigma de aprendizado federado são elencados na Seção 3.5 e os principais desafios de pesquisa são analisados na Seção 3.6. O roteiro para desenvolvimento de uma aplicação prática é apontado na Seção 3.7. Por fim, a Seção 3.8 conclui o capítulo.

## 3.2. Fundamentos de Aprendizado de Máquina e Aprendizado Colaborativo

O aprendizado de máquina permite inferir soluções para problemas que possam ser representados por um amplo conjunto de dados. Assim, de forma simplificada, é possível dividir os problemas de aprendizado de máquina em quatro categorias: agrupamentos (*clustering*), classificação, regressão e extração de regras. *Problemas de agrupamento* têm como objetivo aglomerar as amostras em grupos de acordo com a sua similaridade [Medeiros et al., 2019]. Assim, esses algoritmos de aprendizado buscam minimizar a distância entre amostras de dados com características similares em um mesmo grupo, enquanto maximizam a separação entre grupos distintos. *Problemas de classificação* caracterizam-se por, dado um conjunto de possibilidades de saídas, tentar classificar cada entrada do conjunto de dados em uma saída possível. Assim, esse tipo de aprendizagem mapeia as entradas em classes-alvos de saída, que são representadas pelo conjunto discreto de valores de saída. Os *problemas de regressão*, assim como os problemas de classificação, mapeiam uma entrada em um valor de saída. Contudo, ao invés de classificar uma amostra em uma classe discreta (0 ou 1, "cachorro"ou "gato", etc.), mapeiam essa entrada em um valor em um intervalo contínuo. Assim, na regressão, os algoritmos geram modelos matemáticos que tendem a se comportar como funções multivariáveis em que os valores de entrada são mapeados em valores contínuos de saída. Já os *problemas de extração* de regras são diferentes dos demais, pois o objetivo desse problema não é inferir valores de saída para cada conjunto de entrada, mas identificar relações estatísticas entre os dados.

Existem quatro principais paradigmas de aprendizado de máquina, que são: i) aprendizado supervisionado, ii) aprendizado não-supervisionado, iii) aprendizado semi-supervisionado e iv) aprendizado por reforço (*reinforcement learning*) [Medeiros et al., 2019]. O aprendizado supervisionado necessita de um conjunto de dados rotulados, chamado de conjunto de treinamento, para criar o modelo de classificação ou regressão dos dados. Esse paradigma de aprendizagem requer a existência *a priori* de um conjunto de dados rotulados para a criação e treinamento do modelo de aprendizado de máquina. Já o *aprendizado não-supervisionado* busca padrões nos dados de treinamento e, portanto, não requer a existência prévia de dados rotulados. O aprendizado não-supervisionado é adequado para problemas de agrupamentos (*clustering*), nos quais se quer agrupar as entradas em grupos que não foram definidos (rotulados) *a priori*. O aprendizado não-supervisionado também pode ser utilizado na redução de dimensão de um conjunto de dados, em que encontra-se direções de maximização de variância em que os dados se projetam. Por sua vez, o paradigma de *aprendizado semi-supervisionado* suporta o uso de conjuntos de treinamento com rótulos faltantes ou incompletos. São utilizadas técnicas de aprendizado não-supervisionado nos dados não rotulados na tentativa de normalizar os dados. Ao final da etapa não-supervisionada, o modelo associa cada observação com uma pontuação proporcional à probabilidade do dado ter vindo de um determinado grupo de amostras rotuladas. Em seguida, pode-se usar a parte rotulada dos dados para definir um

limiar na pontuação, a partir do qual é considerado, qual rótulo é mais apropriado para a amostra. Por fim, o paradigma de *aprendizado por reforço* consiste em um processo iterativo, em que agentes aprendem efetivamente novos conhecimentos na ausência de um conjunto de treinamento, com pouco ou nenhum conhecimento do ambiente [Tesauro, 2007]. A ideia central do aprendizado por reforço é que o aprendizado de um agente é baseado em exemplos do conjunto de treinamento, nos quais os agentes interagem com o mundo externo e aprendem com reforços providos pelo ambiente. Os reforços providos podem ser recompensas ou penalidades. Assim, o conjunto de treinamento consiste de pares de amostras de dados e reforços, sejam recompensas ou penalidades. A retroalimentação do ambiente impulsiona o agente para a melhor sequência de ações. O aprendizado por reforço é adequado para problemas de tomada de decisão e planejamento [Tesauro, 2007].

Os algoritmos convencionais de aprendizado de máquina, normalmente, dependem da realização de engenharia de características para processar os conjuntos de dados brutos [Trigeorgis et al., 2016]. A engenharia de características é o processo que transforma dados brutos, ou seja, dados coletados diretamente de uma determinada fonte, em características que melhor representam o problema, para resultar no melhor desempenho do modelo de aprendizado de máquina. O tratamento dessas novas características é feito manualmente por especialistas na construção de um dataset. Dessa forma, a especialização no domínio costuma ser um pré-requisito para a construção de um modelo de aprendizado de máquina eficaz. Paralelamente, as redes neurais profundas são baseadas no aprendizado por representação, ou seja, podem descobrir e aprender automaticamente essas características a partir dos dados brutos [Lecun et al., 1998]. Com isso, muitas vezes, esse tipo de abordagem supera os algoritmos convencionais de aprendizado de máquina, especialmente quando há dados em abundância.

Os principais aspectos das técnicas que se caracterizam por serem de aprendizado profundo (*deep learning*) são os modelos que consistem em rede neurais com múltiplas camadas ou estágios de processamento não-lineares e os métodos para aprendizagem supervisionada ou não supervisionada de representação de características em camadas sucessivamente mais altas e mais abstratas.Dessa forma, a ideia chave do aprendizado profundo é gerar novas representações dos dados a cada camada, aumentando o grau de abstração da representação. A popularidade crescente das técnicas de aprendizado profundo deve-se ao aumento acelerado da capacidade de processamento, e.g. processamento gráfico (Graphical Processing Unit — GPUs); à grande produção de dados; e aos avanços nas pesquisas de aprendizado de máquina [Kwon et al., 2017]. Esses avanços permitiram a exploração de funções complexas não-lineares, a aprendizagem distribuída e uso efetivo de dados rotulados e não rotulados. Os principais algoritmos de aprendizado profundo são as Redes Neurais Convolucionais [Lecun et al., 1998], Redes Neurais Recorrentes [Mikolov et al., 2011] e *AutoEncoders* [Baldi, 2011].

### 3.2.1. Aprendizado de Máquina Distribuído

O aprendizado de máquina distribuído é um sistema que busca combinar o poder computacional em um aglomerado computacional independente de seu tamanho (gastando mais tempo fazendo cálculos úteis e menos tempo esperando por comunicação). O aprendizado de máquina distribuído, possui no mínimo duas entidades, os nós trabalhado-

res e o servidor de parâmetros [Ho et al., 2013]. O servidor de parâmetros é responsável por gerenciar as tarefas entre os nós trabalhadores. Por sua vez, os nós trabalhadores são responsáveis por executar as tarefas demandadas pelo servidor de parâmetro [Ho et al., 2013]. O servidor de parâmetros, ou nó central, é a ferramenta fundamental para acelerar o processo de treinamento, armazenando dados em nós de trabalho distribuídos e alocando dados e recursos computacionais por meio de interface para treinar o modelo de forma eficiente. No aprendizado de máquina distribuído, o nó central sempre assume o controle e possui total acesso ao conjunto de treinamento; assim, o aprendizado federado se depara com um ambiente de aprendizado mais complexo, pois enfatiza a proteção da privacidade dos dados dos proprietários durante o processo de treinamento. A Figura 3.1 ilustra a arquitetura básica do aprendizado de máquina distribuído.

O aprendizado federado à primeira vista é semelhante ao aprendizado de máquina distribuído. Contudo, o aprendizado de máquina distribuído cobre muitos aspectos não cobertos pelo aprendizado federado, devido à restrição de preservação de privacidade, como, por exemplo, armazenamento distribuído dos dados de treinamento e operação de tarefas computacionais distribuídas. Na maioria dos casos, em aprendizado de máquina distribuído, o servidor de parâmetros possui total controle dos nós trabalhadores e do conjunto de dados. O servidor de parâmetros, em aprendizado federado, é o servidor agregador, que não possui controle dos participantes. Sua função é selecionar os clientes participantes e agregar os parâmetros locais recebidos pelos clientes selecionados. O participante pode se recusar a participar ou até mesmo perder a conexão durante o treinamento. O nó trabalhador, em aprendizado federado, é representado pelo proprietário dos dados.

Em aprendizado de máquina distribuído, o servidor de parâmetros e os nós trabalhadores estão em um mesmo *datacenter* [Lim et al., 2020]. Logo, comunicação não é um problema desse paradigma, pois os enlaces entre o servidor e os nós trabalhadores possuem altas taxas de transferência. Já o aprendizado federado, os participantes, podem estar em áreas em que a taxa de transferência é desfavorável ao envio dos parâmetros para treinamento. Então, técnicas para reduzir o custo de comunicação são frequentemente abordadas nesse paradigma.

### 3.2.2. Computação na Borda e Banco de Dados Federado

Atualmente, as fontes de dados estão localizadas em dispositivos fora da nuvem. Nesse contexto, a computação móvel na ponta (*Mobile Edge Computing* — MEC) é proposta como uma solução em que os servidores em nuvem são aproximados dos dispositivos finais. Esses servidores ficam localizados nos limites das redes das operadoras, simplificando o acesso aos dados gerados pelos dispositivos móveis. Nesse sentido, a MEC permite que o treinamento do modelo de aprendizado seja trazido para perto de onde os dados são produzidos [Lim et al., 2020], ou seja, nos dispositivos na rede de acesso. A Figura 3.2, apresenta a arquitetura tradicional de uma MEC.

Para o treinamento do modelo de aprendizado de máquina em abordagens MEC convencionais, um paradigma colaborativo foi amplamente utilizado. Nesse paradigma os dados de treinamento são enviados primeiro aos servidores de borda para treinamento do modelo, ao invés de diretamente para os servidores na nuvem para tentar diminuir

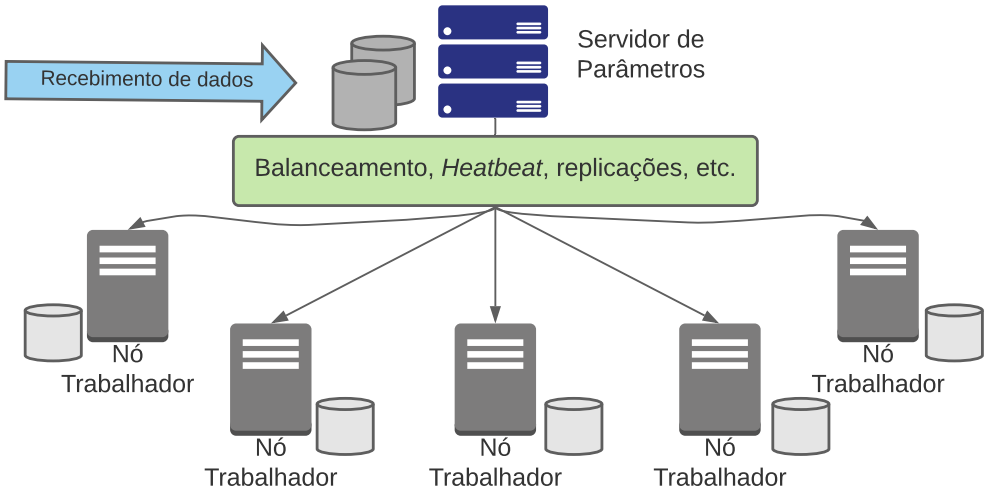



**Figura 3.1. Arquitetura do aprendizado de máquina distribuído. Nesse cenário, o servidor de parâmetros recebe os dados para processamento e distribui os dados e tarefas para execução nos nós trabalhadores. O servidor de parâmetros também verifica o estado dos nós trabalhadores enviando *heartbeat*.**

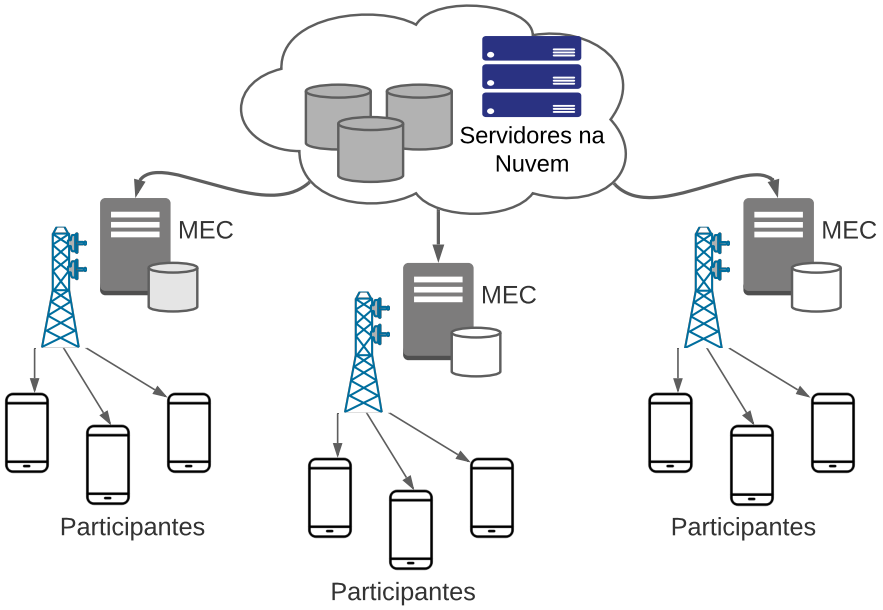



**Figura 3.2. Arquitetura da computação móvel na ponta (MEC), onde os servidores estão localizados na estrutura das operadoras, oferecendo, assim, alta taxa de transferência e baixa latência.**

o custo de comunicação. No entanto, esse paradigma ainda incorre em altos custos de comunicação e é inadequado especialmente para aplicações que requerem treinamento constante [Wang et al., 2020]. Além disso, o processamento dos dados em servidores de borda ainda implica transmissão de dados pessoais potencialmente sensíveis de celulares para servidores na borda da rede. Isso desencoraja clientes preocupados com a privacidade de seus dados de participar do treinamento do modelo, ou ainda, dependendo de como os dados são armazenados e usados, pode até mesmo violar leis de privacidade cada vez mais rigorosas como, por exemplo, a Lei Geral de Proteção de Dados Pessoais

(LGPD). Para garantir que os dados de treinamento permaneçam nos dispositivos pessoais dos participantes e para facilitar o aprendizado de máquina colaborativo de modelos complexos entre dispositivos distribuídos, aplicações MEC estão cada vez mais adotando o aprendizado federado [Wang et al., 2020].

Outro conceito relacionado ao de aprendizado federado é o de sistemas de banco de dados federados [Sheth e Larson, 1990]. Bancos de dados federados são sistemas que integram várias unidades de banco de dados e gerenciam o sistema integrado como um todo. O conceito de banco de dados federado é proposto para alcançar a interoperabilidade entre vários bancos de dados independentes. Um banco de dados federado usa armazenamento distribuído para as unidades de banco de dados e, na prática, os dados em cada unidade de banco de dados são heterogêneos. Portanto, tem muitas semelhanças com o aprendizado federado em termos de tipo e armazenamento de dados. No entanto, o sistema de banco de dados federado não envolve nenhum mecanismo de proteção de privacidade no processo de interação entre os sistemas e todas as unidades de banco de dados são completamente visíveis para o sistema de gerenciamento. Além disso, o foco do sistema de banco de dados federado está nas operações básicas de dados, incluindo inserção, exclusão, pesquisa e fusão, enquanto o objetivo do aprendizado federado é estabelecer um modelo conjunto para cada proprietário de dados sob a premissa de proteção da privacidade dos dados [Yang et al., 2019].

### 3.2.3. Aprendizado multipartidário seguro com preservação de privacidade

O aprendizado federado pode ser considerado um aprendizado de máquina colaborativo descentralizado e com preservação de privacidade [Yang et al., 2019]. Portanto, o aprendizado federado está estreitamente relacionado ao aprendizado de máquina multipartidário seguro (*Secure Multiparty Computacion* — SMC). O SMC é um protocolo de criptografia que distribui uma computação entre várias partes, em que nenhuma dessas partes pode ter acesso aos dados de outras partes. Os protocolos de computação multipartidários seguros permitem que sejam computados dados distribuídos de forma compatível, segura e privada [Bogetoft et al., 2009]. Muitos esforços de pesquisa foram dedicados ao aprendizado de máquina com preservação de privacidade em trabalhos acadêmicos. Por exemplo, trabalhos anteriores [Vaidya et al., 2008, Du e Zhan, 2002] propuseram algoritmos para árvores de decisão multipartidárias seguras para dados particionados verticalmente. Vaidya e Clifton propuseram regras de mineração de associação segura [Vaidya e Clifton, 2002], k-médias seguros [Vaidya e Clifton, 2003] e um classificador Bayesiano ingênuo [Vaidya e Clifton, 2004] para dados particionados verticalmente. Algoritmos de máquinas de vetor suporte seguro foram desenvolvidos para dados particionados verticalmente [Yu et al., 2006b] e dados particionados horizontalmente [Yu et al., 2006a]. Du *et al.* propuseram protocolos seguros para regressão linear e classificação multipartidária [Du et al., 2004]. Wan *et al.* propuseram métodos de gradiente descendente multipartidários seguros [Wan et al., 2007]. Todos esses trabalhos utilizaram SMC [Yao, 1982, Micali et al., 1987] para garantias de privacidade.

## 3.3. Estratégias para o Aprendizado Federado

O aprendizado federado envolve o treinamento de um modelo estatístico global utilizando dados de dispositivos remotos. A principal motivação do aprendizado federado

é a privacidade dos dados dos participantes durante o treinamento colaborativo. Durante o processo de treinamento, os participantes treinam um modelo compartilhado colaborativamente mantendo os dados no dispositivo. O objetivo do aprendizado federado é treinar o modelo global processando os dados localmente nos dispositivos. Assim, os dispositivos enviam atualizações intermediárias a um servidor central durante cada iteração de comunicação. O servidor agrega os modelos intermediários e distribui o novo modelo agregado a todos os participantes.

### 3.3.1. Formulação Matemática

O aprendizado federado é uma tecnologia que viabiliza o treinamento de modelos de aprendizado de máquina em redes de dispositivos móveis mantendo a privacidade dos usuários [Lim et al., 2020]. Existem duas entidades principais no sistema de aprendizado federado, os participantes, que são os donos dos dados, e o servidor agregador, que é responsável por criar o modelo global [Li et al., 2020]. Denota-se $N = \{1, ..., n\}$ como o conjunto de participantes, onde cada participante possui seu próprio conjunto de dados privado $D_n, n \in N$. Cada participante $n$ utiliza seu conjunto de dados $D_n$ para treinar um modelo local $w_n^t$ e realiza o envio apenas dos parâmetros do modelo local para o servidor do aprendizado federado a cada certo período de tempo. Em cada iteração de comunicação, um subconjunto $S^t$, $S^t \in N$ dos dispositivos é selecionado aleatoriamente para fornecer os parâmetros de seus modelos locais para o servidor. Em seguida, todos os parâmetros dos modelos locais selecionados são agregados para gerar um modelo global denominado $w_G^t$. Os modelos locais são atualizados localmente por $\tau$ atualizações locais, antes do envio dos parâmetros dos modelos locais para o servidor realizar a agregação global.

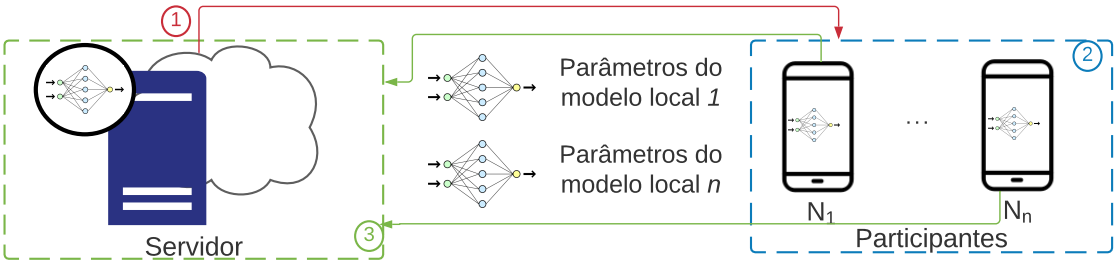



**Figura 3.3. Arquitetura do aprendizado federado. No primeiro passo, o servidor envia o modelo inicial. Posteriormente, cada participante selecionado atualiza o modelo com seus dados locais e, por fim, o servidor agrega os parâmetros recebidos.**

A Figura 3.3 ilustra o processo de treinamento que é adotado no aprendizado federado. O processo de treinamento difere de acordo com a necessidade de cada cenário, porém o conceito base se mantém. Uma suposição subjacente é que os participantes são honestos, o que significa que eles usam seus dados privados reais para fazer o treinamento e enviam os parâmetros verdadeiros para o servidor. O aprendizado federado possui três passos base que são enumerados a seguir.

1. Inicialização. O servidor inicia a tarefa de treinamento e especifica os hiperparâmetros do modelo global como, por exemplo, taxa de aprendizagem, quan-

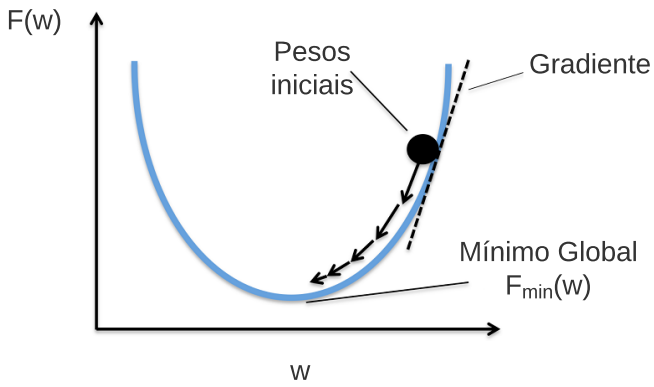
tidade de atualizações locais, tamanho do mini-lote etc. Em seguida, o servidor transmite o modelo global inicializado $w_G^0$ aos participantes selecionados.

2. Atualizações Locais. Com base no modelo global $w_G^t$ recebido pelo servidor, em que $t$ denota o índice de iteração global atual, os participantes selecionados, respectivamente, utilizam seus dados e dispositivos locais para atualizar os parâmetros do modelo local $w_n^t$. O objetivo do participante $n$ na iteração $t$ é encontrar os parâmetros otimizados $w_n^t$ que minimizam a função de perda $L(w_n^t)$.

3. Agregação Global. O servidor agrega os parâmetros dos modelos locais dos participantes selecionados e, em seguida, envia o modelo global atualizado $w_G^{t+1}$ de volta a todos os participantes. O objetivo do servidor é minimizar a função de perda global $L(w_G^t)$.

Os passos de atualização dos modelos locais e agregação do modelo global são repetidos até a convergência da função de perda global ou que outra condição de parada seja atendida. O aprendizado federado funciona com modelos que utilizam métodos de gradiente descendente estocástico, (*Stochastic Gradient Descent* — SGD) [Brendan McMahan et al., 2017], como redes neurais, regressão linear e máquinas de vetor suporte. Gradiente, em termos simples, significa inclinação de uma superfície. Portanto, o gradiente descendente é a direção que indica o declive para chegar ao ponto mínimo dessa superfície. A Figura 3.4(a) ilustra esse processo. Então, o principal objetivo do algoritmo gradiente descendente é encontrar os melhores parâmetros, os pesos no caso das redes neurais, para minimizar uma função. Para isso, é calculada a derivada da função para cada amostra no conjunto de dados. Todavia, esse processo causa sobrecarga, caso o conjunto de dados possua grande número de amostras. O SGD é uma variante do gradiente descendente que foi proposta para minimizar a sobrecarga. Um comportamento estocástico implica um sistema ou processo que está vinculado a uma probabilidade aleatória. Portanto, no gradiente descendente estocástico, algumas amostras são selecionadas aleatoriamente em vez de todo o conjunto de dados a cada iteração. O lote denota o número total de amostras de um conjunto de dados que é usado para calcular o gradiente em cada iteração [Bottou, 2010]. A Figura 3.4(b) apresenta a diferença entre gradiente descendente e SGD.

Os modelos de aprendizado de máquina possuem um conjunto de parâmetros que são atualizados com base nos dados de treinamento. Uma amostra $j$ dos dados de treinamento possui duas partes. A primeira parte é o vetor $x_j$ que são as características da amostra $j$, que são passadas como entrada ao modelo de aprendizado de máquina; a outra parte é um escalar $y_j$ que é a saída desejada pelo modelo. Para otimizar o aprendizado, cada modelo possui uma função de perda definida em seu vetor de parâmetros $w$ para cada amostra de dados $j$. A função de perda calcula o erro do valor previsto pelo modelo em relação ao valor desejado $y_j$ de cada amostra. O processo de aprendizagem do modelo é minimizar a função de perda em uma coleção de amostras de dados de treinamento. Para cada amostra de dados $j$, é definida a função de perda $f(w, x_j, y_j)$, que será denotado em resumo como $f_j(w)$. Cada conjunto de dados $D_n$ possuirá uma função de perda local $F_n(w_n)$, que é definida pela equação:




(a) Representação gráfica do algoritmo gradiente descendente.

Gradiente Descendente

| id | peso | altura | Gênero | |
|---|---|---|---|---|
| 0 | 75 kg | 1.73 m | M | i = 1 |
| 1 | 98 kg | 1.61 m | F | i = 2 |
| 2 | 61 kg | 1.56 m | F | i = 3 |
| 3 | 103 kg | 1.98 m | M | i = 4 |
| 4 | 81 kg | 1.85 m | F | i = 5 |
| 5 | 110 kg | 2.10 m | M | i = 6 |

Gradiente Descendente Estocástico

| id | peso | altura | Gênero | |
|---|---|---|---|---|
| 0 | 75 kg | 1.73 m | M | |
| 1 | 98 kg | 1.61 m | F | i = 1 |
| 2 | 61 kg | 1.56 m | F | |
| 3 | 103 kg | 1.98 m | M | |
| 4 | 81 kg | 1.85 m | F | i = 2 |
| 5 | 110 kg | 2.10 m | M | |

(b) No gradiente descendente, os gradientes são calculados a cada amostra contida no conjunto de dados. Já no gradiente descendente estocástico, o cálculo e feito em uma parte menor chamada de lote.

**Figura 3.4. Gradiente descendente e sua variação gradiente descendente estocástico.**

$$F_n(w) = \frac{1}{D_n} \sum_{j \in D_n} f_j(w). \tag{1}$$

É importante ressaltar que para facilitar a visualização das equações será definido $D_n = |D_n|$, em que $|\cdot|$ denota a cardinalidade de um conjunto e $D = \sum_{n=1}^{N} D_n$. Assumindo que $D_n \cap D_{n'} = \emptyset \; \forall \; n \neq n'$, a função de perda global para todos os conjuntos de dados é definida pela equação

$$F(w) = \sum_{n=1}^{N} \frac{D_n}{D} F_n(w). \tag{2}$$

Note que $F(w)$ não pode ser calculada diretamente sem compartilhar informações entre os participantes [Wang et al., 2019].

O problema de aprendizagem é, então, minimizar $F(w)$, ou seja, encontrar

$$w^* = \arg\min F(w). \tag{3}$$

Devido à complexidade inerente à maioria dos modelos de aprendizado de máquina, é impossível encontrar uma solução de forma fechada para Equação 3. Assim, a Equação 3 é frequentemente resolvida usando técnicas de gradiente descendente [Wang et al., 2019].

Atualmente, com o surgimento do *big data*, a quantidade de dados gerados cresce em um ritmo acelerado. Logo, um único servidor não possui poder computacional suficiente para processar os dados envolvido em nesses problemas de otimização. Então foi proposta a otimização distribuída, que divide a tarefa de otimização em pequenas subtarefas para serem processadas por um aglomerado computacional [Zhang e Xiao, 2018]. A otimização distribuída assume que os dados são independentes e identicamente distribuídos (*Independent and Identically Distributed* — IID) [Zhang e Xiao, 2018]. Na otimização em *data center*, os custos de comunicação são pequenos e os custos computacionais são as principais preocupações, com grande parte da ênfase recente no uso de GPUs para reduzir esses custos. Em contraste, na otimização federada [Brendan McMahan et al., 2017], os custos de comunicação são dominantes, já que normalmente há limitação na largura de banda de envio e, dependendo da rede em que o dispositivo se encontra, o envio é praticamente inviável. O termo otimização federada é utilizado para se referir ao problema de otimização implícito no aprendizado federado, que possui uma conexão e também um contraste com a otimização distribuída [Brendan McMahan et al., 2017]. A otimização federada tem várias propriedades-chave que a diferenciam de um problema típico de otimização distribuída que são:

1. Dados não independentes e identicamente distribuídos (*non-Independent and Identically Distributed* — non-IID): São conjunto de dados em que cada conjunto de dados locais não possui a mesma distribuição de probabilidade que os outros e tendem a ser dependentes entre si. Isso se dá devido ao contexto de uso de cada participantes.

2. Dados desbalanceados: Da mesma forma, alguns usuários possuirão conjuntos de dados grandes com muitas amostras e outros com poucas amostras;

3. Grande quantidade de participantes: É esperado que o número de participantes de uma otimização federada seja grande, logo o algoritmo de aprendizado federado deve lidar com quantidade massiva de participantes;

4. Comunicação limitada: Os dispositivos móveis, típicos de um ambiente de aprendizado federado, estão frequentemente desconectados ou em conexões com baixa taxa de transferência.

Os participantes móveis participam do treinamento quando estiverem com bateria carregada, carregando e/ou em uma conexão wi-fi. Ademais, é esperado que cada usuário participe apenas de um pequeno número de rodadas de atualização por dia. Por outro lado, uma vez que qualquer conjunto de dados no dispositivo é pequeno em relação ao tamanho total de todos os conjuntos de dados e *smartphones* modernos possuem processadores relativamente rápidos e muitos possuem GPUs, a computação torna-se essencialmente barata em comparação com os custos de comunicação para muitos modelos

de dispositivos móveis. Portanto, de modo a diminuir o custo de comunicação utiliza-se o poder computacional dos dispositivos dos participantes para diminuir o número de rodadas de comunicação necessárias para treinar o modelo de aprendizado de máquina.

### 3.3.2. Algoritmo *Federated Averaging*

O método mais utilizado para agregação de modelos locais em aprendizado federado é a média federada (*Federated Averaging* — FedAvg) [Brendan McMahan et al., 2017], um método de treinamento baseado em gradiente descendente estocástico (SGD). O FedAvg foi o primeiro algoritmo de agregação de modelos locais para aprendizado federado [Lim et al., 2020]. O funcionamento do FedAvg foi comprovado empiricamente, especialmente para problemas onde a função de perda utilizada é não convexa. Funções convexas são funções em que mínimos locais são também um mínimo global, já as funções não convexas possuem diversos mínimos locais que não são mínimos globais. Contudo, o FedAvg não possui garantias de convergência e pode divergir em cenários práticos quando os dados são heterogêneos [Li et al., 2020].

Em 2018, o FedAvg foi implementado no Gboard [Hard et al., 2018], o teclado inteligente da Google, para melhorar o modelo de previsão de próxima palavra. Desde então, outros estudos exploraram o uso de aprendizado federado em uma série de cenários em que os dados são de natureza sensível, por exemplo, para desenvolver modelos preditivos para diagnóstico de saúde [Brisimi et al., 2018], para promover a colaboração entre hospitais [Li et al., 2019] e Agências governamentais [Verma et al., 2018].

O algoritmo FedAvg se baseia no SGD devido ao avanço das aplicações de aprendizado profundo, em que o método de aprendizado converge na direção do SGD [Brendan McMahan et al., 2017]. No algoritmo FedAvg é selecionada uma porção $S$ de participantes em cada rodada de comunicação e calcula-se o gradiente da perda sobre todos os dados mantidos por esses participantes. Assim, $S$ controla o tamanho do lote global. Caso $S = 1$, isso corresponderá, então, ao gradiente descendente determinístico, já que nesse caso ocorre a seleção de todos os participantes. Cada participante computa $\nabla F_n(w_t)$, que são os gradientes em seus dados locais para o modelo atual $w_t$, e o servidor agrega esses gradientes aplicando a atualização $w_{t+1} \leftarrow w_t - \eta \sum_{n=1}^{N} \frac{D_n}{D} \nabla F_n(w_t)$. O hiper parâmetro $\eta$ é taxa de aprendizado, que influencia diretamente na velocidade da convergência. Então, uma taxa de aprendizado pequena implica numa trajetória suave e com pequenas mudanças nos pesos a cada iteração. Já uma taxa de aprendizado muito alta implica uma mudança maior nos pesos, aumentando a velocidade da convergência. Contudo, pode levar também a oscilações em torno de um mínimo global. Um tipo de agregação equivalente e mais utilizado é $\forall n$, $w_{t+1}^n \leftarrow w_t^n - \eta \nabla F_n(w_t^n)$ e então $w_{t+1} \leftarrow \sum_{n=1}^{N} \frac{D_n}{D} w_{t+1}^n$. Cada cliente realiza localmente uma ou mais etapas de gradiente descendente no modelo atual usando seus dados locais e o servidor, então, obtém uma média ponderada dos modelos resultantes. Uma vez que o algoritmo é escrito dessa forma, pode-se adicionar mais computação para cada cliente, realizando a atualização local várias vezes antes da etapa de agregação. A quantidade de computação é controlada por três parâmetros principais: $S$, a porção de clientes participantes que realizam a computação em cada rodada de comunicação; $\tau$, o número de iterações de treinamento que cada cliente faz em seu conjunto de dados local; e $B$, o tamanho do mini-lote local usado para as atualizações locais do cliente. O pseudocódigo para o FedAvg é apresentado no Algoritmo 1.

**Algoritmo 1: O algoritmo de média federada (*Federated Averaging*) [Brendan McMahan et al., 2017].**

**Input:** Tamanho do mini-lote local $B$, número de épocas locais $\tau$, número de participantes por iteração de comunicação $m$, taxa de aprendizagem $\eta$, quantidade de iterações de comunicação $T$
**Output:** Modelo global $w_G$

1 *[Participante n - Atualiza o modelo local]*
2 **TreinoLocal**($n$, $w$):
3 Divide o conjunto de dados local $D_n$ em mini-lotes de tamanho $B$ criando o conjunto $B_i$
4 **for** *each epoca_local de 1 até* $\tau$ **do**
5 | **for** *each* $b \in B_i$ **do**
6 | | $w_{t+1}^n \leftarrow w_t^n - \eta \nabla F_n(w_t^n; b)$
7 | **end**
8 **end**
9 *[Servidor - Realiza uma média ponderada global das atualizações locais]*
10 Inicializa $w_G^0$
11 **for** *each iteração t de 1 até* $T$ **do**
12 | Escolhe aleatoriamente um subconjunto $S_n \in N$ de tamanho $m$
13 | **for** *each participante* $n \in S_n$ **do**
14 | | $w_{t+1}^n \leftarrow$ **TreinoLocal**($n$, $w_G^t$)
15 | **end**
16 | $w_G^t = \sum_{n=1}^{N} \frac{D_n}{D} w_{t+1}^n$
17 **end**

O algoritmo segue os passos descritos anteriormente. No passo 1, o servidor inicializa a tarefa (linhas 10 - 15). Então, no passo 2, o participante $n$ realiza o treinamento local e otimiza sua função de perda nos mini-lotes de conjunto de dados local (linhas 2 - 8). Um mini-lote se refere a um subconjunto aleatório do conjunto de dados de cada participante. Na t-ésima iteração (linha 16), o servidor minimiza a perda global agregando a média dos gradientes recebidos dos participantes. O processo de treinamento do aprendizado federado continuará até que a função de perda global convirja ou alcance uma acurácia desejável.

### 3.3.3. Arquiteturas de Referência do Aprendizado Federado

Existem três arquiteturas gerais para um sistema de aprendizado federado que são: i) horizontal, ii) vertical e iii) de transferência [Yang et al., 2019]. Cada arquitetura é descrita como uma matriz, na qual as linhas representam o espaço de amostras e as colunas, o espaço de características. O espaço de características será denotado como $X$, o espaço de rótulos como $Y$ e o $I$ será o espaço de amostras. As características $X$, rótulos $Y$ e os IDs de amostras $I$ constituem o campo do conjunto de dados de treinamento completo $(I, X, Y)$.

#### 3.3.3.1. Aprendizado Federado Horizontal

É a arquitetura típica e mais utilizada de um sistema de aprendizado federado. A Figura 3.3 apresenta a arquitetura do aprendizado federado horizontal. A principal característica dessa arquitetura é que os $n$ participantes possuem a mesma estrutura de dados, ou seja, os conjuntos de dados dos participantes compartilham o mesmo espaço de características, porém possuem espaço de amostras diferentes, como pode ser visto na Figura 3.5. Um exemplo é o caso de dois bancos regionais que podem ter grupos de usuários diferentes de suas respectivas regiões e o conjunto de interseção de seus usuários é muito pequeno. Contudo, seu negócio é semelhante, então os espaços de características são praticamente os mesmos [Yang et al., 2019]. Na arquitetura horizontal, os participantes aprendem de forma colaborativa um modelo de aprendizado de máquina com a ajuda de um servidor na nuvem. Uma suposição típica é que os participantes são honestos, enquanto o servidor é honesto, mas curioso; portanto, nenhum vazamento de informação de nenhum participante para o servidor é permitido [Phong et al., 2018]. Nesse contexto, o aprendizado federado horizontal pode ser definido como

$$X_n = X_j; Y_n = Y_j; I_n \neq I_j; \forall D_n, D_j, n \neq j. \tag{4}$$

O aprendizado federado horizontal foi a primeira arquitetura para aprendizado federado [Brendan McMahan et al., 2017]. Bonawitz *et al.* propõem um esquema de agregação segura que protege os parâmetros compartilhados com o servidor [Bonawitz et al., 2017]. Phong *et al.* propõem a utilização de criptografia homomórfica nos parâmetros dos modelos dos participantes [Phong et al., 2018].

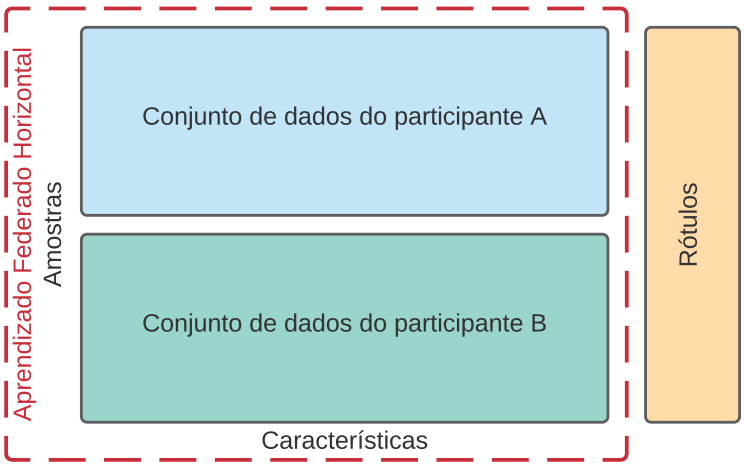



**Figura 3.5. No aprendizado federado horizontal, são utilizados conjuntos de dados que possuem o mesmo espaço de características, mas diferem no espaço de amostras.**

**Analise de segurança**: Na arquitetura do aprendizado federado horizontal, uma das preocupações é proteger os dados sensíveis de um servidor honesto, mas curioso [Yang et al., 2019]. As técnicas mais utilizadas são a criptografia homomórfica [Phong et al., 2018] e a computação multipartidária segura [Bonawitz et al., 2017]. No entanto,

essa arquitetura pode estar sujeita a ataques por um participante mal-intencionado durante o processo de aprendizado colaborativo.

#### 3.3.3.2. Aprendizado Federado Vertical

Alguns algoritmos de aprendizado de máquina que preservam a privacidade para dados particionados verticalmente foram propostos na literatura, incluindo análise estatística cooperativa [Wenliang Du e Atallah, 2001], mineração de regras de associação [Vaidya e Clifton, 2002], regressão linear segura [Karr et al., 2009], classificação [Du et al., 2004] e gradiente descendente [Wan et al., 2007]. O aprendizado federado vertical ou aprendizado federado baseado em características, como pode ser visto na Figura 3.6, é aplicável aos casos em que dois conjuntos de dados compartilham o mesmo espaço de amostras, mas diferem no espaço de características. Por exemplo, duas empresas que operam em uma mesma cidade podem ter clientes semelhantes, mas possuem diferentes informações sobre esses clientes. O aprendizado federado vertical é o processo de agregar essas diferentes características e computar a perda e os gradientes do treinamento de maneira a preservar a privacidade para construir um modelo com dados de ambas as partes de forma colaborativa [Yang et al., 2019].

Trabalhos recentes propõem um esquema de aprendizado federado vertical para treinar um modelo de regressão logística que preserva a privacidade [Hardy et al., 2017, Nock et al., 2018]. Os autores aplicaram a aproximação de Taylor às funções de perda e gradiente descendente de modo que a criptografia homomórfica possa ser adotada para cálculos de preservação de privacidade.

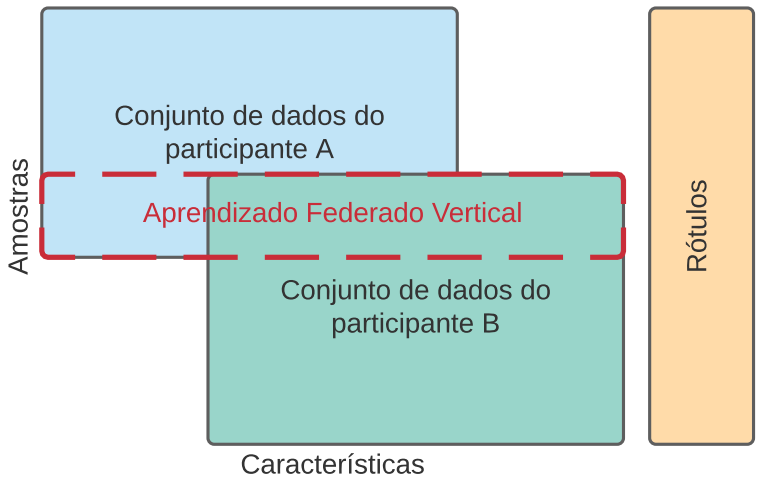



**Figura 3.6. No aprendizado federado vertical, o conjunto de dados possui algumas amostras semelhantes, porém com características diferentes. Antes do início do treinamento, os participantes A e B selecionam, de uma forma segura, a interseção dos espaços de amostras.**

Suponha que as empresas A e B desejem treinar colaborativamente um modelo de aprendizado de máquina em seus sistemas de negócios, cada uma com seus próprios dados. Além disso, a Empresa B também possui dados de rótulos que o modelo precisa prever. Por motivos de privacidade e segurança de dados, A e B não podem trocar

dados diretamente. Para garantir a confidencialidade dos dados durante o processo de treinamento, uma terceira entidade C é envolvida [Yang et al., 2019]. Assume-se que o colaborador C é honesto e não está em conluio com A ou B, mas as partes A e B são honestas, mas curiosas uma para com a outra. Uma terceira parte confiável C é uma suposição razoável, uma vez que a parte C pode ser desempenhada por autoridades como governos ou substituída por um nó de computação seguro [Yang et al., 2019]. O aprendizado federado vertical possui duas partes, como mostrado na Figura 3.7. Na parte 1, é feito um alinhamento entre as entidades utilizando criptografia. Uma vez que os grupos de usuários das duas empresas não são os mesmos, o sistema usa as técnicas de alinhamento de IDs de usuário baseadas em criptografia para confirmar os usuários comuns mútuos em A e B [Liang e Chawathe, 2004, Scannapieco et al., 2007]. Durante o alinhamento das entidades, o sistema não expõe usuários que não se sobrepõem entre si. Na parte 2, é feito o treinamento de modelo criptografado. Depois de determinar as entidades comuns, pode-se usar os dados dessas entidades comuns para treinar o modelo de aprendizado de máquina. O processo de treinamento pode ser dividido nas seguintes quatro etapas:

1. A entidade C cria pares de chaves de criptografia homomórfica aditiva e envia a chave pública para A e B, além de criptografar máscaras para A e B aplicarem.
2. A e B criptografam e trocam seus resultados intermediários utilizando a máscara de C para cálculos de gradiente e perda;
3. A e B calculam gradientes criptografados e adicionam uma máscara adicional. B também calcula a perda criptografada. A e B enviam valores criptografados para C.
4. C descriptografa e envia os gradientes descriptografados e a perda de volta para A e B. A e B desmascaram os gradientes e atualizam os parâmetros do modelo.

O sistema de aprendizado federado vertical ajuda os participantes a estabelecer uma estratégia de "riqueza comum" sem afetar a privacidade dos dados, razão pela qual esse é considerado um sistema de aprendizado colaborativo. Portanto, esse sistema vertical pode ser definido como

$$X_n \neq X_j; Y_n \neq Y_j; I_n = I_j; \forall D_n, D_j, n \neq j. \tag{5}$$

**Análise de segurança**: Um sistema de aprendizado federado vertical normalmente assume participantes honestos, mas curiosos. Em um caso de duas partes, por exemplo, ambas as partes não estão em conluio e uma das partes pode estar comprometida com um adversário. O ponto de segurança é que o adversário pode aprender dados apenas do participante corrupto e não pode aprender dados dos outros participantes. Para facilitar os cálculos seguros entre as duas partes, às vezes um terceiro participante honesto, mas curioso é introduzido, C no caso do exemplo anterior. Nesse caso, assume-se que esse terceiro não está em conluio com nenhuma das partes. A computação multipartidária segura fornece prova de privacidade formal para esses protocolos [Goldreich et al., 2019]. Ao final do aprendizado, cada parte detém apenas os parâmetros do modelo associados às suas características. Portanto, no momento da inferência, as duas partes também precisam colaborar para gerar o modelo de saída.

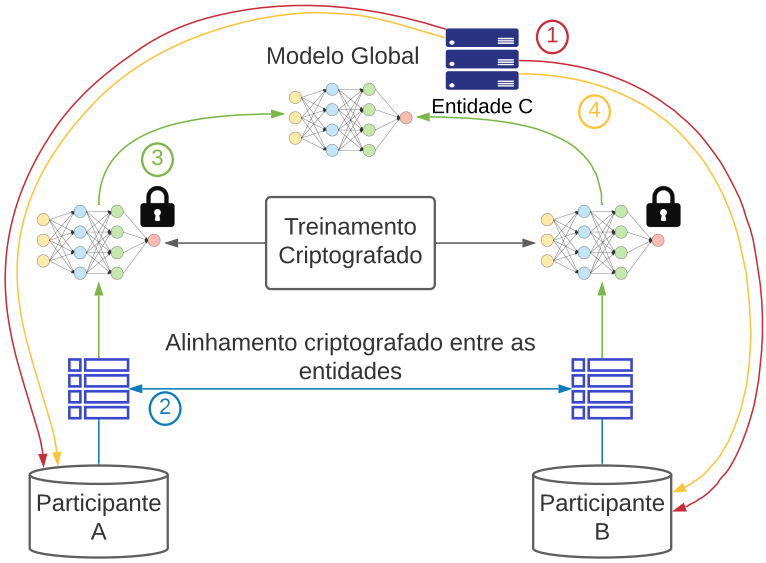



**Figura 3.7. Arquitetura de um sistema de aprendizado federado vertical. No primeiro passo, a entidade cria os pares de chaves e envia a chave pública para os participantes. Já no passo 2, A e B realizam a verificação segura para encontrar interseções em suas amostras, ou seja, clientes em comum. Os participantes enviam seus parâmetros criptografados e com uma máscara para agregação pela entidade C no passo 3. Por fim, no passo 4, a entidade C retorna o resultado para os participantes.**

### 3.3.3.3. Aprendizado por Transferência Federada

O aprendizado por transferência federada se aplica aos cenários nos quais dois conjuntos de dados diferem não apenas nas amostras, mas também no espaço de características. Um exemplo é o caso de duas instituições diferentes operando em países diferentes [Yang et al., 2019]. Nesse caso, as técnicas de aprendizado por transferência [Pan e Yang, 2010] podem ser aplicadas para fornecer soluções para toda a amostra e espaço de características em um ambiente de aprendizado federado, como pode ser visto na Figura 3.8.

Suponha que no exemplo de aprendizado federado vertical discutido anteriormente, as partes A e B tenham apenas um conjunto muito pequeno de amostras sobrepostas e o principal interesse é aprender os rótulos para todo o conjunto de dados do participante A. A arquitetura descrita na seção acima funciona até agora apenas para o conjunto de dados sobreposto. Para estender sua cobertura a todo o espaço amostral, o aprendizado por transferência é proposto [Yang et al., 2019]. O aprendizado por transferência federada não muda a arquitetura geral mostrada na Figura 3.7, porém muda os detalhes dos resultados intermediários que são trocados entre as partes A e B. O aprendizado por transferência normalmente envolve aprender uma representação comum entre as características das partes A e B e minimizar o erro na previsão dos rótulos para a parte do domínio de destino, aproveitando os rótulos na parte do domínio de origem (B, nesse caso). Portanto, os cálculos de gradiente para as partes A e B são diferentes daqueles cenário de aprendizado federado vertical. O aprendizado por transferência federada é

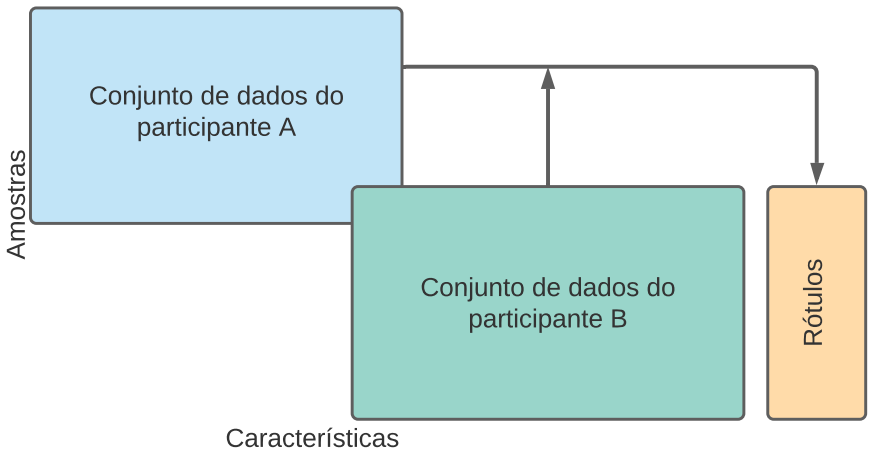



**Figura 3.8. No aprendizado por transferência federada, o participante A quer aprender com todo o conjunto de dados do participante B. Para isso, são realizadas técnicas de aprendizado por transferência, que transfere o conhecimento de um modelo para outro sem expor o conjunto de dados que foi utilizado para treinar o modelo de origem.**

uma extensão para os sistemas de aprendizado federado existentes, pois lida com problemas que excedem o escopo dos algoritmos de aprendizado federado existentes. Assim, o aprendizado por transferência federada é definido por

$$X_n \neq X_j; Y_n \neq Y_j; I_n \neq I_j; \forall D_n, D_j, n \neq j. \tag{6}$$

## 3.4. Principais Aplicações

O aprendizado federado é uma técnica promissora em diversas aplicações móveis como redes veiculares [Samarakoon et al., 2018], detecção de ataques cibernéticos [Nguyen et al., 2018], cache de borda e descarregamento de computação (*Edge Caching and Computation Offloading*) [Wang et al., 2019b] e associação de estações base [Chen et al., 2020]. O aprendizado federado também possui aplicações para dispositivos em redes tradicionais como detecção de intrusão [Preuveneers et al., 2018], vendas [Yang et al., 2019], aplicações de saúde [Brisimi et al., 2018], entre outras. O principal foco dessa seção são as aplicações de aprendizado federado em redes móveis e tradicionais.

### 3.4.1. Ataques Cibernéticos

A detecção de ataques cibernéticos é uma etapa importante para prevenir e mitigar as consequências de ataques em redes. Atualmente, diversos trabalhos utilizando aprendizado de máquina para detecção de ataques foram propostos [Andreoni Lopez et al., 2019]. Entre as diferentes abordagens para detecção de ataques cibernéticos, o aprendizado profundo é considerado a ferramenta mais eficaz para detectar uma ampla gama de ataques com alta acurácia [Nguyen et al., 2018]. Contudo, a acurácia do modelo de detecção irá depender do conjunto de dados utilizado. Os modelos de aprendizado profundo só superam os modelos tradicionais de aprendizado de máquina quando há um conjunto de dados

relativamente grande para o treinamento. No entanto, esses dados podem ser de natureza confidencial. Portanto, alguns modelos de detecção de ataques baseados em aprendizado federado foram propostos para resolver esse problema de privacidade.

Nguyen *et al.* propõem um modelo de detecção de ataque cibernético para uma rede de borda utilizando aprendizado federado [Nguyen et al., 2019]. Nesse trabalho, os *gateways* IoT operam como participantes do aprendizado federado e um provedor de serviços de segurança IoT funciona como o servidor para agregar modelos treinados de forma compartilhada pelos participantes. Os autores apresentam uma aplicação de um sistema de detecção de intrusão utilizando aprendizado federado que foi avaliado em um ambiente real de casas inteligentes e conseguiu detectar com sucesso 95,6% dos ataques em aproximadamente 257 ms sem disparar nenhum alarme falso. No trabalho, assume-se que os participantes são honestos e estão dispostos a contribuir no treinamento utilizando os parâmetros de seus modelos atualizados. No entanto, se alguns dos participantes forem maliciosos, eles poderão corromper toda a detecção de intrusão. Em contrapartida, Preuveneers *et al.* propõem o uso da tecnologia de cadeia de blocos para o gerenciamento dos dados compartilhados pelos participantes [Preuveneers et al., 2018]. Ao usar cadeia de blocos, todas as atualizações incrementais do modelo de detecção de anomalias do aprendizado de máquina são armazenadas no livro-razão facilitando, portanto, a identificação de um participante malicioso. Além disso, com base nos modelos compartilhados dos participantes honestos armazenados no livro-razão, o modelo global pode ser facilmente recuperado por qualquer usuário da rede federada. Os autores identificaram que o uso da cadeia de blocos gera latência no treinamento ao comparar com o treinamento utilizando o algoritmo FedAvg sem cadeia de blocos. O trabalho, portanto, foca apenas em utilizar aprendizado federado para detecção de intrusão, mas não é escopo do trabalho a utilização de modelos eficientes. O trabalho possui uma arquitetura frágil que depende que os nós da rede monitorada pelos participantes do treinamento federado enviem todos os seus fluxos para análise. A Figura 3.9 mostra uma arquitetura de referência para sistemas de detecção de ataque utilizando aprendizado federado.

### 3.4.2. Cache de Borda e Descarregamento de Computação (*Edge Caching and Computation Offloading*)

Para tratar das condições dinâmicas e variáveis temporais em um sistema de rede móvel de borda (*Mobile edge network* — MEC), Wang *et al.* propõem o uso de aprendizado por reforço profundo (*Deep Reinforcement Learning* — DRL) com aprendizado federado para otimizar a memória transitória e as decisões de descarregamento de computação em um sistema MEC [Wang et al., 2019b]. O sistema MEC consiste em um conjunto de equipamentos de usuário cobertos por estações base. Para o armazenamento em memória transitória, o agente DRL toma a decisão de armazenar na memória transitória ou não o arquivo baixado e qual arquivo local substituir, caso ocorra o armazenamento em memória transitória. Para o descarregamento de computação, os equipamentos de usuário podem escolher entre descarregar tarefas de computação para o nó de borda por canais sem fio ou realizar as tarefas localmente. Os estados do sistema MEC incluem as condições da rede sem fio, o consumo de energia do equipamento do usuário e os estados de fila de tarefas, enquanto a função de recompensa é definida como qualidade de experiência (QoE) dos equipamentos do usuário. Dado o grande número de estados e o espaço

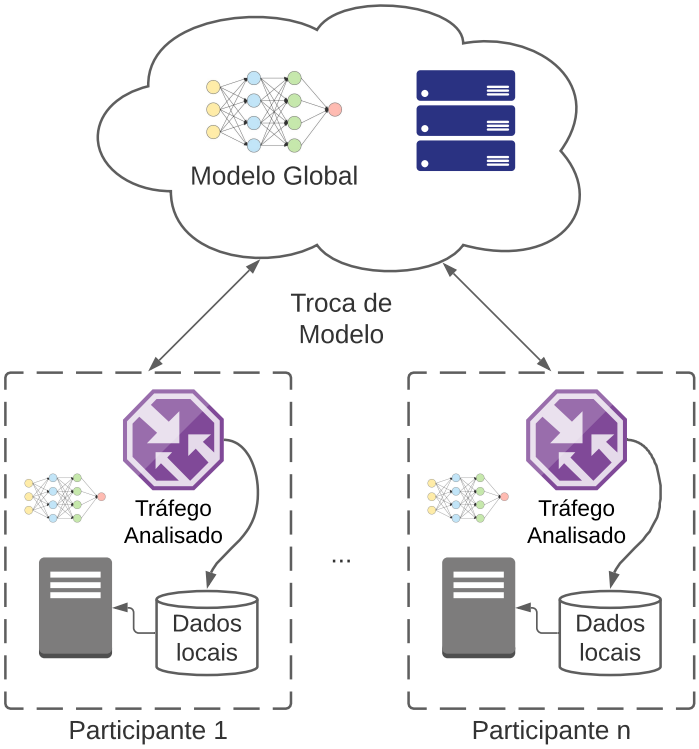



**Figura 3.9. Arquitetura de referência do sistema de detecção de ataque baseado em aprendizado federado proposto em [Preuveneers et al., 2018]. Os participantes, nesse cenário, são sistemas de detecção de intrusão intermediários. O tráfego é enviado para os participantes da rede através de uma API. Esse tráfego é utilizado para treinar um modelo global sem expor os tráfegos pertencentes a cada participante.**

de ações no ambiente MEC, uma abordagem *Double Deep Q-Network* (DDQN) foi adotada para gerenciar em conjunto os recursos de comunicação e computação. Os autores utilizaram DDQN devido ao bom desempenho comprovado em trabalhos anteriores [Sadeghi et al., 2018, He et al., 2018]. Para proteger a privacidade dos usuários, é proposta uma abordagem de aprendizado federado em que o treinamento pode ocorrer com os dados remanescentes nos equipamentos do usuário. Além disso, os algoritmos de aprendizado federado existentes, por exemplo FedAvg [Brendan McMahan et al., 2017], também garantem que o treinamento seja robusto para os dados não balanceados e non-IID dos equipamentos do usuário. Os resultados da simulação mostram que a abordagem DDQN com aprendizado federado atinge resultados semelhantes entre os equipamentos dos usuários em comparação com a abordagem DDQN centralizada (não utilizando aprendizado federado), consumindo menos recursos de comunicação e preservando a privacidade dos usuários participantes. No entanto, as simulações são realizadas apenas com 10 equipamentos de usuários. Se a implementação for expandida para atingir um número maior de equipamentos de usuário heterogêneos, pode haver atrasos significativos no processo de treinamento, especialmente porque o treinamento de um modelo DRL é computacionalmente intenso [Lim et al., 2020]. De maneira semelhante, Ren *et al.* propõem o uso de DRL na otimização de decisões de descarregamento de computação em sistemas IoT [Ren et al., 2019].

### 3.4.3. Associação de Estação Base

Chen *et al.* propõem uma abordagem de redes de estados de eco profundo (*Deep Echo State Networks* — DeepESNs) baseada em aprendizado federado para minimizar quebras de presença (*Breaks In Presence* — BIPs) para usuários de aplicações de realidade virtual (*Virtual Reality* — VR) [Chen et al., 2020, Chung et al., 2010]. Um evento BIP pode ser resultado de atraso na transmissão de informações que pode ser causado quando os movimentos do corpo do usuário obstruem o enlace sem fio. Os BIPs fazem com que o usuário perceba que está em um ambiente virtual, reduzindo assim a qualidade de sua experiência. Logo, uma política de associação de usuário deve ser projetada de forma que os BIPs sejam minimizados. O modelo do sistema consiste em estações base que cobrem um conjunto de usuários de VR. As estações base recebem informações de rastreamento carregadas de cada usuário associado, por exemplo, localização física e orientação, enquanto os usuários baixam vídeos VR para seu uso no aplicativo VR. Para transmissão de dados, os usuários de VR devem se associar a uma das estações base. Assim, um problema de minimização é formulado de forma em que os BIPs são minimizados em relação às localizações e orientações esperadas do usuário de VR. Para obter uma previsão das localizações e orientações dos usuários, a estação base precisa contar com as informações históricas dos usuários. No entanto, a informação histórica armazenada em cada estação base coleta apenas dados parciais de cada usuário, ou seja, um usuário se conecta a várias estações base e seus dados são distribuídos entre elas. Então, na implementação de aprendizado federado realizada pelos autores, cada estação base treina um modelo local usando seus dados parciais. Em seguida, os modelos locais são agregados para formar um modelo global capaz de generalização, ou seja, prever de forma abrangente a mobilidade e orientações de um usuário. Os resultados da simulação mostraram que o algoritmo DeepESN federado atinge menos BIPs, melhorando a experiência dos usuários [Chen et al., 2020].

### 3.4.4. Redes Veiculares

Um pré-requisito essencial para o desenvolvimento de um sistema de transporte inteligente são as redes veiculares de comunicação de baixa latência ultra confiável (*Ultra Reliable Low Latency Communication* — URLLC). No entanto, as técnicas existentes de gerenciamento de recursos de rádio não consideram eventos raros, como grandes comprimentos de fila na distribuição final. Para modelar a ocorrência de tais eventos de baixa probabilidade, Samarakoon *et al.* propõem o uso da teoria dos valores extremos (*Extreme Value Theory* — EVT) [De Haan e Ferreira, 2007], uma ferramenta poderosa que caracteriza as ocorrências de eventos extremos com baixa probabilidade [Samarakoon et al., 2018]. O EVT pode ser usada para modelar a distribuição em termos de três parâmetros conhecidos como localização, escala e forma. O EVT caracteriza assintoticamente as estatísticas de eventos extremos, fornecendo modelos analíticos para analisar o tráfego de rede, atrasos, taxas de pico e comunicação veículo a veículo ultraconfiável em sistemas sem fio. O maior desafio é que a abordagem requer amostras suficientes de informações do estado da fila (*Queue State Information* — QSI) e troca de dados entre veículos para treinar um modelo de inferência eficiente. Os usuários veiculares (*Vehicular Users* — VUEs) têm acesso a um número limitado de QSI que tenha excedido um limiar de sobrecarga, portanto, eles são incapazes de estimar a distribuição final de comprimentos de fila

em toda a rede. Então, os autores propõem uma abordagem de aprendizado federado em que os VUEs treinam o modelo de aprendizagem com dados mantidos localmente e carregam apenas seus parâmetros de modelo atualizados para as unidades de beira de estrada (*Roadside Units* – RSU). O RSU calcula a média dos parâmetros do modelo e retorna um modelo global atualizado para os VUEs. Em uma abordagem síncrona, todos os VUEs carregam seus modelos no final de um intervalo pré-especificado. No entanto, o envio simultâneo por vários veículos pode levar a atrasos na comunicação. Em contraste, para uma abordagem assíncrona, cada VUE apenas avalia e carrega seus parâmetros de modelo após um número predefinido de amostras de QSI coletadas. O modelo global também é atualizado sempre que uma atualização local é recebida, reduzindo assim os atrasos na comunicação. Para reduzir ainda mais a sobrecarga, a otimização de Lyapunov [Neely, 2010] para alocação de energia também é utilizada. Os resultados da simulação mostram que, sob essa estrutura, há uma redução do número de veículos com grandes comprimentos de fila, enquanto o aprendizado federado pode garantir uma troca mínima de dados em relação a uma abordagem centralizada.

Saputra *et al.* propõem uma abordagem chamada de *federated energy demand learning* (FEDL) para gerenciar recursos de energia em estações de carregamento (*Charging Stations* — CSs) para veículos elétricos (*Electric Vehicle* — EVs) [Saputra et al., 2019]. Quando um grande número de EVs se reúne em um CS pode levar a um congestionamento de transferência de energia. Para resolver isso, a energia é fornecida das redes de energia e reservada com antecedência para atender às demandas em tempo real dos EVs [You e Yang, 2014]. Como tal, é necessário prever a demanda de energia para EVs usando dados históricos de carga. No entanto, esses dados são normalmente armazenados separadamente em cada um dos CS que os EVs utilizam e são de natureza privada. Então, na abordagem FEDL proposta, cada CS treina o modelo de previsão de demanda em seu próprio conjunto de dados local antes de enviar apenas as informações de gradiente para o provedor de estação de carga (Charging Station Provider — CSP). Em seguida, as informações de gradiente do CS são agregadas para o treinamento do modelo global. Para melhorar a acurácia do modelo, os CSs são agrupados usando o algoritmo K-médias (*K-Means*) restrito [Bradley et al., 2000] com base em suas localizações físicas. O FEDL baseado em agrupamento reduz a dimensionalidade do conjunto de dados com base na classificação de características úteis e, portanto, a previsão tendenciosa pode ser minimizada [Li et al., 2018]. Os resultados da simulação mostram que a perda de um modelo FEDL agrupado é inferior aos algoritmos convencionais de aprendizado de máquina, por exemplo, regressor *perceptron* multicamadas [Boutaba et al., 2018]. Contudo, a privacidade dos dados do usuário ainda não é protegida por essa abordagem, uma vez que os dados do usuário são armazenados em cada um dos CSs.

## 3.5. Ataque ao Aprendizado Federado

Um dos principais objetivos do aprendizado federado é proteger a privacidade dos participantes do treinamento colaborativo, pois os participantes só precisam compartilhar os parâmetros do modelo treinado em vez de compartilhar seus dados locais. Entretanto, esse processo é suscetível a uma variedade de ataques, como o envenenamento de dados ou do modelo, em que um participante malicioso pode enviar parâmetros incorretos ou modelos corrompidos para enviesar o processo de aprendizagem durante a agregação glo-

bal. Consequentemente, o modelo global será atualizado incorretamente e todo o sistema de aprendizagem será corrompido. Em particular, isso anula o propósito do aprendizado federado, uma vez que o modelo global resultante pode ser corrompido ou os participantes podem ter a privacidade comprometida durante o treinamento do modelo. Esta seção discute os ataques emergentes sobre o aprendizado federado, bem como algumas contramedidas propostas para lidar com esses ataques.

### 3.5.1. Envenenamento do Conjunto de Dados

No aprendizado federado, um participante treina seu modelo local com seus dados e envia o modelo treinado ao servidor para posterior processamento. Nesse caso, é inviável para o servidor verificar se os parâmetros de cada participante são reais [Lim et al., 2020]. Assim, um participante malicioso pode envenenar o modelo global, criando dados rotulados de forma errada para treinar o modelo global com o objetivo de gerar parâmetros falsificados e enviesar o treinamento do modelo global [Fung et al., 2018]. Por exemplo, um participante malicioso pode gerar uma série de amostras falsas, e usá-las para treinar o modelo global para atingir seu objetivos.

Fung *et al.* investigam os impactos de um ataque de envenenamento de dados baseado no ataque Sybil em um sistema de aprendizado federado [Fung et al., 2018]. Em particular, no ataque Sybil, um participante malicioso tenta melhorar a eficácia do envenenamento de dados no treinamento do modelo global, criando vários participantes falsos. Os autores mostraram que com apenas dois participantes falsos, a taxa de sucesso do ataque pode atingir até 96,2%, tornando o modelo global incapaz de classificar corretamente os rótulos. Para mitigar os ataques Sybil, os autores propõem uma estratégia de defesa, chamada de *FoolsGold*. A ideia principal dessa abordagem é que os participantes honestos possam ser diferenciados dos participantes falsos com base em seus gradientes atualizados. No aprendizado federado, os dados de treinamento de cada cliente têm uma distribuição única e não são compartilhados. Os atacantes sybils compartilham um objetivo comum e contribuirão com atualizações que são semelhantes entre si, diferente dos clientes honestos. O FoolsGold usa essa suposição para modificar as taxas de aprendizagem de cada cliente em cada iteração de comunicação. A proposta é manter a taxa de aprendizado de clientes que fornecem atualizações exclusivas, enquanto reduz a taxa de aprendizado de clientes que contribuem repetidamente com atualizações de gradiente de aparência semelhante. Com o FoolsGold, o sistema pode se defender do ataque de envenenamento de dados Sybil com mudanças mínimas no processo de aprendizado federado convencional e sem exigir nenhuma informação auxiliar ao processo de aprendizagem. Através de resultados de simulações em três conjuntos de dados diversos (MNIST [Lecun et al., 1998], KDDCup [Cup, 1999], Amazon Reviews [Asuncion e Newman, 2007]), os autores mostram que o FoolsGold pode mitigar o ataque sob uma variedade de condições, incluindo diferentes distribuições de dados de participantes, variando os alvos de envenenamento e várias estratégias de ataque.

### 3.5.2. Envenenamento do Modelo de Aprendizado de Máquina

Ao contrário dos ataques de envenenamento de dados, que visam gerar dados falsos para causar impactos adversos ao modelo global, um ataque de envenenamento de modelo tenta envenenar diretamente o modelo global enviando parâmetros falsos para a

agregação. Conforme apresentado em trabalhos anteriores [Bhagoji et al., 2019, Bagdasaryan et al., 2020], os ataques de envenenamento de modelo de aprendizado de máquina são muito mais eficazes do que os de ataques de envenenamento de dados para aprendizado federado em grande escala com muitos participantes. O motivo é que, para ataques de envenenamento de dados, as atualizações de um participante malicioso são dimensionadas com base em seu conjunto de dados e no número de participantes do ambiente federado. No entanto, para ataques de envenenamento de modelo, um participante malicioso pode modificar o modelo atualizado, que é enviado ao servidor para agregação, diretamente. Como resultado, mesmo com um único invasor, todo o modelo global pode ser envenenado. Os resultados da simulação confirmam que mesmo um adversário altamente restrito com dados de treinamento limitados pode atingir alta taxa de sucesso na execução de ataques de envenenamento de modelo [Bhagoji et al., 2019]. Portanto, soluções para proteger o modelo global de ataques de envenenamento de modelo devem ser desenvolvidas.

Bhagoji *et al.* sugerem algumas soluções para prevenir ataques de envenenamento de modelo [Bhagoji et al., 2019]. Primeiro, com base nas atualizações dos parâmetros do modelo de um participante, o servidor verifica se o modelo compartilhado contribui na melhoria do desempenho do modelo global ou não. Caso contrário, o participante será marcado como um invasor em potencial e, após algumas rodadas de observação do modelo atualizado desse participante, o servidor pode determinar se esse é um participante malicioso ou não. A segunda solução é baseada na comparação entre os modelos atualizados compartilhados pelos participantes. Se um modelo atualizado de um participante for muito diferente dos demais, o participante é potencialmente malicioso. Então, o servidor continuará observando as atualizações desse participante antes de determinar se esse é um usuário malicioso ou não. No entanto, os ataques de envenenamento de modelo são extremamente difíceis de prevenir porque, ao treinar com grande número de participantes simultâneos, é inviável avaliar a melhoria de cada participante individualmente e em comparação com as dos demais. Como tal, soluções mais eficientes precisam ser investigadas.

Bagdasaryan *et al.* apresentam um ataque de envenenamento de modelo mais eficaz, que demonstrou atingir 100% de acurácia no ataque em apenas uma única rodada de aprendizagem [Bagdasaryan et al., 2020]. Um participante malicioso pode compartilhar seu modelo envenenado, que não apenas é treinado para enviesar o modelo, mas também contém uma função de *backdoor*. Essa função, proposta pelos autores, é chamada de *constrain-and-scale* e pode comprometer um ou mais participantes. Esse algoritmo permite ao invasor produzir um modelo com alta acurácia tanto na tarefa principal quanto na *backdoor*, ainda que não seja rejeitado pelo detector de anomalias do agregador. Essa função maliciosa pode fazer com que o modelo global classifique incorretamente, sem a necessidade de modificar o conjunto de dados local do participante malicioso. Por exemplo, uma função *backdoor* de classificação de imagem pode injetar um rótulo escolhido pelo invasor em todas as imagens com algumas características específicas, por exemplo, todos os cães com listras pretas podem ser classificados erroneamente como gatos. Os resultados das simulações mostraram que este ataque supera os ataques convencionais de envenenamento de dados em aprendizado federado. Bagdasaryan *et al.* mostram que em uma tarefa de previsão de palavras com 80.000 participantes no total, o comprometimento

de apenas oito participantes é o suficiente para atingir 50% de acurácia na classificação maliciosa.

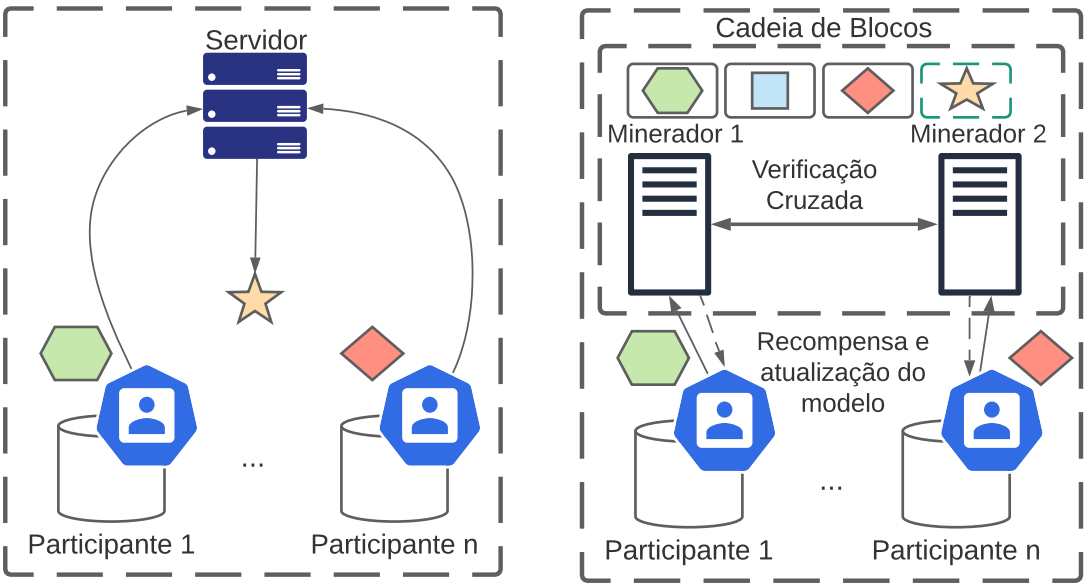
### 3.5.3. Ataque *Free-Riding*

O *free-riding* é outro ataque em aprendizado federado que ocorre quando um participante deseja se beneficiar do modelo global sem contribuir com o processo de aprendizagem. O participante malicioso, chamado de *free-rider*, finge ter um número pequeno de amostras para treinar ou seleciona um pequeno conjunto menor de seu conjunto de dados real para o treinamento, para economizar seus recursos computacionais. Como resultado, os participantes honestos precisam contribuir com mais recursos computacionais no processo de treinamento do modelo global. Para resolver esse problema, Kim *et al.* apresentam uma arquitetura de aprendizado federado baseada em cadeia de blocos, chamada BlockFL, na qual as atualizações do modelo local dos participantes são trocadas e verificadas por meio da tecnologia de cadeia de blocos [Kim et al., 2018]. Cada participante treina e envia o modelo local treinado para seu minerador associado na cadeia de blocos e, em seguida, recebe uma recompensa que é proporcional ao número de amostras de dados treinados, conforme ilustrado na Figura 3.10(b). Dessa forma, o arcabouço proposto não só evita participantes *free-riders*, mas também incentiva todos os participantes a contribuir para o processo de aprendizagem. Um modelo semelhante, também baseado em cadeia de blocos é introduzido por Weng *et al.*, visando fornecer confidencialidade de dados, capacidade de auditoria computacional e incentivos para os participantes do aprendizado federado [Weng et al., 2019]. No entanto, a utilização da tecnologia de cadeia de blocos implica a implementação e manutenção de mineradores para operar a cadeia de blocos. Além disso, os protocolos de consenso usados em redes de cadeia de blocos, como a prova de trabalho (*Proof-of-Work* — PoW), tendem a causar longo atraso na troca de informações e, assim, não são apropriados para implementar modelos de aprendizado federado.

### 3.5.4. Inversão do Modelo e Inferência dos Gradientes

A inversão de modelo é o ataque em que um adversário, em posse de um modelo treinado para prever uma variável específica, utiliza os pesos desse modelo para prever o conjunto de dados usados como entrada para treinar esse modelo, caracterizando assim um ataque à privacidade de atributos [Fredrikson et al., 2014]. O ataque busca tirar proveito da correlação entre o alvo, que seriam as características desconhecidas, e o resultado previsto pelo modelo. Esse ataque, em aprendizado federado, pode ser executado pelo servidor agregador que possui os modelos locais atualizados dos participantes. O servidor agregador pode ser honesto, mas curioso e, então, o servidor mantém a funcionalidade do ambiente de aprendizado federado, mas está disposto a descobrir os dados dos clientes. Fredrikson *et al.* propuseram o ataque de inversão de modelo para recuperar imagens de um modelo de reconhecimento facial [Fredrikson et al., 2015]. Os autores desenvolveram uma nova classe de ataque de inversão de modelo que explora os parâmetros de treinamentos revelados com as previsões.

Triastcyn e Faltings propõem um arcabouço chamado FedGP (*Federated Generative Privacy* – Privacidade Generativa Federada) [Triastcyn e Faltings, 2020]. A ideia principal dessa abordagem, é treinar redes adversárias gerativas (*Generative Adversarial*




(a) Aprendizado Federado tradicional, ou seja, média federada (FedAvg).

(b) Arquitetura da proposta BlockFL, onde o modelo global gerado em cada iteração de comunicação é verificado pelos mineradores e, posteriormente, salvo na cadeia de blocos.

**Figura 3.10. Comparação entre a arquitetura do algoritmo FedAvg e BlockFL.**

*Networks* — GANs) em clientes para produzir dados artificiais que substituam os dados reais dos clientes. Como alguns clientes podem ter dados insuficientes para treinar um GAN localmente, foi treinado um modelo de GAN federado. Dessa forma, os dados do usuário sempre permanecem em seus dispositivos. Além disso, o GAN federado produzirá amostras da distribuição comum entre usuários e não de um único usuário, o que aumenta a privacidade. A Figura 3.11 mostra a arquitetura do arcabouço. Os autores avaliaram a proteção fornecida executando o ataque de inversão de modelo e mostrando que o treinamento com o GAN federado reduz o vazamento de informações.

Zhang *et al.* propõem um esquema de criptografia homomórfica para preservar os parâmetros dos modelos dos usuários participantes de cada iteração de comunicação do aprendizado federado [Zhang et al., 2019]. Os autores propõem o *Privacy-Enhanced Federated Learning* (PEFL) para proteger os gradientes de um servidor não confiável. Isso é viabilizado principalmente pela criptografia dos gradientes locais dos participantes com o sistema de criptografia homomórfico de Paillier. Para reduzir os custos de computação do sistema de criptografia, é utilizado o método *Distributed Selective Stochastic Gradient Descent* (DSSGD) [Dean et al., 2012] na fase de treinamento local para diminuir o custo computacional da criptografia distribuída. Além disso, os gradientes criptografados são usados para agregação de soma segura no lado do servidor conforme mostrado na Figura 3.12. Dessa forma, o servidor não confiável aprende apenas as estatísticas agregadas de todas as atualizações dos participantes, enquanto as informações particulares de cada indivíduo estarão protegidas. Para a análise de segurança, os autores provam teoricamente que o esquema é seguro. Os resultados experimentais demonstram que o PEFL tem baixos custos de computação e, ao mesmo tempo, atinge alta precisão nas configurações do aprendizado federado. Zhang *et al.* também analisam o tempo para criptografar os parâ-

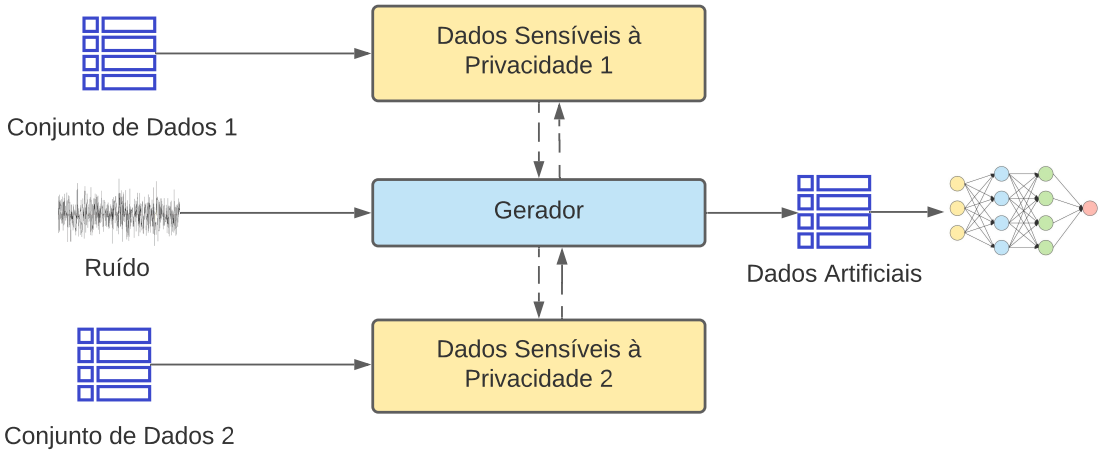



**Figura 3.11. Arquitetura do FedGP com dois participantes. Os dados confidenciais são usados para treinar um GAN que posteriormente produz um conjunto de dados artificial privado.**

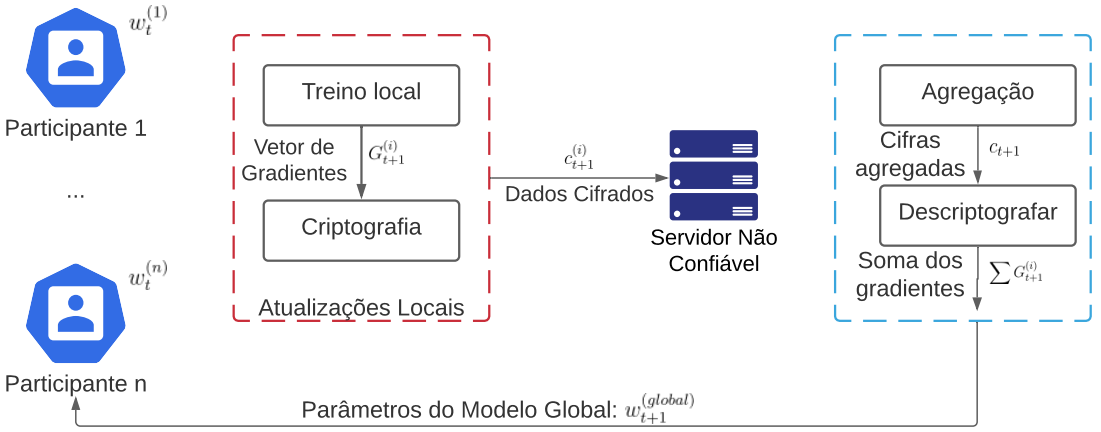



**Figura 3.12. Na Arquitetura do PEFL, os participantes enviam os dados criptografados utilizando criptografia homomórfica e o servidor realiza a agregação retornando os pesos agregados a todos os participantes.**

metros de um fragmentos dos pesos, utilizando DSSGD, e concluem que é pequeno, por vezes menor que 1 segundo [Zhang et al., 2019]. Contudo, os autores utilizam equipamentos de última geração nas avaliações, o que não corresponde à realidade do ambiente móvel real.

## 3.6. Desafios e Oportunidades de Pesquisa em Aprendizado Federado

Os desafios tornam a execução do aprendizado federado distinta de outros problemas clássicos, como aprendizado distribuído em configurações de *data center* ou análises de dados privados tradicionais. Os três principais desafios em aprendizado federado são:

1. o custo de comunicação - O ambiente de aprendizado federado compreende um grande número de dispositivos, por exemplo, milhões de *smartphones*, e a comunicação da rede pode ser mais lenta do que a computação local, devido à limitação de

recursos, como largura de banda e energia [Yang et al., 2019]. Para reduzir a comunicação no ambiente federado, dois aspectos principais devem ser considerados: i) reduzir o número total de rodadas de comunicação e ii) diminuir o tamanho das mensagens transmitidas em cada rodada;

2. a heterogeneidade de dispositivos - Os recursos de armazenamento, computação e comunicação de cada dispositivo do ambiente federado podem diferir devido à variabilidade no hardware (CPU e memória), conectividade de rede (2G, 3G, 4G, 5G e Wi-Fi) e energia (nível de bateria) [Yang et al., 2019]. Além disso, o tamanho da rede e as restrições relacionadas aos sistemas de cada dispositivo normalmente resultam em apenas uma pequena fração dos dispositivos ativos ao mesmo tempo [Bonawitz et al., 2019]. É comum que um dispositivo ativo falhe em uma determinada iteração devido a restrições de conectividade ou energia [Bonawitz et al., 2019]. A heterogeneidade de dispositivos intensifica os desafios, como mitigação de retardatários e tolerância a falhas. Os métodos de aprendizado federado desenvolvidos devem, portanto, i) prever a falha dos participantes, ii) tolerar hardware heterogêneo e iii) ser robustos o suficiente para que a falha dos participantes não afete a agregação;

3. a heterogeneidade estatística dos dados - Dispositivos frequentemente geram e coletam dados de maneira non-IID no ambiente federado. Usuários de telefones celulares, por exemplo, têm uso variado de linguagem no contexto de uma tarefa de previsão de próxima palavra. Além disso, o número de pontos de dados (vetor de características) entre os dispositivos pode variar significativamente e pode haver uma estrutura estatística subjacente que captura a relação entre os dispositivos e suas distribuições associadas [Yang et al., 2019]. Esse paradigma de geração de dados viola as suposições de dados independentes e identicamente distribuídos (i.i.d.) usadas frequentemente na otimização distribuída e pode adicionar complexidade na modelagem de problemas, análise teórica e avaliação empírica de soluções.

### 3.6.1. Custo de Comunicação

Comunicação é um ponto crucial em aprendizado federado. No aprendizado federado, é necessária uma série de rodadas de comunicação entre os participantes e o servidor para atingir a acurácia desejada. O treinamento de modelo de aprendizado profundo complexo, como em redes neurais convolucionais, pode compreender milhões de parâmetros em cada atualização [He et al., 2016]. A alta dimensionalidade das atualizações resulta na ocorrência de altos custos de comunicação e leva a um gargalo no treinamento. O gargalo é agravado devido a i) condições de rede dos dispositivos participantes [Wang et al., 2019] e ii) assimetrias nas conexões à Internet em que a taxa de envio é menor do que a taxa de recebimento, resultando em atrasos no envio dos modelos locais dos participantes [Konečnỳ et al., 2016]. Então, alguns trabalhos na literatura [Liu et al., 2019, Yao et al., 2018, Wang et al., 2019, Konečnỳ et al., 2016, Caldas et al., 2018, Tao e Li, 2018] visam melhorar a comunicação do aprendizado federado de três formas: i) aumentando a computação local, reduzindo, assim, a necessidade de rodadas de comunicação; ii) realizando a compressão do modelo local, reduzindo o tamanho dos dados enviados ao

servidor; iii) realizando atualizações com base na importância, onde é enviado somente os parâmetros que tiveram mudanças relevantes durante o treinamento local.

Para diminuir o número de rodadas de comunicação, a computação pode ser realizada nos dispositivos finais dos participantes antes de cada iteração de comunicação para agregação global. Por exemplo, o protocolo FedAvg [Brendan McMahan et al., 2017] considera duas maneiras de aumentar a computação nos dispositivos participantes, reduzindo o volume de dados a serem transmitidos: i) aumentar o paralelismo, selecionando mais participantes para participar de cada rodada de treinamento e ii) aumentar o esforço computacional por participante, no qual cada participante realiza mais atualizações locais antes da comunicação para agregação global.

Como uma extensão do FedAvg, Liu *et al.* validam um conceito semelhante [Liu et al., 2019]. Liu *et al.* propõem o algoritmo *Federated Stochastic Block Coordinate Descent* (FedBCD), no qual cada dispositivo participante realiza várias atualizações locais antes da comunicação para agregação global. A garantia de convergência é fornecida com uma calibração aproximada do número ideal de atualizações locais calculado a cada intervalo de comunicação. Semelhantemente, Yao *et al.* propõem computação aumentada em cada dispositivo participante, adotando um modelo de dois fluxos(*two-stream model*) [Yao et al., 2018].O modelo de dois fluxos é comumente usado no aprendizado por transferência e adaptação de domínio [Long et al., 2015]. Durante cada rodada de treinamento, o modelo global é recebido pelos participantes e fixado como referência no processo de treinamento local. Durante a atualização local, o participante aprende não apenas com dados locais, mas também com de outros participantes utilizando o modelo global como referência. Isso é feito por meio da incorporação da Discrepância Média Máxima (Maximum Mean Discrepancy — MMD) na função de perda. O MMD mede a distância entre as médias de duas distribuições de dados [Long et al., 2015]. Ao minimizar a perda do MMD entre os modelos local e global, o participante pode extrair características mais generalizadas do modelo global, acelerando assim a convergência do processo de treinamento e reduzindo rodadas de comunicação. As simulações foram realizados utilizando os conjuntos de dados CIFAR-10 e MNIST e usando modelos de aprendizado profundo, como redes neurais convolucionais. Os resultados mostram que o aprendizado federado de dois fluxos proposto pode atingir a acurácia desejável em 20% menos rodadas de comunicação, mesmo quando os dados são non-IID [Yao et al., 2018].

Wang *et al.* propõem um algoritmo para determinar a frequência de agregação global para que os recursos disponíveis sejam usados com maior eficiência [Wang et al., 2019]. Para isso, os autores analisam o limite de convergência do aprendizado federado baseada no gradiente descendente a partir de uma perspectiva teórica, na qual um novo limite de convergência é proposto, que incorpora a distribuição dos dados non-iid entre os nós e um número arbitrário de atualizações locais entre duas agregações globais. Utilizando esse limite de convergência teórico, os autores propõem um algoritmo de controle que aprende a distribuição dos dados, dinamicidade do sistema e características do modelo. Com base no algoritmo proposto, o sistema adapta dinamicamente a frequência da agregação global em tempo real para minimizar a perda de aprendizado. Por fim, os autores avaliam o desempenho do algoritmo de controle proposto através de experimentos utilizando conjuntos de dados reais, tanto em um cenário com protótipo de hardware quanto em um ambiente simulado. Os resultados confirmam que a abordagem proposta

fornece desempenho próximo ao ideal para diferentes distribuições de dados, vários modelos de aprendizado de máquina e configurações do sistema com diferentes números de nós de borda. O sistema obtém uma relação de compromisso desejável entre atualização local e agregação global, de modo a minimizar a função de perda.

Embora os métodos de atualização local possam reduzir o número total de rodadas de comunicação, esquemas de compressão de modelo também podem ser utilizados para reduzir o volume de dados em trânsito no aprendizado federado. Alguns exemplos desses esquemas de compressão incluem "esparsificação´´, subamostragem e quantização, os quais reduzem significativamente o tamanho das mensagens comunicadas em cada rodada. Esses métodos foram amplamente estudados, tanto empiricamente quanto teoricamente, na literatura para treinamento distribuído em ambientes de *datacenter*. Konevcny *et al.* propõem duas maneiras de atualizações do modelo local, a atualização estruturada (*Structured update*) e a atualização esboçada (*Sketched update*) [Konečnỳ et al., 2016]. Os modelos são propostos para reduzir o tamanho das atualizações enviadas dos participantes para o servidor durante cada rodada de comunicação. As atualizações estruturadas restringem as atualizações dos participantes a terem uma estrutura pré-especificada, com baixa classificação (*Low rank*) e máscara aleatória (*Random mask*). Na estrutura de baixa classificação, cada atualização é forçada a ser o produto de duas matrizes. Dessas duas matrizes, uma matriz é gerada aleatoriamente e mantida constante durante cada rodada de comunicação, enquanto a outra é otimizada. Assim, a matriz aleatória pode ser neste caso compactada na forma de uma semente aleatória e apenas a matriz otimizada precisa ser enviada ao servidor. Para a estrutura de máscara aleatória, cada atualização local é restrita a ser uma matriz esparsa seguindo um padrão esparso aleatório predefinido, gerado de forma independente durante cada rodada. Apenas as entradas diferentes de zero devem ser enviadas para o servidor. Por outro lado, as atualizações de esboço referem-se à abordagem de codificar a atualização em uma forma compactada antes da comunicação com o servidor, que subsequentemente decodifica as atualizações antes da agregação. Um exemplo de atualização esboçada é a abordagem de subamostragem, na qual cada participante comunica apenas um subconjunto aleatório da matriz de atualização. O servidor então calcula a média das atualizações sub-amostradas para obter uma estimativa imparcial da média real. Outro exemplo, a abordagem de quantização probabilística [Han et al., 2015] prevê que as matrizes de atualização são vetorizadas e quantizadas para cada escalar. Para reduzir o erro de quantização, uma rotação aleatória estruturada, que é o produto de uma matriz Walsh-Hadamard e uma matriz diagonal binária, é aplicada antes da quantização [Suresh et al., 2017].

Os resultados da simulação realizada por Konevcny *et al.* na tarefa de classificação de imagens utilizando o conjunto de dados CIFAR-10 mostram que, para atualizações estruturadas, a máscara aleatória obteve desempenho melhor do que a abordagem de classificação baixa [Konečnỳ et al., 2016]. A abordagem de máscara aleatória também atinge maior acurácia do que atualizações esboçadas, uma vez que esta envolve a remoção de algumas informações obtidas durante o treinamento. No entanto, a combinação de todas as três ferramentas de esboço, ou seja, subamostragem, quantização e rotação, pode atingir maior taxa de compressão e convergência mais rápida, embora com alguns sacrifícios de acurácia. Caldas *et al.* realizaram uma extensão dos estudos de Konevcny *et al.*, propondo uma compressão com perdas para reduzir os custos de comunicação [Caldas et al., 2018].

Outra forma de poupar a quantidade de bytes que precisa ser transmitido no aprendizado federado é chamada de atualização baseada em importância. Nessa técnica, considera-se o fato de que a maioria dos valores dos parâmetros de um modelo de redes neurais profundas são esparsamente distribuídos e próximos de zero [Strom, 2015]. Dessa forma, Tao *et al.* propõem o algoritmo de gradiente descendente estocástico de borda (*edge Stochastic Gradient Descent* — eSGD), que seleciona apenas uma pequena fração de gradientes importantes para enviar ao servidor do aprendizado federado para atualização de parâmetros durante cada rodada de comunicação [Tao e Li, 2018]. O algoritmo eSGD acompanha os valores de perda em duas iterações de treinamento consecutivas. Se o valor de perda da iteração atual for menor que a iteração anterior, isso implica que os gradientes de treinamento atuais e os parâmetros do modelo são importantes para a minimização da perda de treinamento, portanto, seus respectivos pesos ocultos são atribuídos a um valor positivo. Além disso, o gradiente também é comunicado ao servidor para atualização dos parâmetros. Por outro lado, se a perda aumenta em comparação à iteração anterior, outros parâmetros são selecionados para serem atualizados com base em seus valores de pesos ocultos. Um parâmetro com maior valor de peso oculto tem maior probabilidade de ser selecionado, pois foi rotulado como importante várias vezes durante o treinamento. O peso oculto é sempre um valor real positivo e os parâmetros que possuem pesos ocultos pequenos podem retardar a convergência. Os valores de gradiente de cada rodada são acumulados como valores residuais. Uma vez que os resíduos surgem de diferentes iterações de treinamento, cada atualização do resíduo é ponderada com um fator de desconto usando a técnica de correção de *momentum*. Quando o gradiente residual acumulado atinge um limiar, eles são escolhidos para substituir as coordenadas dos gradientes menos importantes de acordo com os valores de pesos ocultos. Os resultados da simulação mostraram que eSGD com uma taxa de remoção de 50% atinge maior acurácia do que o algoritmo *thresholdSGD* [Strom, 2015], que usa um valor de limiar fixo para determinar quais coordenadas de gradiente devem ser descartadas. Assim, o eSGD reduz o tamanho do gradiente comunicado, mas ainda sofre com a perda de acurácia em comparação com as abordagens SGD padrão. Por sua vez, Wang *et al.* propõem o algoritmo Aprendizado federado com redução de comunicação (*Communication-Mitigated Federated Learning* — CMFL) que carrega apenas atualizações de modelos locais relevantes para reduzir os custos de comunicação, garantindo a convergência global [Wang et al., 2019]. Em cada iteração, a atualização do modelo local de um participante é primeiro comparada com o modelo global para identificar se a atualização é relevante. Os resultados da simulação mostraram que o CMFL requer 3,47 vezes menos rodadas de comunicação para atingir 80% de acurácia para classificação de imagem utilizando o conjunto de dados MNIST e 13,97 vezes menos para previsão de próxima palavra, ambos em comparação com o algoritmo FedAvg, que é usualmente utilizado como parâmetro de comparação.

### 3.6.2. Heterogeneidade de Dispositivos

No ambiente de aprendizado federado, há uma variação significativa nas características dos dispositivos dos participantes na rede, pois esses dispositivos podem possuir hardware, conectividade de rede e nível de bateria distintos. Essas características de sistema tornam problemas como atrasos significativamente mais prevalentes do que em ambientes típicos de *datacenter*. Para resolver esse problema, soluções foram propostas.

A seleção de participantes refere-se à seleção de dispositivos para participar de cada rodada de comunicação. Normalmente, um conjunto de participantes é selecionado aleatoriamente pelo servidor. O progresso do treinamento do aprendizado federado é limitado pelo tempo de treinamento dos dispositivos participantes mais lentos [Lim et al., 2020], ou seja, retardatários. Novos protocolos de seleção de participantes são, portanto, investigados para resolver o gargalo de treinamento em aprendizado federado. Nishio *et al.* propõem um novo protocolo de aprendizado federado chamado FedCS [Nishio e Yonetani, 2019]. A proposta é um arcabouço MEC em que a operadora do MEC é o servidor de aprendizado federado que coordena o treinamento em uma rede celular que compreende dispositivos móveis participantes que possuem recursos heterogêneos. O servidor primeiro conduz uma etapa de solicitação de recursos para reunir informações como estados de canais sem fio e recursos de computação de um subconjunto de participantes selecionados aleatoriamente. Com base nessas informações, o operador do MEC seleciona o número máximo possível de participantes que pode concluir o treinamento dentro de um prazo pré-estabelecido para a fase de agregação global subsequente. Os resultados da simulação mostraram que, em comparação com o protocolo FedAvg, que contabiliza apenas o prazo de treinamento, o FedCS obtém maior acurácia, pois envolve mais participantes em cada rodada de treinamento ao invés de um número fixo. Da mesma forma, Kang *et al.* consideram sobrecargas de sistemas incorridas a cada dispositivo ao projetar mecanismos de incentivo para encorajar dispositivos com dados de alta qualidade a participarem do processo de aprendizagem [Kang et al., 2019a]. Embora esses métodos se concentrem principalmente na variabilidade dos sistemas para realizar a amostragem ativa, é vantajoso considerar a amostragem ativa de um conjunto de dispositivos pequenos, mas suficientemente representativos, com base na estrutura estatística dos dados.

Outra forma de tentar sobrepor atrasos gerados pela heterogeneidade dos dispositivos é utilizar esquemas de aprendizado federado baseado em uma comunicação assíncrona. Configurações tradicionais de *data centers* são baseadas em esquemas síncronos, nos quais nós trabalhadores esperam uns aos outros para sincronização, e assíncronos, em que os nós trabalhadores executando de forma independente sem sincronização são usados para paralelizar algoritmos de otimização iterativa [Li et al., 2020]. Os esquemas síncronos são simples e garantem um modelo computacional serial equivalente trivial, mas também são mais suscetíveis a atrasos devido à variabilidade dos dispositivos. Os esquemas assíncronos são abordagens usadas para mitigar retardatários em ambientes heterogêneos, particularmente em sistemas de memória compartilhada. No entanto, dependem de suposições de atraso limitado para controlar o grau de desatualização. O algoritmo FedAvg [Brendan McMahan et al., 2017] agrega parâmetros de forma síncrona e é, portanto, suscetível ao efeito retardatário, já que cada rodada de treinamento progride na velocidade do dispositivo mais lento, uma vez que o servidor espera que todos os dispositivos concluam o treinamento local antes que a agregação global possa ocorrer. Além disso, o algoritmo não considera participantes que ingressam quando a rodada de treinamento já está em andamento.

Sprague *et al.* constataram empiricamente que uma abordagem assíncrona é robusta para participantes que ingressam durante a rodada de treinamento em progresso, bem como quando a federação envolve dispositivos participantes com recursos de processamento heterogêneos [Sprague et al., 2019]. No entanto, a convergência do modelo é

significativamente atrasada quando os dados são non-IID e desbalanceados. Como uma melhoria, Xie *et al.* propõem o algoritmo FedAsync, em que cada atualização local recém-recebida é ponderada de forma adaptativa de acordo com seu grau de obsolência, o qual é definido como a diferença entre a época atual e a iteração à qual pertence a atualização recebida [Xie et al., 2019]. Assim, uma atualização desatualizada de um retardatário que está obsoleta, pois deveria ter sido recebida em rodadas de treinamento anteriores, possui menor peso, mas ainda é considerada. Além disso, os autores comprovam a garantia de convergência para um grupo restrito de problemas não convexos. No entanto, os hiperparâmetros do algoritmo FedAsync precisam ser ajustados para garantir a convergência em diferentes configurações. O algoritmo é incapaz de generalizar para se adequar às restrições de computação dinâmica de dispositivos heterogêneos. Dada a incerteza em torno da confiabilidade do aprendizado federado assíncrono, métodos síncronos continuam sendo mais utilizados atualmente [Lim et al., 2020].

Em todos os cenários considerados, na prática, os participantes podem relutar em participar de uma federação sem receber compensação, uma vez que os modelos de treinamento consomem recursos computacionais. Além disso, existe assimetria de informações entre o servidor e os participantes, uma vez que os participantes têm maior conhecimento dos recursos de computação disponíveis e da qualidade dos dados. Portanto, os mecanismos de incentivo devem ser cuidadosamente projetados para incentivar a participação e reduzir os potenciais impactos adversos da assimetria de informação.

Feng *et al.* propõem um esquema de precificação de serviço no qual os participantes atuam como provedores de serviços de treinamento para um proprietário de modelo, *i.e.*, o servidor do aprendizado federado [Feng et al., 2019]. Para superar a ineficiência energética na transferência de atualizações de modelos, uma rede de retransmissão cooperativa é proposta para apoiar a transferência e comercialização de atualizações de modelos. A interação entre os participantes e o proprietário do modelo é modelada como um jogo de Stackelberg [Osborne et al., 2004], no qual o proprietário do modelo é o comprador e os participantes são os vendedores. A proposta do jogo de Stackelberg é que cada participante pode decidir, não cooperativamente, sobre seu próprio preço de maximização de lucro. No sub-jogo de nível inferior, o proprietário do modelo determina o tamanho dos dados de treinamento para maximizar os lucros, considerando a relação côncava crescente entre a acurácia do modelo e o tamanho dos dados de treinamento. No sub-jogo de nível superior, os participantes decidem o preço por unidade de dados para maximizar seus lucros individuais. Os resultados da simulação mostraram que o mecanismo proposto pode garantir a unicidade do equilíbrio de Stackelberg. Atualizações de modelo que contêm informações valiosas têm preços mais elevados no equilíbrio de Stackelberg. De forma semelhante ao trabalho anterior, Sarikaya e Ercetin modelam a interação entre os participantes e o proprietário do modelo como um jogo Stackelberg, que é adequado para representar a interação servidor/participante dos envolvidos no treinamento [Sarikaya e Ercetin, 2020].

Ao contrário das abordagens anteriores, Zhan *et al.* adotam uma abordagem baseada em aprendizado por reforço profundo (*Deep Reinforcement Learning* – DRL) para resolver as formulações de Stackelberg [Zhan et al., 2020]. Na formulação do DRL, o servidor atua como um agente que decide um pagamento em resposta ao nível de participação e histórico de pagamento dos nós participantes, com o objetivo de minimizar

despesas com incentivos. Em seguida, os participantes determinam um nível de participação ideal em resposta à política de pagamento. Esse mecanismo de incentivo baseado em aprendizado por reforço permite que o servidor obtenha uma política ótima em resposta ao seu estado observado, sem necessitar nenhuma informação prévia.

Kang *et al.* apresentam a reputação como uma métrica para medir a confiabilidade dos participantes do aprendizado federado e projetam um esquema de seleção de participantes baseado em reputação [Kang et al., 2019b, Kang et al., 2019a]. Nesse cenário, cada participante tem um valor de reputação derivado de duas fontes, i) opiniões de reputação diretas de interações anteriores com o servidor e ii) opiniões de reputação indiretas de outros editores de tarefas,que são outros servidores de aprendizado federado de outras aplicações. As opiniões indiretas de reputação são armazenadas em uma cadeia de blocos aberta de reputação [Dennis e Owen, 2015] para garantir o gerenciamento seguro da reputação de uma maneira descentralizada. Antes do treinamento do modelo, os participantes escolhem um contrato que melhor se adapta à acurácia de seu conjunto de dados e às condições de seus recursos locais.

### 3.6.3. Heterogeneidade Estatística dos Dados

Em aprendizado de máquina distribuído, tradicionalmente, o servidor central tem acesso a todo o conjunto de dados de treinamento [Zhan et al., 2020]. Como tal, o servidor pode dividir o conjunto de dados em subconjuntos que seguem distribuições semelhantes. Os subconjuntos são posteriormente enviados aos nós participantes para treinamento distribuído. No entanto, essa abordagem é impraticável para aprendizado federado, uma vez que o conjunto de dados local só pode ser acessado pelo proprietário dos dados. Os desafios surgem ao treinar modelos federados utilizando dados que são altamente distribuídos de forma não idêntica entre os dispositivos, tanto em modelagem de dados heterogêneos quanto em análise comportamental da convergência dos procedimentos de treinamento.

Em aprendizado de máquina, existem trabalhos na literatura que visam modelar a heterogeneidade estatística utilizando métodos como meta-aprendizado e aprendizado multitarefa. O meta-aprendizado consiste de algoritmos de aprendizado automático aplicados a metadados [Vilalta e Drissi, 2002]. O aprendizado multitarefa é uma abordagem de transferência de aprendizado que melhora a generalização usando as informações contidas nos parâmetros de treinamento de tarefas relacionadas como um viés indutivo (*bias*) [Caruana, 1997]. Nesse contexto, uma tarefa pode ser minimizar a função de perda. Essas ideias foram recentemente estendidas ao ambiente federado [Li et al., 2020].

Smith *et al.* propõem uma estrutura de otimização projetada para o ambiente federado chamada MOCHA, que permite a personalização através do aprendizado de modelos separados, porém relacionados, para cada dispositivo, ao mesmo tempo em que potencializa uma representação compartilhada através do aprendizado multitarefa [Smith et al., 2017]. O MOCHA é calibrado com base nas restrições de recursos de um dispositivo participante como, por exemplo, condições da rede e estados da CPU dos dispositivos. No entanto, o MOCHA não pode ser aplicado a modelos de aprendizado profundo não-convexos [Smith et al., 2017].

Frequentemente, há relações internas entre a estrutura dos modelos locais dos participantes. Para capturar essas relações, o aprendizado multitarefa é uma estratégia

natural que melhora o desempenho e aumenta o tamanho da amostra eficaz para cada nó. Esse método tem garantias de convergência teórica comprováveis para os objetivos considerados, mas é limitado em sua capacidade de escalar para redes massivas e é restrito a objetivos convexos [Smith et al., 2017].

Eichner *et al.* investigam uma solução pluralista adaptativa, escolhendo entre um modelo global e modelos específicos de dispositivo, para abordar os padrões cíclicos em amostras de dados durante o treinamento federado [Eichner et al., 2019]. Outra abordagem visa modelar uma topologia em estrela como uma rede bayesiana e realiza inferência variacional durante o aprendizado [Corinzia e Buhmann, 2019]. A inferência variacional é uma abordagem que estima distribuições difíceis ou complexas *a posteriori*, i.e., a probabilidade de uma variável dado um evento. Embora esse método lide com funções não-convexas, é caro realizar generalização em redes federadas grandes. Apesar desses avanços recentes, os principais desafios ainda permanecem no desenvolvimento de métodos para modelagem heterogênea que sejam robustos, escaláveis e automatizados em configurações federadas.

Ao modelar dados no ambiente de aprendizado federado, é importante considerar questões além da acurácia. Em particular, resolver ingenuamente uma função de perda agregada, como a vista na Equação 2, pode implicar implicitamente obter vantagem e/ou desvantagem de alguns dos dispositivos, pois o modelo aprendido pode se tornar tendencioso para dispositivos com maiores quantidades de dados [Li et al., 2020]. Para garantir a convergência, Chiu *et al.* propõem o FedProx, que modifica a função de perda global para incluir também um parâmetro ajustável que restringe o quanto as atualizações locais afetam os parâmetros do modelo predominante [Chiu et al., 2020]. O algoritmo FedProx é ajustado de forma adaptativa. Quando a perda de treinamento está aumentando, as atualizações do modelo são ajustadas para afetar menos os parâmetros atuais. Da mesma forma, Huang *et al.* propõem o algoritmo LoAdaBoost FedAvg, no qual os participantes treinam o modelo em seus dados locais e comparam a perda da entropia cruzada com a perda mediana da rodada de treinamento anterior [Huang et al., 2020]. Se a perda da entropia cruzada atual for maior, o modelo é retreinado antes da agregação global para aumentar a eficiência do aprendizado.

## 3.7. Exemplo de Criação de uma Aplicação de Aprendizado Federado

A seguir é criado um cenário prático de aplicação de aprendizado federado através de simulação. O cenário é um dos mais utilizados para a avaliação de novos algoritmos e métodos em aprendizado federado [Brendan McMahan et al., 2017, Yang et al., 2019]. Na simulação, é utilizado o conjunto de dados clássico MNIST para classificação de imagem. A simulação descreve passo a passo da criação de um ambiente utilizando a linguagem Python [8]. O código-fonte está disponível no GitHub [9]. Em um cenário real, cada participante apresenta seus próprios conjuntos de dados local, isoladamente. É importante lembrar que o objetivo do aprendizado federado é enviar os parâmetros dos modelos locais para o servidor e nunca os dados dos participantes. Para simular os dados dos participantes, também chamados de clientes, são criados fragmentos do conjunto de dados principal.

[8]Disponível em: https://www.python.org/.

[9]Disponível em: https://github.com/helioncneto/FederatedLearning/blob/master/FederatedLearning_python.py.

Para essa simulação, são criados 10 fragmentos do conjunto de dados MNIST [10], um para cada participante. O conjunto de dados consiste em imagens contendo 70.000 dígitos escritos manualmente, em que cada classe é mantida em diretórios separados.

O aprendizado federado é mais adequado para modelos de aprendizado parametrizados como as redes neurais. Técnicas de aprendizado de máquina, como KNN ou similares, que simplesmente armazenam os dados de treinamento durante o aprendizado, não se beneficiam do aprendizado federado. Assim, a aplicação prática cria uma rede perceptron multicamadas (*Multilayer Perceptron* — MLP) com 3 camadas para a tarefa de classificação, utilizado a biblioteca Keras [11]. O primeiro passo é instalar os pacotes necessários para a simulação. A lista de pacotes esté no arquivo requirements.txt. Execute o comando:

```
$ pip install -r requirements.txt
```

e, então, são instaladas as seguintes bibliotecas: *Sci-kit learn*, TensorFlow e numpy. Inicialmente, são importadas as bibliotecas necessárias, como mostrado no Código Fonte 3.1.

**Código Fonte 3.1. Importação das bibliotecas necessárias para a execução da aplicação prática.**

```
1  import numpy as np
2  import random
3  import cv2
4  import os
5  import tensorflow as tf
6  from imutils import paths
7  from sklearn.preprocessing import LabelBinarizer
8  from sklearn.model_selection import train_test_split
9  from sklearn.metrics import accuracy_score
10 from tensorflow.keras.models import Sequential
11 from tensorflow.keras.layers import Activation
12 from tensorflow.keras.layers import Dense
13 from tensorflow.keras.optimizers import SGD
14 from tensorflow.keras import backend as K
```

Uma função para carregar os dados na memória é criada e são mantidos 30% dos dados para testar o modelo global treinado mais tarde. As imagens são lidas do disco rígido em escala cinza e posteriormente achatada para criar um vetor de característricas. A etapa de achatamento da imagem é utilizada, pois, é usada uma arquitetura de rede MLP e cada imagem deve ser passada para o modelo em forma de um vetor. Para obter o rótulo de classe de uma imagem, é utilizada a *string* do caminho, pois cada dígito está em um diretório separado. Outro pré-processamento realizado é o dimensionamento das imagens entre 0 e 1, para diminuir o impacto da variação do brilho do *pixel*. Depois disso,

[10] Disponível em http://yann.lecun.com/exdb/mnist/.
[11] Disponível em https://keras.io/. Acessado em 26/09/2020

**Código Fonte 3.2. Preprocessamento do conjunto de dados.**

```
def load_mnist_bypath(paths, verbose=-1):
    data = list()
    labels = list()
    # loop nas imagens de entrada
    for (i, imgpath) in enumerate(paths):
        # carrega a imagem e extrai os rótulos da classe
        im_gray = cv2.imread(imgpath, cv2.IMREAD_GRAYSCALE)
        image = np.array(im_gray).flatten()
        label = imgpath.split(os.path.sep)[-2]
        # aqui é feita a escala da img para [0, 1] para diminuir o
        # impacto do brilho de cada pixel
        data.append(image/255)
        labels.append(label)
    # retorna uma tupla dos dados e rótulos
    return data, labels

# Declara o caminho do conjunto de dados mnist
img_path = 'mnist/trainingSet/trainingSet'

# Gera a lista de caminhos, utilizando a funcao list_images da
# biblioteca paths
image_paths = list(paths.list_images(img_path))

# Carrega as imagens em arrays
image_list, label_list = load_mnist_bypath(image_paths, verbose=10000)

# Realiza o one-hot encoded para podermos utilizar a
# funcao de perda sparse-categorical-entropy
lb = LabelBinarizer()
label_list = lb.fit_transform(label_list)
# divide os dados em treinamento e conjunto de teste
X_train, X_test, y_train, y_test = train_test_split(image_list,
            label_list, test_size=0.3, random_state=42)
```

é utilizado o objeto `LabelBinarizer` da biblioteca *sci-kit learn* para realizar o *one-hot-encoding* nos rótulos. No *one-hot-encoding*, em vez de ter o rótulo do dígito 1 como número 1, ele terá a forma de um vetor como [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]. Com esse estilo de rotulagem, pode-se usar a função de perda de entropia cruzada dos modelos. Foi utilizado o objeto `train_test_split` da biblioteca *sci-kit learn* para dividir os dados em treino/teste. As etapas de preprocessamento são evidenciadas no Código Fonte 3.2.

O próximo passo é criar uma função que gere os dados locais dos clientes de forma non-IID, como mostrado no Código Fonte 3.3, utilizando o conjunto de dados de treinamento. É criada uma lista de nomes de clientes usando um prefixo que é passado como parâmetro da função. As listas de dados e seus respectivos rótulos são compactados

**Código Fonte 3.3. Função de geração dos dados locais dos clientes.**

```
def create_clients(image_list, label_list, num_clients=10,
                   initial='clients'):

    # cria a lista de nomes de clientes
    client_names = ['{}_{}'.format(initial, i + 1)
                    for i in range(num_clients)]

    # embaralha os dados
    data = list(zip(image_list, label_list))
    random.shuffle(data)

    # fragmenta os dados e divide para cada cliente
    size = len(data) # num_clients

    shards = [data[i:i + size]
            for i in range(0, size * num_clients, size)]

    # Verifica se o numero de fragmento é igual ao de clientes
    assert(len(shards) == len(client_names))

    return {client_names[i]: shards[i] for i in range(len(client_names))}

# Cria os clientes
clients = create_clients(X_train, y_train, num_clients=100,
                   initial='client')
```

e depois embaralhados. Por fim, essa função cria fragmentos desses dados baseado no número desejado de clientes, `num_clients`. Então, é retornado um dicionário contendo o nome de cada cliente como chave e seu conjunto de dados como valor. Essa função, então, é aplicada ao conjunto de dados de treinamento.

Em seguida, cada conjuntos de dados dos clientes é convertido para um conjunto de dados do TensorFlow, e, posteriormente, transformado em lote. Para simplificar essa etapa de conversão de dados e evitar a repetição, é realizado o encapsulamento desse procedimento em uma função chamada `batch_data`, como mostrado no Código Fonte 3.4. É importante lembrar que cada conjunto de dados dos clientes deve se tornar uma lista de tuplas (dados, rótulo) de `create_clients`. Nessa função, primeiro, é feita a separação da tupla em listas. Em seguida, é criado o conjunto de dados TensorFlow, realizado o embaralhamento dos dados e o agrupamento a partir dessas listas. Durante a aplicação da função, é separado o conjunto de teste e treinamento de cada cliente.

O Código Fonte 3.5 mostra a criação do modelo global MLP de 3 camadas que serve de modelo para iniciar a tarefa de classificação. Na simulação, são utilizados os módulos Keras que foram importados anteriormente. Para construir um novo modelo, é criada uma classe MLP e o seu construtor inicia o modelo. Como é usado o conjunto de

**Código Fonte 3.4. Função de encapsulamento para a adequação do conjunto de dados à biblioteca TensorFlow.**

```
1  def batch_data(data_shard, b=32):
2      # Separa cada fragmento em listas de dados e rótulo
3      data, label = zip(*data_shard)
4      dataset = tf.data.Dataset.from_tensor_slices((list(data),
5                  list(label)))
6      return dataset.shuffle(len(label)).batch(b)
7
8
9  # Processa e agrupa os dados de treinamento para cada cliente
10 clients_batched = dict()
11 for (client_name, data) in clients.items():
12     clients_batched[client_name] = batch_data(data)
13
14 # Processa e agrupa conjunto de teste
15 test_batched = tf.data.Dataset.from_tensor_slices((X_test, y_test))
16 .batch(len(y_test))
```

dados MNIST, o tamanho do parâmetro de entrada é $28 * 28 * 1 = 784$, pois cada imagem possui o formato 28x28 *pixels*, enquanto o número de classes é 10. Após, são definidos um otimizador, uma função de perda e métricas para avaliar o modelo posteriormente. É utilizado o otimizador SGD. A função de perda é a `categorical_crossentropy`.

Finalmente, a métrica utilizada para avaliação é a acurácia. A variável `comms_round` controla o número de agregações globais que são executadas durante o treinamento. Portanto, em vez de diminuir a taxa de aprendizado em relação ao número de épocas locais, é diminuído em relação ao número de agregações globais [Zhao et al., 2018]. Essa escolha de hiper-parâmetros depende da implementação.

Até esse ponto, foram apresentados os passos da implementação de uma aplicação de aprendizado profundo padrão, com exceção do particionamento de dados e da criação dos clientes. A partir de então, é implementado o algoritmo *FedAvg*, mostrado no Codigo Fonte 3.6, que é o ponto principal desta seção.

Os dados estão particionados horizontalmente [Yang et al., 2019]. Assim, é feita a média dos parâmetros dos componentes, que representa um peso de acordo a proporção de amostras de dados fornecidos por cada cliente participante. Essa parte é realizada de acordo com as Equações 2 e 1. Esse procedimento foi encapsulado em três funções:

1. *weight_scalling_factor* - Calcula a proporção dos dados de treinamento local de um cliente com os dados gerais de treinamento mantidos por todos os clientes. Primeiro, é obtido o tamanho do lote do cliente (linha 3), que é utilizado para calcular o número de pontos de dados (linha 5). Em seguida, é obtido o tamanho geral dos dados de treinamento globa (linha 9). Por fim, é calculado o fator de escala de um determinado cliente como uma fração (linha 11). Essa, certamente, não pode ser a abordagem em uma aplicação do mundo real, pois esse escalar deve ser calculado

pelo servidor. Nesse caso, cada cliente deverá indicar o número de pontos de dados com os quais treinou enquanto atualiza o servidor com novos parâmetros após cada etapa de treinamento local.

2. *scale_model_weights*: Dimensiona cada um dos pesos do modelo local com base no valor de seu fator de escala calculado em weight_scalling_factor.

3. *sum_scaled_weights*: Soma todos os pesos já escalados dos clientes.

São criadas duas funções, uma para avaliar o modelo global (`test_model`) e outra para verificar a perda dos modelos locais(`check_local_loss`), mostradas no Código Fonte 3.7. A lógica de treinamento tem dois laços principais, evidenciados no Código Fonte 3.8. O laços externo é para a iteração global, enquanto o interno é para iterar através do treinamento local de cada cliente. No entanto, há um terceiro laço implícito, que são as épocas locais, que é controlado pelo argumento *epochs* do método `model.fit`.

A simulação proposta começa com a construção do modelo global com 784 entradas e 10 classes. Em seguida, é feita a inicialização dos pesos do modelo global que consiste, neste exemplo, como uma sequência de zeros. Em seguida, é embaralhada a ordem do dicionário de clientes para garantir a aleatoriedade. A partir daí, começa a iteração por meio do treinamento local dos clientes. A quantidade de épocas locais é controlada pela variável `local_epochs`. Após o treinamento, os novos pesos são escalonados e

**Código Fonte 3.5. Criação da classe que implementa a rede neural *multilayer perceptron*.**

```
class MLP:
    @staticmethod
    def build(shape, classes):
        model = Sequential()
        model.add(Dense(200, input_shape=(shape,)))
        model.add(Activation("relu"))
        model.add(Dense(200))
        model.add(Activation("relu"))
        model.add(Dense(classes))
        model.add(Activation("softmax"))
        return model

lr = 0.01
comms_round = 100
loss = 'categorical_crossentropy'
metrics = ['accuracy']
optimizer = SGD(lr=lr, decay=lr / comms_round, momentum=0.9)
```

**Código Fonte 3.6. Funções de implementação do algoritmo de média federada (*Federated Averaging*).**

```
def weight_scalling_factor(clients_trn_data, client_name, participants):
    # calcula o tamanho do batch
    bs = list(clients_trn_data[client_name])[0][0].shape[0]
    # primeiro calcula o total de  dados de treinamento entre clientes
    global_count = sum([tf.data.experimental.cardinality
    (clients_trn_data[client_name]).numpy()
    for client_name in participants]) * bs
    # obtem o número total de pontos de dados mantidos por um cliente
    local_count = tf.data.experimental.cardinality
    (clients_trn_data[client_name]).numpy() * bs
    return local_count / global_count


def scale_model_weights(weight, scalar):
    '''Escala os pesos do modelo'''
    weight_final = []
    steps = len(weight)
    for i in range(steps):
        weight_final.append(scalar * weight[i])
    return weight_final


def sum_scaled_weights(scaled_weight_list):
    avg_grad = list()
    # obtem a média dos gradientes dos cliente
    for grad_list_tuple in zip(*scaled_weight_list):
        layer_mean = tf.math.reduce_sum(grad_list_tuple, axis=0)
        avg_grad.append(layer_mean)
    return avg_grad
```

anexados à lista `scaled_local_weight_list`. Essa etapa representa o treinamento local dos participantes. Voltando ao laço externo, é feita a soma de todos os pesos locais treinados e escalados e é realizada atualização do modelo global. Isso encerra uma época de treinamento global.

Os resultados mostram que, após 100 iterações de comunicação, o modelo global obtém 96,5% de acurácia. Existem diversos arcabouços de aprendizado federado que implementam o algoritmo FedAvg. O TensorFlow possui o módulo *TensorFlow Federated* (TFF) [12], que além de implementar o FedAvg, permite que o usuário crie seu próprio algoritmo customizado. Outro arcabouço que implementa o aprendizado federado é o PyTorch [13]. Existe também um arcabouço de conjunto de dados chamado LEAF, que contém conjuntos de dados particionados para aprendizado federado, facilitando assim a

[12]Disponível em https://www.tensorflow.org/federated. Acessado em 26/09/2020
[13]Disponível em https://pytorch.org/mobile/home/. Acessado em 26/09/2020

**Código Fonte 3.7. Funções para a medição da acurácia e perda do modelo e verificação local da perda.**

```
def test_model(X_test, Y_test, model, comm_round):
    cce = tf.keras.losses.CategoricalCrossentropy(from_logits=True)
    # logits = model.predict(X_test, batch_size=100)
    logits = model.predict(X_test)
    loss = cce(Y_test, logits)
    acc = accuracy_score(tf.argmax(logits, axis=1),
    tf.argmax(Y_test, axis=1))
    print('Rodada de Comunicação: {} | global_acc: {:.3\%}
    | global_loss: {}'.format(comm_round, acc, loss))
    return acc, loss


def check_local_loss(client, model):
    # Verificar perda local
    cce_l = tf.keras.losses.CategoricalCrossentropy
    (from_logits=True)
    client_x = np.array([i[0] for i in clients[client]])
    client_y = np.array([i[1] for i in clients[client]])
    logits_l = model.predict(client_x)
    loss_l = cce_l(client_y, logits_l)
    acc_l = accuracy_score(tf.argmax(logits_l, axis=1),
                    \ tf.argmax(client_y, axis=1))
    print('Local accuracy: {}. Local loss: {}'.format(acc_l, loss_l))
    return acc_l, loss_l
```

avaliação de modelos globais.

## 3.8. Considerações Finais e Perspectivas Futuras

O aprendizado federado consiste em uma área de pesquisa que está em ascendência em função do aumento da capacidade de processamento de dispositivos móveis e da necessidade de se garantir a privacidade de dados pessoais de usuários. A técnica de aprendizado federado se baseia em desenvolver um modelo de aprendizagem colaborativa em que participantes, muitas vezes nós móveis, executam parte da tarefa de aprendizado de maneira local e contribuem para um modelo global. Contudo, o desenvolvimento de modelos de aprendizado colaborativos apresentam diversos desafios, entre eles a heterogeneidade de dispositivos participantes e a heterogeneidade estatística do dados. Esse trabalho analisou as principais arquiteturas de referência de aprendizado federado e destacou as dificuldades que essas arquiteturas encontram ao tratar dados e dispositivos heterogêneos. Quanto a heterogeneidade de dispositivos, o trabalho destacou o compromisso entre arquiteturas síncronas, limitadas pelo dispositivo com menor capacidade sujeito a ser retardatário, e arquiteturas assíncronas, que permitem a entrega de resultadas mesmo após a rodada ao custo de uma menor participação no modelo global. Quanto a hetero-

**Código Fonte 3.8. Execução da aplicação de aprendizado federado implementada.**

```
smlp_global = MLP()
global_model = smlp_global.build(784, 10)

for comm_round in range(comms_round):

    # obtem os pesos do modelo global
    #servirá como os pesos iniciais para todos os modelos locais
    global_weights = global_model.get_weights()

    # Cria a lista para coletar os pesos do modelo local após o
    # escalonamento
    scaled_local_weight_list = list()
    client_names = list(clients_batched.keys())
    random.shuffle(client_names)
    client_select = client_names[0:55]

    for client in client_select:
        smlp_local = MLP()
        local_model = smlp_local.build(784, 10)
        local_model.compile(loss=loss, optimizer=optimizer,
        metrics=metrics)

        # Definir o peso do modelo local para o peso do modelo global
        local_model.set_weights(global_weights)
        local_model.fit(clients_batched[client], epochs=1, verbose=0)

        # Dimensiona os pesos do modelo e adicionar à lista
        scaling_factor = weight_scalling_factor
        (clients_batched, client, client_select)
        scaled_weights = scale_model_weights(local_model.get_weights(),
                \ scaling_factor)
        scaled_local_weight_list.append(scaled_weights)

        acc_l, loss_l = check_local_loss(client, local_model)

        K.clear_session()

    # Obtém a média de todo o modelo local, simplesmente
    # pegamos a soma dos pesos escalados
    average_weights = sum_scaled_weights(scaled_local_weight_list)

    # Atualização do modelo global
    global_model.set_weights(average_weights)
```

geneidade de dados, o trabalho destacou o impacto de dados que não são independentes e não são identicamente distribuídos entre clientes. Dados heterogêneos tendem a criar enviesamento nos modelos de aprendizado global.

O trabalho abordou ainda os principais ataques ao aprendizado federado. Ataques aos conjuntos de dados locais tendem a ter influência moderada sobre o modelo global agregado. Contudo, ataques Sybil podem facilmente comprometer o modelo global. Ademais, ataques ao modelo em si, como o envio de parâmetros locais desviados, tendem a ter forte influência sobre o modelo global agregado. Esse minicurso destacou as soluções propostas para os ataques elencados. As principais soluções se baseiam na criptografia homomórfica e na computação multiparte segura. Paralelamente, também foi elicitado o ataque de *free riding*, no qual o atacante egoísta se aproveita do modelo global agregado, porém pouco contribui com recursos locais para o cálculo do modelo. As soluções prevalentes na literatura para este ataque consistem em criar incentivos, como a participação dos clientes em cadeias de blocos, e modelos de reputação. O trabalho elencou ainda diversas aplicações em redes móveis e redes tradicionais que se utilizam do aprendizado federado, demonstrando que, embora recente, já há diversas aplicações que o utilizam. Por fim, foi demonstrada uma simulação de aprendizado federado desenvolvida em Python que se baseia no algoritmo da média federada (*Federated Averaging* – FedAvg), um dos principais algoritmos para o aprendizado federado.

Considerando as principais propostas voltadas para a pesquisa em aprendizado federado, é possível constatar que ainda há diversos desafios a serem superados para que as técnicas de aprendizado federado atinjam a maturidade tecnológica. No entanto, há grande potencial no aprendizado federado para a proteção da privacidade de dados pessoais em ambientes sensíveis, como aplicações móveis ou na federação para extração de conhecimento entre empresas distintas. O aprendizado federado está alinhado com as necessidades atuais de atendimento às leis de proteção de dados pessoais, na medida em que organizações que compartilham dados de usuários para extraírem conhecimentos tendem a ter suas ações mais restritas. Contudo, ao aplicar o aprendizado federado é possível compartilhar modelos de aprendizado sem expor os dados pessoais de usuários e, ainda assim, obter a extração de conhecimento com alta acurácia e baixa perda.

## Referências

[Andreoni Lopez et al., 2019] Andreoni Lopez, M., Mattos, D. M., Duarte, O. C. M. e Pujolle, G. (2019). Toward a monitoring and threat detection system based on stream processing as a virtual network function for big data. *Concurrency and Computation: Practice and Experience*, 31(20):e5344.

[Asuncion e Newman, 2007] Asuncion, A. e Newman, D. (2007). Uci machine learning repository.

[Bagdasaryan et al., 2020] Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D. e Shmatikov, V. (2020). How to backdoor federated learning. Em *Proceedings of Machine Learning Research*, volume 108, p. 2938–2948, Online. PMLR.

[Baldi, 2011] Baldi, P. (2011). Autoencoders, unsupervised learning and deep architectures. Em *Proceedings of the 2011 International Conference on Unsupervised and*

*Transfer Learning Workshop - Volume 27*, UTLW'11, p. 37–50. JMLR.org.

[Bhagoji et al., 2019] Bhagoji, A. N., Chakraborty, S., Mittal, P. e Calo, S. (2019). Analyzing federated learning through an adversarial lens. Em *International Conference on Machine Learning*, p. 634–643.

[Bogetoft et al., 2009] Bogetoft, P., Christensen, D. L. e Damg, I. (2009). Multiparty Computation Goes Live. *Review Literature And Arts Of The Americas*, p. 1–13.

[Bonawitz et al., 2019] Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečnỳ, J., Mazzocchi, S., McMahan, H. B. et al. (2019). Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*.

[Bonawitz et al., 2017] Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A. e Seth, K. (2017). Practical secure aggregation for privacy-preserving machine learning. Em *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, p. 1175–1191, New York, NY, USA. Association for Computing Machinery.

[Bottou, 2010] Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. Em *Proceedings of COMPSTAT'2010*, p. 177–186, Heidelberg. Physica-Verlag HD.

[Boutaba et al., 2018] Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F. e Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):16.

[Bradley et al., 2000] Bradley, P. S., Bennett, K. P. e Demiriz, A. (2000). Constrained k-means clustering. *Microsoft Research, Redmond*, 20(0):0.

[Brasil, 2018] Brasil (2018). Lei nº 13.709, de 14 de agosto de 2018. Institui a Lei Geral de Proteção de Dados Pessoais (LGPD).

[Brendan McMahan et al., 2017] Brendan McMahan, H., Moore, E., Ramage, D., Hampson, S. e Agüera y Arcas, B. (2017). Communication-efficient learning of deep networks from decentralized data. Em *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017*, volume 54.

[Brisimi et al., 2018] Brisimi, T. S., Chen, R., Mela, T., Olshevsky, A., Paschalidis, I. C. e Shi, W. (2018). Federated learning of predictive models from federated electronic health records. *International Journal of Medical Informatics*, 112:59 – 67.

[Caldas et al., 2018] Caldas, S., Konečny, J., McMahan, H. B. e Talwalkar, A. (2018). Expanding the reach of federated learning by reducing client resource requirements. *arXiv preprint arXiv:1812.07210*.

[Caruana, 1997] Caruana, R. (1997). Multitask learning. *Machine learning*, 28(1):41–75.

[Chen et al., 2020] Chen, M., Semiari, O., Saad, W., Liu, X. e Yin, C. (2020). Federated echo state learning for minimizing breaks in presence in wireless virtual reality networks. *IEEE Transactions on Wireless Communications*, 19(1):177–191.

[Chiu et al., 2020] Chiu, T., Shih, Y., Pang, A., Wang, C., Weng, W. e Chou, C. (2020). Semi-supervised distributed learning with non-iid data for aiot service platform. *IEEE Internet of Things Journal*, p. 1–1.

[Chung et al., 2010] Chung, J., Yoon, H.-J. e Gardner, H. J. (2010). Analysis of break in presence during game play using a linear mixed model. *ETRI journal*, 32(5):687–694.

[Corinzia e Buhmann, 2019] Corinzia, L. e Buhmann, J. M. (2019). Variational federated multi-task learning. *arXiv preprint arXiv:1906.06268*.

[Cup, 1999] Cup, K. (1999). Available on: http://kdd. ics. uci. edu/databases/kddcup99/kddcup99. html.

[De Haan e Ferreira, 2007] De Haan, L. e Ferreira, A. (2007). *Extreme value theory: an introduction*. Springer Science & Business Media.

[de Oliveira et al., 2020a] de Oliveira, N. R., Lúcio Reis, H. A., Fernandes, N. C., Carlos Bastos, A. M., Dianne Medeiros, S. V. e Diogo Mattos, M. F. (2020a). Natural language processing characterization of recurring calls in public security services. Em *2020 International Conference on Computing, Networking and Communications (ICNC)*, p. 1009–1013.

[de Oliveira et al., 2020b] de Oliveira, N. R., Medeiros, D. S. V. e Mattos, D. M. F. (2020b). A sensitive stylistic approach to identify fake news on social networking. *IEEE Signal Processing Letters*, 27:1250–1254.

[Dean et al., 2012] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K. et al. (2012). Large scale distributed deep networks. Em *Advances in neural information processing systems*, p. 1223–1231.

[Dennis e Owen, 2015] Dennis, R. e Owen, G. (2015). Rep on the block: A next generation reputation system based on the blockchain. Em *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, p. 131–138.

[Du et al., 2004] Du, W., Han, Y. S. e Chen, S. (2004). Privacy-preserving multivariate statistical analysis: Linear regression and classification. Em *Proceedings of the 2004 SIAM international conference on data mining*, p. 222–233. SIAM.

[Du e Zhan, 2002] Du, W. e Zhan, Z. (2002). Building decision tree classifier on private data. *Syracuse University - Electrical Engineering and Computer Science*.

[Eichner et al., 2019] Eichner, H., Koren, T., McMahan, H. B., Srebro, N. e Talwar, K. (2019). Semi-cyclic stochastic gradient descent. *arXiv preprint arXiv:1904.10120*.

[Feng et al., 2019] Feng, S., Niyato, D., Wang, P., Kim, D. I. e Liang, Y. (2019). Joint service pricing and cooperative relay communication for federated learning. Em *2019 International Conference 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)*, p. 815–820.

[Fredrikson et al., 2015] Fredrikson, M., Jha, S. e Ristenpart, T. (2015). Model inversion attacks that exploit confidence information and basic countermeasures. Em *CCS '15*, p. 1322–1333, New York, NY, USA. Association for Computing Machinery.

[Fredrikson et al., 2014] Fredrikson, M., Lantz, E., Jha, S., Lin, S., Page, D. e Ristenpart, T. (2014). Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. Em *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, p. 17–32.

[Fung et al., 2018] Fung, C., Yoon, C. J. e Beschastnikh, I. (2018). Mitigating sybils in federated learning poisoning. *arXiv preprint arXiv:1808.04866*.

[Goldreich et al., 2019] Goldreich, O., Micali, S. e Wigderson, A. (2019). *How to Play Any Mental Game, or a Completeness Theorem for Protocols with Honest Majority*, p. 307–328. Association for Computing Machinery, New York, NY, USA.

[Gupta e Jha, 2015] Gupta, A. e Jha, R. K. (2015). A survey of 5g network: Architecture and emerging technologies. *IEEE Access*, 3:1206–1232.

[Han et al., 2015] Han, S., Mao, H. e Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

[Hard et al., 2018] Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C. e Ramage, D. (2018). Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*.

[Hardy et al., 2017] Hardy, S., Henecka, W., Ivey-Law, H., Nock, R., Patrini, G., Smith, G. e Thorne, B. (2017). Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677*.

[He et al., 2016] He, K., Zhang, X., Ren, S. e Sun, J. (2016). Deep residual learning for image recognition. Em *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[He et al., 2018] He, Y., Zhao, N. e Yin, H. (2018). Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach. *IEEE Transactions on Vehicular Technology*, 67(1):44–55.

[Ho et al., 2013] Ho, Q., Cipar, J., Cui, H., Lee, S., Kim, J. K., Gibbons, P. B., Gibson, G. A., Ganger, G. e Xing, E. P. (2013). More effective distributed ml via a stale synchronous parallel parameter server. Em *Advances in Neural Information Processing Systems 26*, p. 1223–1231. Curran Associates, Inc.

[Huang et al., 2020] Huang, L., Yin, Y., Fu, Z., Zhang, S., Deng, H. e Liu, D. (2020). Loadaboost: Loss-based adaboost federated machine learning with reduced computational complexity on iid and non-iid intensive care data. *Plos one*, 15(4):e0230706.

[Kang et al., 2019a] Kang, J., Xiong, Z., Niyato, D., Xie, S. e Zhang, J. (2019a). Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory. *IEEE Internet of Things Journal*, 6(6):10700–10714.

[Kang et al., 2019b] Kang, J., Xiong, Z., Niyato, D., Yu, H., Liang, Y. e Kim, D. I. (2019b). Incentive design for efficient federated learning in mobile networks: A contract theory approach. Em *2019 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS)*, p. 1–5.

[Karr et al., 2009] Karr, A. F., Lin, X., Sanil, A. P. e Reiter, J. P. (2009). Privacy-preserving analysis of vertically partitioned data using secure matrix products. *Journal of Official Statistics*, 25(1):125.

[Kim et al., 2018] Kim, H., Park, J., Bennis, M. e Kim, S.-L. (2018). On-device federated learning via blockchain and its latency analysis. *arXiv preprint arXiv:1808.03949*.

[Konečnỳ et al., 2016] Konečnỳ, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T. e Bacon, D. (2016). Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*.

[Kwon et al., 2017] Kwon, D., Kim, H., Kim, J., Suh, S. C., Kim, I. e Kim, K. J. (2017). A survey of deep learning-based network anomaly detection. *Cluster Computing*.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y. e Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Li et al., 2017] Li, P., Li, J., Huang, Z., Li, T., Gao, C.-Z., Yiu, S.-M. e Chen, K. (2017). Multi-key privacy-preserving deep learning in cloud computing. *Future Generation Computer Systems*, 74:76 – 85.

[Li et al., 2020] Li, T., Sahu, A. K., Talwalkar, A. e Smith, V. (2020). Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60.

[Li et al., 2018] Li, W., Logenthiran, T., Phan, V. e Woo, W. L. (2018). Implemented iot-based self-learning home management system (shms) for singapore. *IEEE Internet of Things Journal*, 5(3):2212–2219.

[Li et al., 2019] Li, W., Milletarì, F., Xu, D., Rieke, N., Hancox, J., Zhu, W., Baust, M., Cheng, Y., Ourselin, S., Cardoso, M. J. e Feng, A. (2019). Privacy-preserving federated brain tumour segmentation. Em *Machine Learning in Medical Imaging*, p. 133–141, Cham. Springer International Publishing.

[Liang e Chawathe, 2004] Liang, G. e Chawathe, S. S. (2004). Privacy-preserving inter-database operations. Em *International Conference on Intelligence and Security Informatics*, p. 66–82. Springer.

[Lim et al., 2020] Lim, W. Y. B., Luong, N. C., Hoang, D. T., Jiao, Y., Liang, Y. C., Yang, Q., Niyato, D. e Miao, C. (2020). Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys Tutorials*.

[Liu et al., 2019] Liu, Y., Kang, Y., Zhang, X., Li, L., Cheng, Y., Chen, T., Hong, M. e Yang, Q. (2019). A communication efficient vertical federated learning framework. *arXiv preprint arXiv:1912.11187*.

[Lobato et al., 2018] Lobato, A. G. P., Lopez, M. A., Sanz, I. J., Cardenas, A. A., Duarte, O. C. M. B. e Pujolle, G. (2018). An adaptive real-time architecture for zero-day threat detection. Em *2018 IEEE International Conference on Communications (ICC)*, p. 1–6.

[Long et al., 2015] Long, M., Cao, Y., Wang, J. e Jordan, M. (2015). Learning transferable features with deep adaptation networks. Em *International conference on machine learning*, p. 97–105. PMLR.

[Mattos et al., 2019] Mattos, D. M. F., Velloso, P. B. e Duarte, O. C. M. B. (2019). An agile and effective network function virtualization infrastructure for the internet of things. *Journal of Internet Services and Applications*, 10(1):6.

[Medeiros et al., 2019] Medeiros, D. S. V., Cunha Neto, H. N., Andreoni Lopez, M., Magalhães, L. C. S., Silva, E. F., Vieira, A. B., Fernandes, N. C. e Mattos, D. M. F. (2019). Análise de dados em redes sem fio de grande porte: Processamento em fluxo em tempo real, tendências e desafios. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC*, 2019:142–195.

[Micali et al., 1987] Micali, S., Goldreich, O. e Wigderson, A. (1987). How to play any mental game. Em *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*, p. 218–229.

[Mikolov et al., 2011] Mikolov, T., Kombrink, S., Burget, L., Černockỳ, J. e Khudanpur, S. (2011). Extensions of recurrent neural network language model. Em *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, p. 5528–5531. IEEE.

[Neely, 2010] Neely, M. J. (2010). Stochastic network optimization with application to communication and queueing systems. *Synthesis Lectures on Communication Networks*, 3(1):1–211.

[Nguyen et al., 2018] Nguyen, K. K., Hoang, D. T., Niyato, D., Wang, P., Nguyen, D. e Dutkiewicz, E. (2018). Cyberattack detection in mobile cloud computing: A deep learning approach. Em *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, p. 1–6.

[Nguyen et al., 2019] Nguyen, T. D., Marchal, S., Miettinen, M., Fereidooni, H., Asokan, N. e Sadeghi, A. (2019). DÏot: A federated self-learning anomaly detection system for iot. Em *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, p. 756–767.

[Nishio e Yonetani, 2019] Nishio, T. e Yonetani, R. (2019). Client selection for federated learning with heterogeneous resources in mobile edge. Em *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, p. 1–7.

[Nock et al., 2018] Nock, R., Hardy, S., Henecka, W., Ivey-Law, H., Patrini, G., Smith, G. e Thorne, B. (2018). Entity resolution and federated learning get a federated resolution. *arXiv preprint arXiv:1803.04035*.

[Osborne et al., 2004] Osborne, M. J. et al. (2004). *An introduction to game theory*, volume 3. Oxford university press New York.

[Pan e Yang, 2010] Pan, S. J. e Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

[Parlamento Europeu e Conselho da União Européia, 2016] Parlamento Europeu e Conselho da União Européia (2016). Regulamento (ue) 2016/679. `https://eur-lex.europa.eu/legal-content/PT/TXT/PDF/?uri=CELEX:32016R0679&from=PT`.

[Phong et al., 2018] Phong, L. T., Aono, Y., Hayashi, T., Wang, L. e Moriai, S. (2018). Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345.

[Preuveneers et al., 2018] Preuveneers, D., Rimmer, V., Tsingenopoulos, I., Spooren, J., Joosen, W. e Ilie-Zudor, E. (2018). Chained anomaly detection models for federated learning: An intrusion detection case study. *Applied Sciences*, 8(12):2663.

[Reis et al., 2020] Reis, L. H. A., Murillo Piedrahita, A., Rueda, S., Fernandes, N. C., Medeiros, D. S. V., de Amorim, M. D. e Mattos, D. M. F. (2020). Unsupervised and incremental learning orchestration for cyber-physical security. *Transactions on Emerging Telecommunications Technologies*, 31(7):e4011.

[Ren et al., 2019] Ren, J., Wang, H., Hou, T., Zheng, S. e Tang, C. (2019). Federated learning-based computation offloading optimization in edge computing-supported internet of things. *IEEE Access*, 7:69194–69201.

[Sadeghi et al., 2018] Sadeghi, A., Sheikholeslami, F. e Giannakis, G. B. (2018). Optimal and scalable caching for 5g using reinforcement learning of space-time popularities. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):180–190.

[Samarakoon et al., 2018] Samarakoon, S., Bennis, M., Saad, W. e Debbah, M. (2018). Federated learning for ultra-reliable low-latency v2v communications. Em *2018 IEEE Global Communications Conference (GLOBECOM)*, p. 1–7.

[Saputra et al., 2019] Saputra, Y. M., Hoang, D. T., Nguyen, D. N., Dutkiewicz, E., Mueck, M. D. e Srikanteswara, S. (2019). Energy demand prediction with federated learning for electric vehicle networks. Em *2019 IEEE Global Communications Conference (GLOBECOM)*, p. 1–6.

[Sarikaya e Ercetin, 2020] Sarikaya, Y. e Ercetin, O. (2020). Motivating workers in federated learning: A stackelberg game perspective. *IEEE Networking Letters*, 2(1):23–27.

[Scannapieco et al., 2007] Scannapieco, M., Figotin, I., Bertino, E. e Elmagarmid, A. K. (2007). Privacy preserving schema and data matching. Em *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, p. 653–664.

[Sheth e Larson, 1990] Sheth, A. P. e Larson, J. A. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.*, 22(3):183–236.

[Smith et al., 2017] Smith, V., Chiang, C.-K., Sanjabi, M. e Talwalkar, A. S. (2017). Federated multi-task learning. Em *Advances in Neural Information Processing Systems*, p. 4424–4434.

[Sprague et al., 2019] Sprague, M. R., Jalalirad, A., Scavuzzo, M., Capota, C., Neun, M., Do, L. e Kopp, M. (2019). Asynchronous federated learning for geospatial applications. Em *ECML PKDD 2018 Workshops*, p. 21–28. Springer International Publishing.

[Strom, 2015] Strom, N. (2015). Scalable distributed dnn training using commodity gpu cloud computing. Em *Sixteenth Annual Conference of the International Speech Communication Association*.

[Suresh et al., 2017] Suresh, A. T., Felix, X. Y., Kumar, S. e McMahan, H. B. (2017). Distributed mean estimation with limited communication. Em *International Conference on Machine Learning*, p. 3329–3337.

[Tao e Li, 2018] Tao, Z. e Li, Q. (2018). esgd: Communication efficient distributed deep learning on the edge. Em {*USENIX*} *Workshop on Hot Topics in Edge Computing (HotEdge 18)*.

[Tesauro, 2007] Tesauro, G. (2007). Reinforcement learning in autonomic computing: A manifesto and case studies. *IEEE Internet Computing*, 11(1):22–30.

[Triastcyn e Faltings, 2020] Triastcyn, A. e Faltings, B. (2020). Federated generative privacy. *IEEE Intelligent Systems*, 35(4):50–57.

[Trigeorgis et al., 2016] Trigeorgis, G., Ringeval, F., Brueckner, R., Marchi, E., Nicolaou, M. A., Schuller, B. e Zafeiriou, S. (2016). Adieu features? end-to-end speech emotion recognition using a deep convolutional recurrent network. Em *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, p. 5200–5204.

[Vaidya e Clifton, 2002] Vaidya, J. e Clifton, C. (2002). Privacy preserving association rule mining in vertically partitioned data. Em *KDD '02*, p. 639–644, New York, NY, USA. Association for Computing Machinery.

[Vaidya e Clifton, 2003] Vaidya, J. e Clifton, C. (2003). Privacy-preserving k-means clustering over vertically partitioned data. Em *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, p. 206–215, New York, NY, USA. Association for Computing Machinery.

[Vaidya e Clifton, 2004] Vaidya, J. e Clifton, C. (2004). Privacy preserving naive bayes classifier for vertically partitioned data. Em *Proceedings of the 2004 SIAM international conference on data mining*, p. 522–526. SIAM.

[Vaidya et al., 2008] Vaidya, J., Clifton, C., Kantarcioglu, M. e Patterson, A. S. (2008). Privacy-preserving decision trees over vertically partitioned data. *ACM Trans. Knowl. Discov. Data*, 2(3).

[Verma et al., 2018] Verma, D., Julier, S. e Cirincione, G. (2018). Federated ai for building ai solutions across multiple agencies. *arXiv preprint arXiv:1809.10036*.

[Vilalta e Drissi, 2002] Vilalta, R. e Drissi, Y. (2002). A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95.

[Wan et al., 2007] Wan, L., Ng, W. K., Han, S. e Lee, V. C. (2007). Privacy-preservation for gradient descent methods. Em *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, p. 775–783.

[Wang et al., 2019] Wang, L., Wang, W. e Li, B. (2019). Cmfl: Mitigating communication overhead for federated learning. Em *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, p. 954–964.

[Wang et al., 2019] Wang, S., Tuor, T., Salonidis, T., Leung, K. K., Makaya, C., He, T. e Chan, K. (2019). Adaptive Federated Learning in Resource Constrained Edge Computing Systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221.

[Wang et al., 2020] Wang, X., Han, Y., Leung, V. C. M., Niyato, D., Yan, X. e Chen, X. (2020). Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys Tutorials*, 22(2):869–904.

[Wang et al., 2019a] Wang, X., Han, Y., Wang, C., Zhao, Q., Chen, X. e Chen, M. (2019a). In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Network*, 33(5):156–165.

[Wang et al., 2019b] Wang, X., Han, Y., Wang, C., Zhao, Q., Chen, X. e Chen, M. (2019b). In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Network*, 33(5):156–165.

[Weng et al., 2019] Weng, J., Weng, J., Zhang, J., Li, M., Zhang, Y. e Luo, W. (2019). Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive. *IEEE Transactions on Dependable and Secure Computing*, p. 1–1.

[Wenliang Du e Atallah, 2001] Wenliang Du e Atallah, M. J. (2001). Privacy-preserving cooperative statistical analysis. Em *Seventeenth Annual Computer Security Applications Conference*, p. 102–110.

[Xie et al., 2019] Xie, C., Koyejo, S. e Gupta, I. (2019). Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*.

[Yang et al., 2019] Yang, Q., Liu, Y., Chen, T. e Tong, Y. (2019). Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology*, 10(2):1–19.

[Yao, 1982] Yao, A. C. (1982). Protocols for secure computations. Em *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, p. 160–164.

[Yao et al., 2018] Yao, X., Huang, C. e Sun, L. (2018). Two-stream federated learning: Reduce the communication costs. Em *2018 IEEE Visual Communications and Image Processing (VCIP)*, p. 1–4.

[You e Yang, 2014] You, P. e Yang, Z. (2014). Efficient optimal scheduling of charging station with multiple electric vehicles via v2v. Em *2014 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, p. 716–721.

[Yu et al., 2006a] Yu, H., Jiang, X. e Vaidya, J. (2006a). Privacy-preserving svm using nonlinear kernels on horizontally partitioned data. Em *Proceedings of the 2006 ACM Symposium on Applied Computing*, SAC '06, p. 603–610, New York, NY, USA. Association for Computing Machinery.

[Yu et al., 2006b] Yu, H., Vaidya, J. e Jiang, X. (2006b). Privacy-preserving svm classification on vertically partitioned data. Em *Advances in Knowledge Discovery and Data Mining*, p. 647–656, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Zhan et al., 2020] Zhan, Y., Li, P., Qu, Z., Zeng, D. e Guo, S. (2020). A learning-based incentive mechanism for federated learning. *IEEE Internet of Things Journal*, 7(7):6360–6368.

[Zhang et al., 2019] Zhang, J., Chen, B., Yu, S. e Deng, H. (2019). Pefl: A privacy-enhanced federated learning scheme for big data analytics. Em *2019 IEEE Global Communications Conference (GLOBECOM)*, p. 1–6.

[Zhang e Xiao, 2018] Zhang, Y. e Xiao, L. (2018). *Communication-Efficient Distributed Optimization of Self-concordant Empirical Loss*, p. 289–341. Springer International Publishing, Cham.

[Zhao et al., 2018] Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D. e Chandra, V. (2018). Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*.