Universidade Federal de São Carlos – UFSCar
Centro de Ciências Exatas e de Tecnologia – CCET
Departamento de Computação – DC
Programa de Pós-Graduação em Ciência da Computação – PPGCC

Luís Henrique Puhl de Souza

Uma Implementação distribuída em Névoa do algoritmo de Detecção de Novidade em Fluxos de Dados MINAS

São Carlos
2021

Luís Henrique Puhl de Souza

Uma Implementação distribuída em Névoa do algoritmo de Detecção de Novidade em Fluxos de Dados MINAS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências Exatas e de Tecnologia da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Sistemas de Computação

Orientador: Prof. Dr. Hermes Senger

São Carlos
2021

Puhl, Luis

Uma Implementação distribuída em Névoa do algoritmo de Detecção de Novidade em Fluxos de Dados MINAS / Luis Puhl -- 2021.
65f.

Dissertação (Mestrado) - Universidade Federal de São Carlos, campus São Carlos, São Carlos
Orientador (a): Hermes Senger
Banca Examinadora: Helio Guardia
Bibliografia

1. Detecção de Intrusão. 2. Internet das Coisas. I. Puhl, Luis. II. Título.

Ficha catalográfica desenvolvida pela Secretaria Geral de Informática (SIn)

DADOS FORNECIDOS PELO AUTOR

Bibliotecário responsável: Ronildo Santos Prado - CRB/8 7325

Universidade Federal de São Carlos – UFSCar
Centro de Ciências Exatas e de Tecnologia – CCET
Departamento de Computação – DC
Programa de Pós-Graduação em Ciência da Computação – PPGCC

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato Luís Henrique Puhl de Souza, realizada em 20 de abril de 2021:

Prof. Dr. Hermes Senger
Orientador

Professor
Convidado 1

Professor
Convidado 2

São Carlos
2021

Agradecimentos

Agradecimentos à Roberto Santos Inoue e Tatiane Cristina da Costa Fernandes da Universidade Federal de São Carlos – UFSCar que adaptaram a classe desenvolvida por Athila Quaresma Santos e Renato Machado Monaro para o Programa em Engenharia Elétrica da Escola de Engenharia de São Carlos.

Também agradecemos à Athila Quaresma Santos e Renato Machado Monaro pela adaptação da classe abnTEX2 , bem como à todos aqueles que contribuíram para que a produção de trabalhos acadêmicos conforme as normas ABNT com LATEX fosse possível.

Parte do desenvolvimento deste modelo, especialmente os elementos pré-textuais, foi inspirado no modelo desenvolvido por Samir e modificado por Marcelo Suetake.

Agradecimentos especiais são direcionados aos voluntários do grupo de usuários *latex-br*[1] e aos novos voluntários do grupo *abnTEX2*[2] que contribuíram e que ainda contribuirão para a evolução do abnTEX2.

[1] <http://groups.google.com/group/latex-br>
[2] <http://groups.google.com/group/abntex2> e <http://abntex2.googlecode.com/>

Resumo

Em um cenário de crescente número de dispositivos na Internet das Coisas (IoT), gerando proporcional crescimento no volume dos fluxos de dados gerados, são necessários métodos robustos para a mineração de fluxos contínuos de dados. Uma das áreas afetadas pelo crescimento vertiginoso do número de dispositivos e os fluxos associados a eles é a área de segurança da informação, onde são necessárias ferramentas de detecção de intrusão em redes que operem em ambientes de computação em névoa, devido aos custos de comunicação associados a operar estas ferramentas somente em ambiente de nuvem. As ferramentas de detecção de intrusão utilizam extensivamente algoritmos de detecção de novidade em fluxos de dados para identificar padrões no tráfego da rede. Porém, os algoritmos que tratam adequadamente dos desafios de detecção de novidade em fluxos de dados, como mudança e evolução de conceito e atualização contínua do modelo de classificação sem interferência de especialistas, ainda são pouco utilizados. O algoritmo de detecção de novidade em fluxo de dados MINAS tem recebido atenção de pesquisas recentes por tratar desses desafios de detecção de novidade em fluxos de dados. No entanto, apesar de sua divisão em três partes semi-independentes, este algoritmo ainda não foi adaptado para processar grandes volumes de fluxos reais em ambiente de computação em névoa. O presente trabalho aborda essa lacuna, propondo um sistema que implementa o algoritmo MINAS de maneira distribuída num contexto de detecção de intrusão e computação em névoa. Experimentos mostram que o algoritmo MINAS pode ser paralelizado e distribuído utilizando plataformas de processamento de fluxos como *Apache Flink*.

Palavras-chave: Detecção de Novidades. Detecção de Intrusão. Fluxos de Dados. Computação Distribuída. Computação em Névoa. Internet das Coisas.

Abstract

In a scenario of growing number of devices connected to the Internet of Things (IoT) with proportional growth in the volume of data streams generated, robust methods are needed for mining streams continuous data. One of the areas affected by the huge growth in the number of devices and the streams associated with them is the information security, which needs network intrusion detection tools that operate in fog computing environments due to the cost of operating such tools in a cloud only environment. These tools make extensive use of algorithms for novelty detection in data streams to identify treat patterns in network traffic. However, algorithms in wide use do not adequately address the challenges of novelty detection in data streams, such as concept drift, concept evolution and continuous update of the classification model, without expert interference. The MINAS algorithm addresses those novelty detection in data streams challenges and has received recent research attention. However, despite its division in three semi-independent parts, MINAS has not yet been adapted to process large volumes of real streams or to operate in a fog computing environment. The present work proposes a system that implements the MINAS algorithm in a distributed fog environment in the context of intrusion detection to addresses this gap. Preliminary work shows that it is possible to have a distributed version of the MINAS algorithm by using stream processing platforms such as Apache Flink.

The ongoing implementation of the Internet of Things (IoT) is sharply increasing the number and variety of small devices on edge networks. Likewise, the attack opportunities for hostile agents also increases, requiring more effort from network administrators and strategies to detect and react to those threats. For a network security system to operate in the context of edge and IoT, it has to comply with processing, storage, and energy requirements alongside traditional requirements for stream and network analysis like accuracy and scalability. Using a previously defined architecture (IDSA-IoT), we address the construction and evaluation of a support mechanism for distributed Network Intrusion Detection Systems (NIDS) based on the MINAS Data Stream Novelty Detection (DSND)

algorithm. We discuss the algorithm steps, how it can be deployed in a distributed environment, the impacts on the accuracy and evaluate performance and scalability using a cluster of constrained devices commonly found in IoT scenarios. The obtained results show a negligible accuracy loss in the distributed version but also a small reduction in the execution time using low profile devices. Although not efficient, the parallel version showed to be viable as the proposed granularity provides equivalent accuracy and viable response times.

Keywords: Novelty Detection. Intrusion Detection. Data Streams. Distributed Computing. Fog Computing. IoT devices.

Lista de ilustrações

Lista de tabelas

Lista de siglas

Sumário

Capítulo 1

Introdução

A Internet das Coisas (*Internet of Things* - IoT) é um sistema global de dispositivos (máquinas, objetos físicos ou virtuais, sensores, atuadores e pessoas) com capacidade de comunicação pela Internet, sem depender de interação com interface humano-computador tradicional. Outra característica de dispositivos IoT são os recursos computacionais dimensionados, para propósitos específicos que limitam a capacidade de computar outras funções além da função original do dispositivo. O número de dispositivos categorizados como IoT na última década teve crescimento sem precedentes e, proporcionalmente, cresceu o volume de dados gerados por esses dispositivos. A análise desses dados pode trazer novos conhecimentos e tem sido um tema frequentemente abordado por trabalhos de pesquisa. Contudo, além dos dados de sensores e atuadores, esses dispositivos se subvertidos, podem gerar tráfego maligno , como o gerado pela *botnet* mirai em 2016 (KAMBOURAKIS; KOLIAS; STAVROU, 2017). Nesse cenário, fatores que podem favorecer a subversão dos dispositivos incluem a falta de controle sobre a origem do hardware e software embarcado nos dispositivos, além da falta das cruciais atualizações de segurança.

Com milhares de dispositivos em redes distantes gerando dados (diretamente ligados às suas funções originais ou metadados produzidos como subproduto) em volumes e velocidades consideráveis, formando fluxos contínuos de dados (*Data Stream* - DS), técnicas de mineração de fluxos de dados (*Data Stream Mining*) são amplamente necessárias. Nesses cenários, essas técnicas são aplicadas, por exemplo, em problemas de monitoramento e classificação de valores originários de sensores para tomada de decisão tanto em nível micro, como na modificação de atuadores remotos, ou macro, na otimização de processos industriais. Analogamente, as mesmas técnicas de classificação podem ser aplicadas para os metadados gerados pela comunicação entre esses nós e a Internet, detectando alterações nos padrões de comunicação num serviço de detecção de intrusão (*Network Intrusion*

Detection System, NIDS).

Técnicas de Mineração de Fluxo de Dados (*Data Stream Mining*) envolvem mineração de dados (*Data Mining*), aprendizado de máquina (*Machine Learning*) e, dentro destes tópicos, detecção de novidades (*Novelty Detection*, DSND). Dentre as técnicas de mineração de fluxo de dados, classificadores podem ser utilizados para detectar padrões conhecidos e, em conjunto com algoritmos de detecção de novidades ou detecção de anomalias, detectar novos padrões. Essa capacidade é relevante em especial para o exemplo de detecção de intrusão, onde novidades na rede podem distinguir novas funcionalidades (entregues aos dispositivos após sua implantação em campo) de ataques por agentes externos, sem assinaturas existentes em bancos de dados de ataques conhecidos.

Análises como *Data Stream Mining* e DSND são geralmente implementadas sobre o paradigma de computação na nuvem (*Cloud Computing*) e, recentemente, sobre paradigmas como computação em névoa (*Fog Computing*). Para *fog*, além dos recursos em *cloud*, são explorados os recursos distribuídos pela rede desde o nó remoto até a *cloud*. Processos que dependem desses recursos são distribuídos de acordo com características como sensibilidade à latência, privacidade, consumo computacional ou energético.

1.1 Motivação

Um problema recente que une, em um único contexto, os métodos de computação em névoa, processamento de fluxo de dados e detecção de novidades nesses fluxos é a detecção de intrusão em redes de dispositivos IoT. Para tratar esse problema, a arquitetura IDSA-IoT, recentemente proposta por Cassales et al. (2019), aplica ao problema algoritmos atuais de detecção de novidades em fluxos, executando esses algoritmos em ambiente próximo aos dispositivos e avaliando-os quanto à detecção de intrusão.

Na arquitetura proposta, Cassales et al. (2019) avaliou os algoritmos ECSMiner (MASUD et al., 2011), AnyNovel (ABDALLAH et al., 2016) e MINAS (FARIA; CARVALHO; GAMA, 2015), sendo que o último mostrou resultados promissores. A arquitetura proposta foi avaliada com o conjunto de dados (*data set*) *Kyoto 2006+*, composto de dados coletados de 348 *Honeypots* (máquinas isoladas equipadas com diversos softwares com vulnerabilidades conhecidas expostas à Internet com propósito de atrair ataques) de 2006 até dezembro 2015. O *data set Kyoto 2006+* contém 24 atributos, 3 etiquetas atribuídas por detectores de intrusão comerciais e uma etiqueta distinguindo o tráfego entre normal, ataque conhecido e ataque desconhecido (CASSALES et al., 2019).

Contudo, o algoritmo MINAS ainda não foi implementado e avaliado com paralelismo, multi-processamento ou distribuição computacional, que são necessários para tratar fluxos de dados com grandes volumes e velocidades. O tratamento de distribuição em ambiente *fog computing* é essencial para aplicação deste algoritmo ao problema de detecção de intrusão em redes IoT, pois esta aplicação requer tempo de resposta mínimo e mínima

comunicação entre nós distantes, como aqueles na borda e na nuvem. Ainda observando o algoritmo MINAS, destaca-se a possível divisão em três partes semi-independentes, sendo elas treinamento, classificação e detecção de novidade; a classificação é o elemento central cujos resultados são utilizados para a identificação de intrusões.

Ainda no contexto de DSND como método de detecção de intrusão, outras propostas tratam do caso de fluxos com grandes volumes e velocidades, como é o caso de Viegas et al. (2019), que apresenta o *BigFlow* no intuito de detectar intrusão em redes do tipo *10 Gigabit Ethernet*, que podem produzir um volume considerável, atualmente impossível de ser processado em um único núcleo de processador (*single-threaded*). Essa implementação foi feita sobre uma plataforma distribuída processadora de fluxos (*Apache Flink*) executada em um cluster com até 10 nós de trabalho, cada um com 4 núcleos de processamento, totalizando 40 núcleos, para atingir taxas de até $10,72$ *Gbps*.

Os trabalhos de Cassales et al. (2019) e Viegas et al. (2019) abordam detecção de intrusão em redes utilizando algoritmos de ND em DS, porém com perspectivas diferentes. O primeiro investiga *IoT* e processamento em *fog* e baseia-se em um algoritmo genérico de detecção de novidade. O segundo trabalho trata de *backbones* e processamento em *cloud* e implementa o próprio algoritmo de detecção de novidade. Essas diferenças deixam uma lacuna onde, de um lado, tem-se uma arquitetura mais adequada para o ambiente *fog* com um algoritmo estado da arte de detecção de novidades, porém sem paralelismo e. Do outro lado da lacuna, tem-se um sistema escalável de alto desempenho porém almejando outro ambiente (*cloud*) e com um algoritmo menos preparado para os desafios de detecção de novidades.

1.2 Objetivos

Como estabelecido na Seção 1.1, a lacuna no estado da arte observada é a ausência de uma implementação de algoritmo de detecção de novidades que trate adequadamente os desafios de fluxo de dados contínuos (como volume e velocidade do fluxo, evolução e mudança de conceito) e considere o ambiente de computação em névoa aplicada à detecção de intrusão. Seguindo a comparação entre algoritmos desse gênero realizada por Cassales et al. (2019), esta pesquisa escolheu investigar o algoritmo MINAS (FARIA; CARVALHO; GAMA, 2015) para receber o tratamento necessário para adequá-lo ao ambiente de névoa e para fluxos de grandes volumes e velocidades.

Portanto, seguindo os trabalhos do Grupo de Sistemas Distribuídos e Redes (GSDR) da Universidade Federal de São Carlos (UFSCar), propõem-se a construção de uma

aplicação que implemente o algoritmo MINAS de maneira escalável e distribuível para ambientes de computação em névoa e a avaliação dessa implementação com experimentos baseados na literatura usando conjunto de dados públicos relevantes. O resultado esperado é uma implementação compatível em qualidade de classificação ao algoritmo MINAS e

passível de ser distribuída em um ambiente de computação em névoa aplicado à detecção de intrusão.

Com foco no objetivo geral, alguns objetivos específicos são propostos:

- ❏ Implementar o algoritmo MINAS de maneira distribuída sobre uma plataforma de processamento distribuída de fluxos de dados;
- ❏ Avaliar a qualidade de detecção de intrusão em ambiente distribuído conforme a arquitetura IDSA-IoT;
- ❏ Avaliar o desempenho da implementação em ambiente de computação em névoa.

1.3 Proposta Metodológica

Para cumprir os objetivos citados na Seção 1.2, foi identificada a necessidade de um processo exploratório seguido de experimentação. Tal processo inclui a revisão da literatura, tanto acadêmica quanto técnica, seguida da experimentação através de implementação de aplicação e testes.

O foco da revisão da literatura acadêmica é em trabalhos que abordem processamento de fluxos de dados, classificação de fluxo de dados, detecção de novidades em fluxo de dados e processamento distribuído de fluxo de dados. O objetivo da revisão é o estabelecimento do estado da arte desses assuntos, de forma que alguns desses trabalhos sirvam para comparações e relacionamentos. Além disso, desses trabalhos buscam-se métricas de qualidade de classificação (por exemplo, taxa de falso positivo e matriz de confusão) e métricas de escalabilidade (como taxa de mensagens por segundo e escalabilidade vertical ou horizontal).

A revisão da literatura técnica será focada em plataformas, ferramentas e técnicas para realizar a implementação proposta. Portanto, são selecionadas plataformas de processamento distribuído de DS e técnicas de aprendizado de máquina associadas a elas. Dessa revisão também serão obtidas técnicas ou ferramentas necessárias para extração das métricas de avaliação, bem como *data sets* públicos relevantes para detecção de novidades em DS.

Uma vez definidos o estado da arte, as ferramentas técnicas e os *data sets*, o passo seguinte é a experimentação. Nesse passo, será desenvolvida uma aplicação na plataforma escolhida que, com base no algoritmo MINAS (FARIA; CARVALHO; GAMA, 2015), irá classificar e detectar novidades em DS. Também nesse passo, a implementação será validada comparando os resultados de classificação obtidos com os resultados de classificação do algoritmo original MINAS. Posteriormente, serão realizados experimentos com a implementação e variações em *data sets* e cenários de distribuição em *fog*, coletando as métricas de classificação e escalabilidade.

Ao final, a aplicação, resultados, comparações e discussões serão publicados nos meios e formatos adequados, como repositórios técnicos, eventos ou revistas acadêmicas.

1.4 Organização do trabalho

O restante desse trabalho segue a estrutura: Capítulo 2 aborda conceitos teóricos e técnicos que embasam esse trabalho; Capítulo 3 enumera e discute trabalhos relacionados e estabelece o estado da arte do tema detecção de novidade em fluxos de dados e seu processamento; Capítulo **??** descreve a proposta de implementação, discute as escolhas de plataformas e resultados esperados. Também são discutidos no Capítulo **??** os desafios e resultados preliminares encontrados durante o desenvolvimento do trabalho. Capítulo 4 adiciona considerações gerais e apresenta o plano de trabalho e cronograma até a defesa do mestrado.

Capítulo 2

Fundamentos Científicos e Tecnológicos

Este Capítulo aborda conceitos que embasam esse trabalho, conceitos teóricos de ambientes e arquiteturas de computação distribuída e detecção de novidade e conceitos técnicos, como plataformas de processamento distribuído de fluxo de dados e o algoritmo MINAS.

2.1 Ambientes de Computação Distribuída

Esta Seção relaciona três ambientes de computação distribuída habitualmente utilizados para o processamento de dados massivos relacionados a redes de dispositivos IoT, entre outras aplicações. A computação em nuvem (*cloud computing*) é aplicada a vários problemas e neste trabalho seu papel em sistemas IoT é fornecer vastos recursos e garantias e em que dispositivos enviam todos dados relevantes ao sistema. O segundo e terceiro ambiente são computação de borda (*edge computing*) e a computação em névoa (*fog computing*), que utiliza os recursos computacionais distribuídos presentes em nós localizados entre os dispositivos de borda e a nuvem, com diversas intenções, desde privacidade até redução de latência.

2.1.1 Computação em Nuvem

A computação em nuvem (*cloud computing*), ou simplesmente nuvem (*cloud*), habilita o acesso através da rede a um grupo compartilhado de recursos de computação configuráveis, como servidores, redes, aplicações, armazenamento, etc. Tais recursos podem ser

provisionados ou liberados sob demanda rapidamente com o mínimo esforço de gerenciamento e mínima interação com o provedor destes recursos (MELL; GRANCE, 2012). As principais características do ambiente *cloud computing*, segundo Mell e Grance (2012) são:

- **Serviço sob Demanda:** o cliente pode provisionar ou liberar capacidades de computação (ex: tempo de processamento e armazenamento) conforme o necessário, sem requerer interação com o provedor de serviço;
- **Amplo acesso à rede:** o acesso aos recursos de computação e capacidades ocorre pela rede através de mecanismos padrões que permitem o acesso por plataformas heterogêneas (celulares, computadores, tablets, etc.)
- **Agrupamento de recursos:** para servir múltiplos clientes, os recursos de computação são agrupados usando o modelo *multi-tenancy* com recursos físicos e virtuais diferentes dinamicamente atribuídos e reatribuídos de acordo com a demandas do clientes;
- **Elasticidade:** as capacidades de computação são rapidamente provisionadas ou liberadas, em alguns casos automaticamente, para escalar conforme a demanda;
- **Serviço mensurado:** os recursos de computação são monitorados, controlados e reportados para o provedor de serviços e para o cliente fornecendo transparência sobre as capacidades que foram consumidas.

Segundo, Mell e Grance (2012), a implantação da Computação em Nuvem pode ocorrer através dos seguintes modelos:

- **Nuvem privada:** a infraestrutura da nuvem é provisionada e dedicada para um único cliente ou organização. Nesse modelo, o cliente gerencia e controla a infraestrutura, ou pode delegar essas tarefas a uma outra empresa. A infraestrutura pode estar dentro ou fora das instalações da organização proprietária;
- **Nuvem comunitária:** a infraestrutura de nuvem é fornecida para um grupo exclusivo de clientes que compartilham um mesmo interesse (requerimentos de segurança, desempenho, políticas, etc.). Esse tipo de nuvem pode ser gerenciado pelo próprio grupo, ou por outra organização, podendo estar dentro ou fora das instalações das empresas proprietárias;
- **Nuvem pública:** É gerenciada e operada por um provedor de nuvem e a infraestrutura é provisionada e oferecida para uso público.
- **Nuvem híbrida:** a infraestrutura desse tipo de nuvem é uma composição de dois ou mais modelos de implantação de *cloud* (privada, pública e comunitária) que

formam uma entidade única e são unidos por tecnologias padronizadas que habilitam a portabilidade de dados e aplicações.

2.1.2 Computação de Borda

A computação de borda (*edge computing*) refere-se às tecnologias que permitem que a computação seja executada na borda da rede. Define-se borda ou *edge* como qualquer recurso de computação e de rede ao longo do caminho entre as fontes de dados e os data centers da nuvem (SHI et al., 2016). Na borda, é possível fazer armazenamento, processamento e descarregamento de dados, assim como distribuir as requisições e entregar os serviços das nuvens aos usuários. Shi et al. (2016) ressalta que essas capacidades (dentre outras) dos nós da borda (*edge nodes*) possibilitam que a computação de borda reduza a latência na resposta da nuvem, pré-processando os dados nos nós da borda, aproveitando melhor a banda e a transmissão de dados, e também consumindo menos recursos de computação na nuvem. Além disso, o autor ainda acrescenta que a computação de borda pode aumentar a privacidade dos dados, uma vez que eles podem ser processados no próprio dispositivo final.

A computação de borda tenta trazer a computação mais próxima das fontes de dados. Como é observado na figura, os componentes desse tipo de computação podem ser tanto produtores como consumidores, não só requisitando serviços e conteúdo da nuvem, mas também realizando tarefas da nuvem. Algumas aplicações da computação de borda incluem: análise de vídeo; em sistemas críticos para redução de latência; descarregar a nuvem de parte da computação; privacidade dos dados produzidos, mantendo-os fora de ambientes públicos; redução das cargas de dados na rede e processamento distribuído de sensoriamento massivo em cidades inteligentes (SHI et al., 2016).

2.1.3 Computação em Névoa

Dastjerdi e Buyya (2016) e IEEE Communications Society (2018) mencionam que a enorme massa de dados gerados por ambientes IoT pode ser processada em nuvem, entretanto a latência produzida pela transferência desses dados para a nuvem e o retorno do resultado pode não ser toleradas por sistemas críticos que sejam sensíveis a latência (monitoramento de saúde e resposta a emergências). IEEE Communications Society (2018) ainda acrescenta que enviar tantos dados à nuvem para processamento e armazenamento pode ser ineficiente e não escalável, devido à saturação de dados na rede. O ambiente *edge computing* foi proposto para trazer o processamento e armazenamento para os dispositivos de borda tentando solucionar esses problemas. Porém, dispositivos de borda comumente não podem lidar com várias aplicações IoT competindo pelos seus recursos limitados, o que poderia causar a contenção dos recursos e o aumento na latência do processamento

(DASTJERDI; BUYYA, 2016). Portanto, para solucionar estas questões de latência e capacidade limitada dos dispositivos de borda, a computação em névoa foi proposta.

A computação em névoa (*fog computing*) é um paradigma que distribui as capacidades de computação, armazenamento e rede entre os nós próximos das fontes dados e dos dispositivos finais, mas não necessariamente localizados na borda, dando a esses nós características de uma nuvem (BONOMI et al., 2012; DASTJERDI; BUYYA, 2016; IEEE Communications Society, 2018). Esse tipo de computação evita a sobrecarga dos dispositivos de borda. Bonomi et al. (2012) e Dastjerdi e Buyya (2016) consideram computação em névoa como complementar da computação em borda, podendo a computação em névoa aproveitar os recursos da nuvem e da borda. IEEE Communications Society (2018) considera que a principal diferença entre esses dois tipos de computação está no número de camadas. Enquanto *edge computing* tem camadas menores, pois atua só nos dispositivos de borda, *fog computing* tem mais camadas e um modelo hierárquico, pois não atua só na camada de borda.

Segundo Bonomi et al. (2012) e Dastjerdi e Buyya (2016), as principais características da computação em névoa são:

- **Mobilidade:** é essencial que as aplicações *fog* sejam capazes de se comunicar com dispositivos móveis, por exemplo, utilizando protocolos que considerem a mobilidade dos nós;
- **Heterogeneidade:** os nós nesse tipo de paradigma possuem configurações e formatos diferentes e podem estar implantados em ambientes distintos;
- **Baixa Latência:** foi proposta para atender aplicações que requeiram baixa latência (monitoramento de saúde, jogos, realidade aumentada, etc.);
- **Distribuição geográfica:** computação em névoa pode possuir milhares de sensores e dispositivos distribuídos geograficamente, com consciência de suas localizações (*location awareness*);
- **Alto número de nós:** seguindo os ambientes IoT, a computação em névoa pode ser composta por milhares de nós;
- **Interoperabilidade e federação:** os componentes da computação em névoa devem ser capazes de interoperar, e o serviços devem ser federados ;
- **Uso de fluxo de dados e aplicações em tempo real:** a computação em névoa pode envolver aplicações que processam em lote, mas na maior parte das vezes envolve aplicações com requisito de processamento em tempo real, e para isso fazem o uso de fluxo de dados. Por exemplo, os sensores de um rede IoT escrevem a informação no fluxo de dados, a informação é processada, ações são inferidas e traduzidos em ações nos componentes atuadores.

Algumas aplicações para computação em névoa são: cidades inteligentes e semáforos inteligentes que enviam sinais de alerta aos veículos e coordenam os sinais verdes com outros semáforos através de sensores (veículos, pedestres, ciclistas); na área de saúde, para monitorar e prever situações de pacientes que estão conectados a sensores; em prédios inteligentes, que são dotados de sensores de umidade, temperatura, qualidade do ar, ocupação, sendo que a partir das informações deles, é possível alertar os ocupantes do prédio em algum caso de emergência.

2.2 Mineração de Dados e Fluxo de Dados

A Mineração de Dados é o processo de descoberta de padrões em conjuntos de dados utilizando métodos derivados de aprendizagem de máquina, estatística e banco de dados (GABER; ZASLAVSKY; KRISHNASWAMY, 2005). Além de mineração de dados tradicional, *Big Data* trata de conjuntos de dados que não podem ser processados em tempo viável, devido a limitações como memória ou armazenamento principal.

Definição 1. *Um Fluxo de Dados S é uma sequência massiva, potencialmente ilimitada de exemplos multi-dimensionais $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n, \ldots$ recebida em instantes $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_n, \ldots$ (AGGARWAL et al., 2003).*

Além da dimensão de armazenamento, outra dimensão que afeta a maneira como dados são modelados e manipulados é o tempo. Técnicas e algoritmos de mineração de fluxo de dados atendem a esses desafios utilizando restrições como apenas uma leitura do conjunto de dados e baixo tempo de processamento na construção de seus algoritmos (GAMA; RODRIGUES, 2007; GABER; ZASLAVSKY; KRISHNASWAMY, 2005).

As características de fluxos de dados e mineração de dados e os requisitos de seu processamento regularmente superam as capacidades computacionais de um único nó computacional convencional, de forma que a distribuição dos requisitos em múltiplos nós computacionais em um sistema distribuído pode ser necessária (GABER; ZASLAVSKY; KRISHNASWAMY, 2005).

Computação distribuída é a área da ciência da computação que estuda sistemas em que os componentes são localizados em diferentes computadores (nós), que comunicam-se apenas por troca de mensagens e, para que o objetivo do sistema seja atingido, a cooperação entre os nós é necessária. Outras propriedades de um sistema distribuído são a concorrência entre os nós e possibilidade de falhas em partes independentes (TANENBAUM; STEEN, 2018).

Para a construção de sistemas que apliquem técnicas de mineração de fluxos de dados são necessárias bibliotecas e plataformas (*frameworks*) que são abordadas na Seção 2.3.

2.3 Arquiteturas e Plataformas de Processamento de Fluxos

Tradicionalmente, aplicações foram construídas com um sistema gerenciador de banco de dados (SGBD) relacional ou não-relacional associado. Essa arquitetura, nomeada de "arquitetura totalmente incremental" por Marz e Warren (2015), foi evoluída e simplificada iterativamente durante décadas de uso, porém ela não é adequada para sistemas em tempo real, como os sistema de fluxo de dados. O volume e a velocidade de dados em um *Data Stream* leva à necessidade de distribuir o processamento, acrescentando poder computacional a cada nó adicionado. Porém, desafios como comunicação eficiente e sincronização de estado entre os nós, assim como tolerância a falhas, aumentam a complexidade de construção de um sistema distribuído em relação a um sistema tradicional.

Para mitigar problemas associados à construção de sistemas *Big Data* e *Data Streams*, arquiteturas de processamento de fluxo de dados distribuído foram propostas, como a arquitetura *Lambda* (MARZ; WARREN, 2015) e *Kappa* (KREPS, 2014), além de diversas plataformas, tanto de *Big Data* com características de tempo real, como especializadas em fluxo de dados.

MapReduce é a primeira plataforma de processamento de conjuntos massivos de dados que atingiu uso generalizado. Nessa implementação, uma biblioteca gerencia a distribuição, paralelização, tolerância a falhas e balanceamento de carga. Ao usuário da biblioteca resta implementar duas funções: *Map*, que recebe um par ordenado $(chave, valor)$ e emite um conjunto de pares intermediários na mesma estrutura; *Reduce*, que recebe uma chave e um conjunto de valores gerado pelo agrupamento de pares com essa mesma chave (DEAN; GHEMAWAT, 2004).

Em prática, um *cluster MapReduce* tem centenas de processadores e o conjunto de dados é armazenado em um sistema de arquivos distribuído que é lido pela plataforma com programas escritos por usuários sendo executados sob supervisão de um nó mestre. Essa implementação tem esquema geral de processamento em lotes que não atende o requisito de baixa latência. *MapReduce* é uma das principais influências na criação da arquitetura *Lambda* (MARZ; WARREN, 2015).

Apache Hadoop é uma coleção de ferramentas, incluindo: *Hadoop Distributed File System* (HDFS, um sistema de arquivos distribuído), *Hadoop YARN* um gerenciador de recursos em cluster e escalonador de trabalhos e, *Hadoop MapReduce*, um sistema baseado em *YARN*, implementando o modelo *MapReduce* (Apache Hadoop, 2020).

Apache Spark, analogamente ao *Hadoop*, é um *framework* para construção de sistemas de computação distribuída em *cluster*, com garantias de tolerância a falhas. No entanto, o modelo de processamento diverge significativamente do tradicional *MapReduce*, utilizando em lugar do HDFS um multiconjunto imutável distribuído (*Resilient Distributed Dataset* - RDD) com um escalonador de trabalhos representados por grafos acíclicos direcionados

(*directed acyclic graph* - DAG), otimizador de consultas e motor de execução (Apache Spark, 2020).

Uma das extensões de *Apache Spark* é *Spark Streaming*, que é um sistema de processamento de fluxo de dados escalável e tolerante a falhas (ZAHARIA et al., 2016a; ZAHARIA et al., 2016b). *Spark Streaming* implementa processamento incremental de fluxo de dados usando o modelo de fluxos discretizados em que dividem-se os dados de entrada em micro-lotes (ex: a cada 100 milissegundos) e combinam-se regularmente com o estado nos RDDs para produzir novos resultados (ZAHARIA et al., 2016a). Essa estratégia traz benefícios sobre os sistemas de fluxos de dados distribuídos tradicionais, pois permite a consistência e recuperação de falhas rapidamente, devido à (*RDD lineage*) e à combinação do fluxo de dados com consultas em lotes e interativas (ZAHARIA et al., 2016b; LOPEZ, 2018).

Apache Storm é um sistema de computação tolerante a falhas em tempo real que de fluxo de dados (FOUNDATION, 2020; LOPEZ, 2018). Ao invés de executar trabalhos (*jobs*) como algumas ferramentas citadas anteriormente, *Apache Storm* . Os *jobs* eventualmente finalizam, e as topologias executam continuamente até serem finalizadas por comandos. Uma topologia constitui-se de processos trabalhadores (*workers*) sendo executados em um *cluster* de nós que são gerenciados pelo nó mestre que além de coordenar e distribuir execução, monitora falhas. Uma topologia pode ser representada por um grafo de computação direcionado acíclico (DAG).

O *Apache Flink* é uma plataforma de processamento distribuído para computação com estado gerenciado (*stateful*) sobre fluxo de dados limitados (têm início e fim) e ilimitados (não têm fim definido) (Apache Flink, 2020). Essa plataforma segue um paradigma que abrange o processamento de fluxos de dados contínuos e o processamento em lote (CARBONE et al., 2015; LOPEZ, 2018). O *Apache Flink* pode ser integrado a vários gerenciadores de *cluster* comuns, como *Hadoop Yarn*, *Apache Mesos*, e *Kubernetes*, mas também pode ser configurado para ser executado como um *cluster stand-alone*. Já o acesso programático a essa plataforma pode ser feito através das linguagens Java, Scala ou Python.

2.4 Detecção de Novidade

No âmbito de classificação de dados, parte da área de aprendizado de máquina, os métodos de detecção de novidade (*Novelty Detection*, DSND) lidam com o reconhecimento e a classificação de exemplos que diferem de exemplos anteriores (PERNER, 2007; GAMA; RODRIGUES, 2010). Esses métodos tratam da classificação em fluxos de dados que evoluem com o tempo, levando em consideração as características desse tipo de fluxos.

Tratando-se de fluxos de dados contínuos, são características dos padrões observados: evolução de conceito (*Concept Evolution*) em que novos padrões podem surgir; desaparecimento ou recorrência de conceito, em que padrões podem desaparecer e também podem reaparecer; mudança de conceito (*Concept Drift*, também nomeado deriva ou desvio) onde um padrão gradualmente se transforma; presença de ruído e *outliers* (GAMA; RODRIGUES, 2010).

Os métodos de DSND são aplicados a diversos problemas como detecção de intrusos (COULL et al., 2003; SPINOSA; CARVALHO; GAMA, 2008; VIEGAS et al., 2019; CASSALES et al., 2019), detecção de falhas (ZHANG et al., 2006), diagnósticos médicos (PERNER, 2009), detecção de regiões de interesse em imagens (SINGH; MARKOU, 2004), detecção de fraudes (WANG et al., 2003; ABDALLAH; MAAROF; ZAINAL, 2016), filtros de spam (HAYAT; HASHEMI, 2010) e detecção de variações comportamentais em um jogador (VALLIM et al., 2013).

Alguns métodos de DSND utilizam tratam de novidades como uma classificação de uma ou duas classes (binariamente) onde um conceito representa a classe normal e as anomalias são representadas pela falta de conceito no modelo ou como um segundo conceito no modelo. Além da abordagem de classificação binária, múltiplos conceitos em um mesmo conjunto de dados, para isso é necessário abordar DSND como classificação multi-classe. Alguns métodos que abordam DSND como classificação multi-classe não atendem completamente características de conjuntos com evolução temporal, como *Concept Evolution* e *Concept Drift*, deixando de detectar múltiplos padrões que surgem simultaneamente num intervalo de avaliação (FARIA et al., 2015; GAMA; RODRIGUES, 2010).

A maioria dos métodos de DSND são construídos seguindo a abordagem de aprendizado *Offline-Online*. Essa abordagem estabelece que o método seja dividido em duas fases: a primeira fase (*Offline*) usa um conjunto de exemplos rotulados para deles extrair conceitos conhecidos e gerar um modelo; a segunda fase (*Online*) consome um conjunto ou fluxo de exemplos não rotulados e detecta padrões-novidade. Além de detectar padrões-novidade, alguns algoritmos classificam cada exemplo em um dos conceitos do modelo, ou marca o exemplo como desconhecido. Ainda na segunda fase, para atualizar o modelo, os exemplos marcados como desconhecidos são utilizados para a extração de novos conceitos ou variações em conceitos conhecidos (GAMA; RODRIGUES, 2010).

Dentre os métodos de DSND que baseiam-se em aprendizado *Offline-Online*, muitos são baseados em algoritmos de agrupamento não supervisionados, tanto para construção

do modelo inicial como na extração de novos conceitos dos exemplos não explicados pelo modelo marcados como desconhecidos (SPINOSA; CARVALHO; GAMA, 2009; MASUD et al., 2011; FARIA et al., 2013).

2.4.1 O algoritmo MINAS

Um algoritmo de DSND que tem recebido atenção nos últimos anos é o algoritmo MINAS, originalmente proposto por Faria et al. (2013), refinado por Faria, Carvalho e Gama (2015) e recentemente aprimorado por Silva (2018), com o uso de conceitos *Fuzzy*, e expandido por Costa (2019), para tratar problemas multi-rótulo além dos problemas multi-classe já tratados na versão original. Esse algoritmo segue a abordagem de duas fases no modelo *Offline-Online* e usa por base algoritmos de agrupamento não supervisionados como *K-means* e *CluStream*.

O algoritmo MINAS em sua fase *Offline* consome um conjunto de treinamento contendo exemplos etiquetados. Esse conjunto de treinamento é dividido em grupos usando como chave a etiqueta, e para cada grupo de exemplos o método de agrupamento (*clustering*) é executado. O método de agrupamento objetiva resumir um conjunto maior de exemplos em um conjunto menor de *micro-clusters*.

Um *micro-cluster* é uma tupla de quatro components $(N, \mathbf{LS}, \mathbf{SS}, T)$ derivados dos exemplos representados por este *micro-cluster*, onde: N número de exemplos, **LS** soma linear dos exemplos, **SS** soma quadrada dos exemplos, T instante de chegada do último exemplo adicionado ao *micro-cluster*. Deste sumário extrai-se, entre outras estatísticas, o centro e raio que são utilizados na operação de classificação da fase *Online*. A cada *micro-cluster* é adicionada a etiqueta do grupo original e todos *micro-clusters* são arranjados em um único conjunto formando o modelo de decisão.

Na fase *Online*, listada no Algoritmo **??**, o algoritmo MINAS opera com três operações: classificação de novos exemplos, detecção de padrões-novidade e atualização do modelo de decisão (FARIA; CARVALHO; GAMA, 2015). O primeiro método é o de classificação, onde exemplos do fluxo de dados são consumidos e avaliados pelo modelo de decisão. O modelo de decisão avalia cada exemplo calculando a distância euclidiana entre o exemplo e todos *micro-clusters* do modelo, selecionando o *micro-cluster* de menor distância. Se a distância entre o exemplo e o centro do *micro-cluster* for menor que o raio do *micro-cluster*, o exemplo é classificado com a etiqueta do *micro-cluster* e o sumário estatístico do *micro-cluster* é atualizado. Caso a distância (mínima no modelo) seja maior que o raio, o exemplo é marcado como desconhecido e armazenado em conjunto próprio (FARIA; CARVALHO; GAMA, 2015).

O segundo método da fase *Online* é a detecção de padrões novidade, que é executada quando o tamanho do conjunto de desconhecidos é maior que um parâmetro predefinido. Esse método executa o agrupamento (*clustering* descrito na fase *Offline*) e valida os *micro-clusters* gerados verificando sua representatividade e coesão.

2.4.2 MINAS

MINAS (FARIA; CARVALHO; GAMA, 2015) is an offline-online DSND algorithm, meaning it has two distinct phases. The first phase (offline) creates an initial model set with several clusters based on a clustering algorithm with a training set. Each cluster can be associated with only one class of the problem, but each class can have many clusters.

During its online phase, which is the main focus of our work, MINAS performs three tasks in (near) real-time, in summary, classification, novelty detection, and model update tasks in a potentially infinite data stream, as shown in Algorithm 1.

MINAS attempts to classify each incoming unlabeled instance according to the current decision model. Instances not explained by the current model receive an *unknown* label and are stored in an unknowns-buffer. When the unknowns-buffer reaches a preset threshold, MINAS executes the Novelty Detection function. After a set interval, samples in the unknowns-buffer are considered to be noise or outliers and removed. The algorithm also has a mechanism to forget clusters that became obsolete and unrepresentative of the current data stream distribution, removing them from the Model and storing in a Sleep Model for possible recurring pattern detection (FARIA; CARVALHO; GAMA, 2015).

```
   Input: ModelSet, inputStream
   Output: outputStream
   Parameters: cleaningWindow, noveltyDetectionTrigger
 1 Function MinasOnline(ModelSet, inputStream):
 2     UnkownSet ← ∅, ModelSleepSet ← ∅ ;
 3     lastCleanup ← 0 , noveltyIndex ← 0;
 4     foreach sample_i ∈ inputStream do
 5         nearest ← nearestCluster (sample, ModelSet);
 6         if nearest.distance < nearest.cluster.radius then
 7             sample.label ← nearest.cluster.label;
 8             nearest.cluster.lastUsed ← i ;
 9         else
10             sample.label ← unknown;
11             UnkownSet ← UnkownSet ∪ sample;
12             if | UnkownSet | ≥ noveltyDetectionTrigger then
13                 novelties ← NoveltyDetection (ModelSet ∪ ModelSleepSet,
                    *UnkownSet);
14                 ModelSet ← ModelSet ∪ novelties;
15             if i > ( lastCleanup + cleaningWindow) then
16                 ModelSet ← moveToSleep (ModelSet, *ModelSleepSet,
                    lastCleanup);
17                 UnkownSet ← removeOldSamples (UnkownSet, lastCleanup);
18                 lastCleanup ← i;
19         outputStream.append(sample);
```

Algorithm 1: Our interpretation of MINAS baseado em (FARIA; CARVALHO; GAMA, 2015)

The Novelty Detection function, illustrated in Algorithm 2, groups the instances to form new clusters, and each new cluster is validated to discard the non-cohesive or unrepresentative ones. Valid clusters are analyzed to decide if they represent an extension of a known pattern or a completely new pattern. In both cases, the model absorbs the valid clusters and starts using them to classify new instances.

```
   Parameters: minExamplesPerCluster, noveltyFactor
 1 Function NoveltyDetection(Model, Unknowns):
 2     newModelSet ← ∅;
 3     foreach cl in clustering (Unknowns) do
 4         if | cl.sampleSet | ≥ minExamplesPerCluster then
 5             (distance, near) ← nearestCluster (cl, Model);
 6             if distance < near.radius × noveltyFactor then
 7                 cl.label ← near.label;
 8                 cl.type ← extension;
 9             else
10                 cl.label ← noveltyIndex;
11                 noveltyIndex ← noveltyIndex +1;
12                 cl.type ← novelty;
13             Unknowns ← Unknowns − cl.sampleSet;
14             newModelSet ← newModelSet ∪ cl;
15     return newModelSet;
```

Algorithm 2: MINAS (FARIA; CARVALHO; GAMA, 2015) Novelty Detection task.

Para atribuição de etiquetas aos *micro-clusters* gerados, o algoritmo MINAS encontra no modelo atual o *micro-cluster* mais próximo pela distância euclidiana e classifica em dois tipos de conceito. Se a distância é menor que um parâmetro predefinido, o novo *micro-cluster* gerado recebe como etiqueta o valor de extensão de conceito conhecido. Caso contrário, se o novo *micro-cluster* está mais distante, um novo conceito foi encontrado e a etiqueta marca um padrão novidade. Após a atribuição da etiqueta do novo *micro-cluster*, ele é adicionado ao modelo de decisão.

O algoritmo MINAS, como já foi discutido na Seção 2.4.1, classifica exemplos e detecta novidades em DS e considera em sua composição *concept drift* e *concept evolution*, sendo capaz de classificar como extensão de classe conhecida e identificar padrões novidade sem intervenção de especialista (FARIA; CARVALHO; GAMA, 2015). Neste trabalho, consideram-se algoritmos derivados do algoritmo MINAS aqueles apresentados em trabalhos publicados após 2016, que estendem a implementação original seguindo sua estrutura básica.

2.4.3 Algoritmo FuzzyND

O algoritmo FuzzyND, derivado do MINAS foi proposto por Da Silva et al. (2018). FuzzyND incrementa o algoritmo original, aplicando a ele teorias de conjuntos *fuzzy* pela

modificação da representação dos *clusters*. A modificação afeta o método de construção de *clusters*, método de classificação de exemplos e método de detecção de novidades de acordo com a nova representação.

A avaliação do algoritmo FuzzyND foi feita por meio de experimentos usando 3 *data sets* sintéticos (*MOA3*, *RBF*, *SynEDC*) e por comparação com o MINAS. O método de avaliação utilizado baseia-se na matriz de confusão incremental descrita por Faria et al. (2015), extraindo dessa matriz duas métricas: acurácia (*Macro F-Score*) (SOKOLOVA; LAPALME, 2009) e taxa de desconhecidos (*UnkR*) (FARIA; CARVALHO; GAMA, 2015). Em geral, o algoritmo FuzzyND detecta melhor novidades e, consequentemente, é mais robusto a valores atípicos (*outlier*), porém perde a capacidade de reconhecer padrões recorrentes.

2.4.4 Algoritmos MINAS-LC e MINAS-BR

O algoritmo MINAS-LC foi proposto por Costa (2019) e trata a classificação multi-rótulo, porém não trata evoluções de conceito (*Concept Evolution*). As alterações fundamentais propostas são: a representação de *cluster* onde MINAS-LC troca a etiqueta, que era única, por uma multi-rótulo; a transformação de problema aplicada ao conjunto de treinamento para transformá-lo de um conjunto multi-rótulo para um conjunto multi-classe (simplificação) em duas variações *Label Powerset* e *Pruned Sets* com mineração de conjunto de itens frequentes.

Já o trabalho de Júnior et al. (2019), estende o algoritmo original para que classifique um exemplo com uma ou mais etiquetas usando a transformação *Binary Relevance*, o que deu origem ao algoritmo MINAS-BR. O algoritmo modifica a representação do modelo, originalmente conjunto de *clusters*, para um grupo de *clusters* por classe (etiqueta). Também modifica o método de agrupamento, substituindo a inicialização do algoritmo *K-means*, originalmente aleatória, pelo algoritmo *Leader Incremental Clustering* (VIJAYA; MURTY; SUBRAMANIAN, 2004).

O algoritmo MINAS-BR também é experimentalmente avaliado com 4 *data sets* sintéticos: *MOA-3C-5C-2D*, *MOA-5C-7C-2D*, *MOA-5C-7C-3* da ferramenta MOA (BIFET et al., 2010) e *4CRE-V2* [1] gerados pelo método *Radial Basis Function* (SOUZA et al., 2015; JÚNIOR et al., 2019). O algoritmo MINAS-BR foi comparado com 7 algoritmos da literatura também disponíveis na ferramenta MOA (BIFET et al., 2010), diferente da avaliação do FuzzyND que compara diretamente com MINAS. Para análise, os 7 algoritmos foram divididos em dois grupos (JÚNIOR et al., 2019). O primeiro grupo de 3 algoritmos com acesso às etiquetas corretas para atualização do modelo e com a técnica ADWIN (*ADaptive WINdowing*) para detectar mudanças de conceito (*Concept Drift*) O

[1] A versão original do *data set* 4CRE-V2 está disponível em <https://sites.google.com/site/nonstationaryarchive/home>.

segundo grupo com os 4 algoritmos sem acesso às etiquetas corretas, ou seja, sem *feedback* externo, mesma condição do MINAS-BR (JÚNIOR et al., 2019).

A avaliação elencada por Júnior et al. (2019) leva em consideração que as classes contidas no conjunto de testes podem não ter correlação direta com os padrões identificados pelos algoritmos. Para tratar a divergência, uma estratégia baseada em proposta anterior por Faria et al. (2015) foi apresentada com alterações para exemplos multi-rótulo. Após associação entre padrões de novidade e classes novidade foi possível calcular métricas tradicionais. A estratégia é executada na fase de classificação seguindo as regras:

1. após o consumo do exemplo X_n;
2. para todo padrão P_i (etiqueta atribuída) identificado sem associação até o momento;
3. com classes novidade y_j (etiqueta real) presentes em exemplos antes X_n;
4. preenche-se a tabela de contingência $\mathbf{T}_{(i,j)}$ relacionando padrão P_i e classe y_j;
5. calcula-se o grau de dependência *F1* derivado da tabela de contingência $F1_{(i,j)} = f(\mathbf{T}_{(i,j)})$;
6. valores $F1_{(i,j)} = 0$ são descartados;
7. dentre os valores restantes: o padrão P_i é associado à classe y_j se $F1_{(i,j)}$ é máximo.

As métricas utilizadas por Júnior et al. (2019) após a associação de classes e padrões são as tradicionais taxa de desconhecidos (*UnkRM*) e *F1M*. Os resultados apresentados indicam que MINAS-BR capturou todas as novidades dos *data sets* sintéticos de teste e mostrou, como esperado, melhores métricas que os 4 algoritmos equivalentes da literatura ficando abaixo dos 3 com *feedback* externo.

Os trabalhos abordados nessa Seção 2.4, têm em comum, além do algoritmo base, as métricas de avaliação acurácia (*Macro F-Score* e *Macro F-Measure* F1M) e taxa de desconhecidos, aplicadas com devido tratamento. Também é comum entre eles o uso de *data sets* sintéticos. Outro potencial não explorado do MINAS é em aplicações reais, ou seja, consumindo além de *data sets* reais, fluxos realistas em ambientes simulados ou reais porém considerando uso de recursos computacionais.

Observando a arquitetura dos algoritmos abordados na Seção 2.4, nota-se as semelhanças: a fase offline centrada no processo de agrupamento e criação de modelo; a fase online dividida em classificação (com atualização das estatísticas do modelo) e detecção de padrões, onde novamente o processo de agrupamento é central. Portanto, apesar de outros trabalhos expandirem o algoritmo com diferentes técnicas, seu núcleo continua relevante.

Propostas de modificação do algoritmo MINAS estão longe de serem exauridas. Não cabe ao presente trabalho expandir e validar conceitos de aprendizagem de máquina, porém alguns exemplos mencionados ainda não abordados são (Da Silva et al., 2018; SILVA, 2018; JÚNIOR et al., 2019):

a) diferentes métodos de cálculo de distância entre pontos além da distância euclidiana;

b) a mudança de representação de *clusters*, atualmente hiper-esferas (COSTA, 2019), para hiper-cubos tratando *data sets* onde as características representadas pelas dimensões são completamente independentes;

c) um modo interativo onde o *cluster* é formado, mostrado ao especialista que o classifica como inválido (ruído ou não representativo) ou válido, podendo conter uma ou mais classes e, se contiver mais que uma classe corte em grupos menores até conter somente uma classe;

d) ainda considerando interação com especialista, a possibilidade de injetar um exemplo não pertencente a uma classe, ou seja, marcar o exemplo como não pertencente a uma classe para mantê-lo na memória de desconhecidos e, eventualmente forçar criação de um *cluster* que represente uma classe geometricamente próxima mas semanticamente distinta;

e) na fase *offline* a verificação de sobreposição de *clusters* pertencentes a classes distintas e tratamento adequado.

Capítulo 3

Trabalhos Relacionados

Este Capítulo trata dos trabalhos relacionados e apresenta aspectos do estado da arte dos tópicos Detecção de Novidades em Fluxos de Dados, e Processamento Distribuído de Fluxos de Dados.

Nesta Capítulo, abordam-se trabalhos que aplicam em de fluxo de dados em tempo real. Um sumário dos trabalhos abordados pode ser visto na Tabela 1.

Tabela 1 – Sumário dos trabalhos relacionados

Trabalho	Plataforma	Técnica	Conjunto de dados	Métricas
Ferramenta BigFlow (VIEGAS et al., 2019)	*Python*, *flowtbag*, *Apache Kafka* e *Apache Flink*	*Hoeffding Tree*, *OzaBoosting*, *Leveraging Bag* e comitê	*MAWILab*	Acurácia (geral e por classe), Taxa de bytes
Ferramenta CATRACA (LOPEZ, 2018)	*Virtual Network Function*, *Apache Kafka* e *Apache Spark*	PCA, SFS, e SVM-RFE	NSL-KDD, GTA/UFRJ e NetOp	Acurácia, precisão, sensibilidade e F1-score
Arquitetura IDSA-IoT (CASSALES et al., 2019)	Java, *Apache Kafka* e *Python*	ECSMiner, AnyNovel e MINAS	*Kyoto 2006+*	Fnew, Mnew e erro

3.1 Ferramenta BigFlow

Proposta por Viegas et al. (2019), a ferramenta BigFlow é um sistema de detecção de intrusão em rede (*Network Intrusion Detection System*, NIDS) baseado em detecção de anomalias. Duas abordagens, detecção por assinatura e detecção por anomalia, . Para a detecção de novos tipos de ataque (*zero day*), a abordagem de detecção por anomalia é vantajosa, em contraste com a abordagem de detecção por assinatura, devido ao tempo

de resposta (que envolve a identificação e criação de uma assinatura), grande demais para prevenir esse tipo de intrusão.

A ferramenta BigFlow é composta pelos módulos de extração de atributos e de aprendizado confiável. O módulo de extração de atributos é responsável por coletar da rede monitorada, com estatísticas de comunicação e enviar informações desses fluxos como exemplos para o módulo de aprendizado confiável. O módulo de aprendizado confiável, é composto pelos submódulos: submódulo classificador, responsável por classificar exemplos; submódulo de verificação, responsável por verificar o resultado de classificação; submódulo de exemplos rejeitados, responsável por requisitar a um especialista etiquetas para exemplos rejeitados e; submódulo de atualização incremental, que atualiza e distribui o modelo aos classificadores.

Viegas et al. (2019) destaca que *data sets* adequados para NIDS são poucos, devido ao conjunto de qualidades que os mesmos devem atender, como realismo, validade, etiquetamento, grande variabilidade e reprodutividade (disponibilidade pública).

Para avaliar o desempenho de NIDS, o *data set* MAWIFlow é proposto por Viegas et al. (2019). Este *data set* é derivado do *data set Packet traces from WIDE backbone, samplepoint-F*, composto por seções de captura de pacotes diárias de 15 minutos de um link de 1Gbps entre Japão e EUA, com início em 2006 continuamente até hoje, anonimizados e etiquetados por MAWILab (MAWI Working Group Traffic Archive, 2020; FONTUGNE et al., 2010). Desse *data set* original, o *data set* MAWIFlow utiliza apenas os eventos de 2016, dos quais 158 atributos são extraídos resultando em 7.9 TB de captura de pacotes. Além disso, os dados são estratificados para redução de seu tamanho a um centésimo, as proporções de etiquetas (Ataque e Normal), o compartilhamento e avaliação de NIDS, além de atender às qualidades anteriormente mencionadas.

Com o *data set* MAWIFlow reduzido a 62 atributos, foram avaliados quatro classificadores da literatura em dois modos de operação. O primeiro modo de operação usa somente a primeira semana do ano como conjunto de treinamento e as demais como conjunto teste. O segundo modo usa o conjunto da semana anterior como treinamento e o conjunto da semana seguinte como teste. Comparando os resultados entre os modos de operação, os autores demonstram que a qualidade da classificação reduz-se com o tempo, quando não há atualização frequente do modelo classificador.

Com base na avaliação dos classificadores da literatura, para a ferramenta BigFlow é proposta a utilização de 4 algoritmos de classificação com capacidade de atualização, sendo todos variações de árvore de decisão *Hoeffding* (VIEGAS et al., 2019; DOMINGOS; HULTEN, 2000). A avaliação da ferramenta foi executada de maneira semelhante à avaliação dos algoritmos da literatura, onde o conjunto de dados da primeira semana foi usado para treinamento e o conjunto de dados do restante do ano como conjunto de teste. Além do conjunto de treinamento, o modelo é atualizado semanalmente com base nas instâncias rejeitadas pelo submódulo de verificação.

Quanto à distribuição do processamento, a ferramenta BigFlow faz uso das plataformas *Apache Flink* e *Apache Kafka*. Em especial, destaca-se o uso do serviço gerenciador de trabalhos (*Job Manager*) e as múltiplas instâncias do serviço gerenciador de tarefas (*Task Manager*).

Em conclusão, a ferramenta BigFlow demonstra capacidade de classificação e detecção de anomalias em fluxos de dados de alta velocidade no contexto de detecção de intrusão.

3.2 Ferramenta CATRACA

O trabalho de Lopez (2018) aborda a detecção de ameaças a redes de computadores em tempo real e, para atingir esse objetivo, propôs a ferramenta CATRACA[1]. A ferramenta CATRACA é composta de três camadas: captura, processamento e visualização.

Na camada de captura, pacotes são capturados da rede e são geradas informações sumário de fluxos por uma aplicação *Python* utilizando a biblioteca *flowtbag*[2]. Esses sumários são enviados para um tópico de um sistema de fila de mensagens (*Apache Kafka*) hospedado em nuvem. Essa aplicação *Python* é distribuída como uma função virtual de rede (*Network Function Virtualization*) executada em dispositivos de rede virtuais.

A camada de processamento consome o tópico de mensagens que contém os fluxos da camada de captura e extrai características dos fluxos, detecta e classifica ameaças, enriquece o resultado (com localização geográfica por exemplo) e envia para a próxima camada na arquitetura por meio de um banco de dados (SGBD). A última camada da ferramenta fornece uma interface gráfica que apresentada a visualização dos fluxos processados bem como os conhecimentos extraídos e armazenados no banco de dados (SGBD). Ambas as camadas de processamento e visualização são executadas em ambiente de computação em nuvem (*cloud computing*).

Para o desenvolvimento da ferramenta CATRACA, Lopez (2018) avaliou e comparou as plataformas de processamento de fluxo de dados em tempo real disponíveis (*Apache Storm*, *Apache Flink*, *Apache Spark Streaming*). A avaliação extraiu a velocidade máxima, em mensagens por minuto, de cada plataforma, variando a configuração de paralelismo em dois programas. Ambos consumiam dados de um tópico de um sistema de fila de mensagens (*Apache Kafka*) e produziam para outro tópico. O primeiro programa consiste de um detector de ameças composto por uma rede neural classificadora escrito em *Java*, que foi testado com o conjunto de dados sintético UFRJ/GTA (LOPEZ, 2018). O segundo programa conta quantas repetições de uma palavra existem em um fluxo de dados, exem-

[1] A ferramenta e sua documentação estão disponíveis em <http://gta.ufrj.br/catraca> e <https://github.com/tinchoa/catraca>.

[2] Disponível em <https://github.com/danielarndt/flowtbag> e <https://dan.arndt.ca/projects/netmate-flowcalc/>.

plo muito comum em tutoriais de plataformas desse gênero, e é avaliado com um conjunto de *Tweets*.

Para o modelo de classificação, a ferramenta CATRACA utiliza o método árvore de decisão, escolhido pelo rápido treinamento e pela alta precisão e acurácia[3]. O modelo é criado na fase *Offline* e utilizado na classificação binária (normal e ameaça) da fase *Online*, sendo recalculado quando uma ameaça é encontrada.

Pra avaliação da ferramenta CATRACA dois conjuntos de dados são utilizados. O primeiro conjunto, UFRJ/GTA, é sintético e foi criado por uma simulação de rede de computadores, contendo 214 200 fluxos de rede e totalizando 95GB de pacotes capturados, este *data set* é composto de 24 atributos e 16 classes. O outro conjunto, referido como NetOp, foi coletado de um operador de rede que atendia 373 residências na cidade do Rio de Janeiro em 2017. O conjunto NetOp é formado por 5 TB de pacotes capturados e etiquetados por um detector de intrusão comercial.

Também para a avaliação da ferramenta CATRACA, foram utilizadas as métricas de qualidade de classificação acurácia, precisão, sensibilidade e F1M, com intervalo de confiança de 95%. As métricas de qualidade, dependendo do tamanho do conjunto, foram extraídas por métodos de avaliação amplamente utilizados para avaliar modelos de aprendizado de máquina (*machine learning*) como validação cruzada com proporção 70% do conjunto base para treinamento e 30% para teste. Para as métricas de escalabilidade foram utilizadas a latência e fator de aceleração *speedup factor* (latência observada com paralelismo 1 dividida pela latência observada com paralelismo variável).

Em conclusão, a ferramenta CATRACA apresenta uma arquitetura dividida em camadas alocadas em ambientes de névoa (*fog computing*) e nuvem (*cloud computing*). Essa ferramenta foi avaliada com métricas de qualidade, métricas de escalabilidade e dois conjuntos de dados relevantes. No entanto, o algoritmo de detecção de anomalias desenvolvido para a ferramenta consiste de um modelo de classificação pelo método árvore de decisão e a atualização do modelo durante a fase *Online* depende de todos os exemplos do último intervalo de atualização. Esse tipo de algoritmo de detecção de anomalias de dados, como os descritos na Seção 2.4 (*Concept Drift*, *Concept Evolution*, limitado a ler o conjunto somente uma vez), que são atendidos por algoritmos de detecção de novidade.

3.3 Arquitetura IDSA-IoT

A arquitetura IDSA-IoT, proposta por Cassales et al. (2019), tem por objetivo monitorar uma rede local com dispositivos IoT e detectar tentativas de intrusão e alguma subversão do comportamento das transmissões destes dispositivos. O principal destaque da arquitetura é a distribuição de tarefas do sistema de detecção de intrusão entre nós na

[3] A precisão e a acurácia do método árvore de decisão podem estar associadas à independência entre as características (*features*) de cada exemplo, típico de conjuntos derivados de pacotes de rede.

e nós em nuvem pública (*cloud computing*). O objetivo dessa distribuição é a redução de latência, que torna inviável a hospedagem de um sistema detector de intrusão somente em ambiente *cloud computing*, e também possibilitar a análise de grandes volumes de dados por algoritmos de maior complexidade, que são de custo computacional proibitivo para nós de borda. A Figura 1 ilustra a estrutura física da arquitetura IDSA-IoT, destacando os dispositivos IoT, dispositivos de borda e nuvem pública.

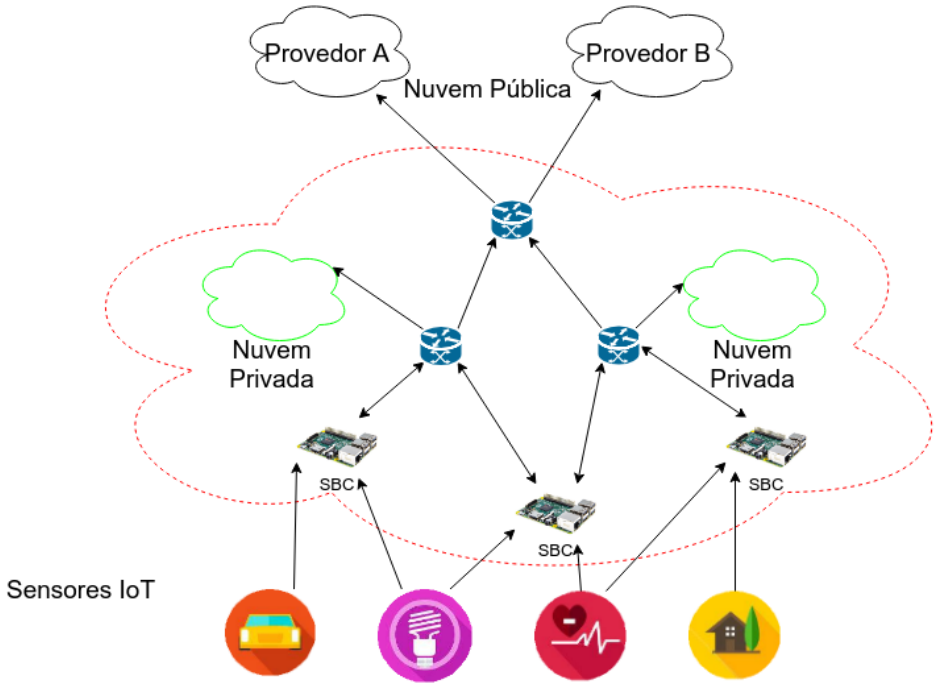

Figura 1 – Estrutura Física da Arquitetura IDSA-IoT. Produzida e traduzida por Cassales et al. (2019).

A arquitetura proposta é avaliada com três algoritmos de detecção de novidade: ECS-Miner (MASUD et al., 2011), AnyNovel (ABDALLAH et al., 2016) e MINAS (FARIA; CARVALHO; GAMA, 2015). A avaliação foi feita com o *data set Kyoto 2006+*, composto de dados coletados de 348 *Honeypots* (máquinas isoladas, equipadas com diversos softwares com vulnerabilidades conhecidas e expostas à Internet, com propósito de atrair ataques) de 2006 até dezembro 2015. Esse *data set* tem as características desejáveis de um conjunto para detectção de novidades como: realismo, validade, etiquetas previamente definidas, alta variabilidade, reprodutibilidade e disponibilidade pública. O *data set Kyoto 2006+* contém 24 atributos, 3 etiquetas atribuídas por detectores de intrusão comerciais e uma etiqueta distinguindo o tráfego entre normal, ataque conhecido e ataque desconhecido.

A avaliação da arquitetura foi realizada utilizando as métricas de qualidade Fnew, Mnew e erro. A métrica Fnew (ou Falso Positivo) é a fração dos exemplos de uma classe normal classificados com etiqueta novidade ou etiqueta extensão. A métrica Mnew (ou Falso Negativo) é a fração dos exemplos de uma classe novidade classificados com etiqueta normal. A métrica erro é a soma dos valores falso positivo e falso negativo dividida pelo

número de exemplos classificados. Além das métricas de qualidade de classificação tradicionais, também foi medida a quantidade de requisições de classificação por especialista.

Outra avaliação dos algoritmos foi a extração de métricas de uso de recursos computacionais e tempo total de processamento em dispositivos limitados. Essa avaliação envolveu dois computadores. Para tanto, um computador pessoal com recursos convencionais produzia exemplos e adicionava como mensagens em um tópico no sistema de fila de mensagens *Apache Kafka*; já o outro computador, com recursos limitados, consumia as mensagens do tópico e classificava os exemplos.

Ambas as avaliações demonstraram o equilíbrio entre qualidade de classificação e velocidade ou uso de recursos. O algoritmo ECSMiner mostrou melhor qualidade de classificação, porém com velocidade inferior e maior consumo de recursos comparado aos outros algoritmos. Já o algoritmo MINAS, apesar de maiores valores na métrica erro, mostrou-se adequado para dispositivos limitados com baixo consumo de recursos computacionais e manteve a métrica Fnew constante e baixa. O algoritmo AnyNovel não apresentou consistência nos resultados e o consumo de recursos computacionais (memória) foi elevado.

Com as avaliações realizadas, a arquitetura IDSA-IoT as tarefas de mineração dos fluxos para detecção de intrusão em serviços e aloca os serviços em diferentes camadas físicas, conforme ilustrado na Figura 2.

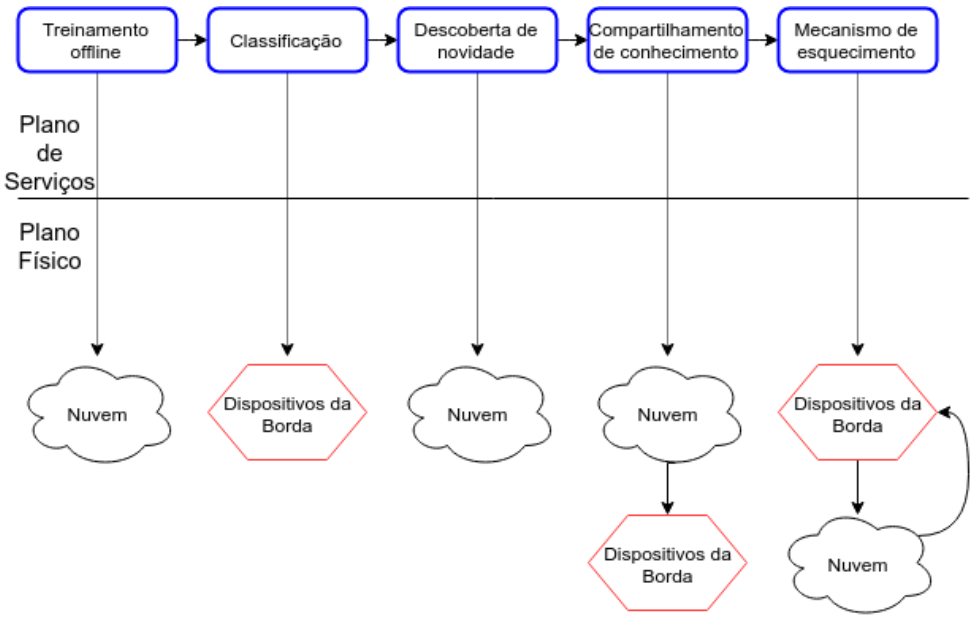

Figura 2 – Distribuição de Serviços da Arquitetura IDSA-IoT. Produzida e traduzida por Cassales et al. (2019).

A distribuição das tarefas em serviços proposta abre oportunidades para a discussão de diferentes métodos de distribuição dessas tarefas em diferentes ambientes computacionais. Contudo, o algoritmo MINAS ainda não foi implementado e avaliado com ou , que são necessários para tratar fluxos de dados com grandes volumes e velocidades.

3.4 Conclusão

Em conclusão, os trabalhos discutidos nesse Capítulo têm temas complementares em áreas distintas. A área de aprendizado de máquina, com o tema detecção de novidades em fluxos de dados, preocupa-se em fornecer melhores previsões através de algoritmos classificadores que atendam as características de cada problema. A área de computação distribuída aborda os temas de processamento distribuído de fluxos contínuos em ambientes de computação em nuvem e em névoa, fornecendo métodos para processar grandes volume de dados com mínima latência.

Apesar de já existirem propostas que estabelecem o estado da arte separadamente em cada um dos temas, entre o estado da arte em de novidade e o estado da arte em de fluxos de dados, em especial para focado em relacionados a

Capítulo 4

Considerações Finais

Este trabalho reúne conceitos de aprendizado de máquina com ênfase em detecção de novidades em fluxos contínuos de dados e conceitos de processamento distribuído de fluxos contínuos, com o objetivo de unir a lacuna no estado da arte desses conceitos à luz de uma implementação e avaliação no cenário de detecção de intrusão em redes de dispositivos da Internet das Coisas (IoT) em ambiente de computação em névoa (*fog computing*).

O objeto central desse trabalho (sistema M-FOG) trata da implementação do algoritmo MINAS na plataforma de processamento de fluxos *Apache Flink*, em três módulos que podem ser distribuídos em um ambiente de *fog computing*. Sua distribuição permite selecionar o nó que tem os recursos computacionais mais adequados para cada tarefa. A avaliação do sistema M-FOG será feita por meio de métricas de qualidade de classificação e métricas de escalabilidade.

Dando continuidade a este trabalho, segue-se com o desenvolvimento da implementação objeto (sistema M-FOG) bem como a contínua avaliação comparativa dos resultados produzidos pelo sistema M-FOG com seu algoritmo base, MINAS. Também será dada continuidade nos experimentos com os conjuntos de dados (*data sets*) diversos e configurações variadas de distribuição de processamento em *fog computing* extraindo desses experimentos as métricas previamente discutidas.

Dessa forma, o sistema M-FOG pode contribuir com adição de uma ferramenta para os interessados em sistemas de detecção de intrusão de redes de dispositivos IoT ou outros sistemas que tratam de fluxos contínuos que tradicionalmente sofrem com os ônus de latência e largura de banda na comunicação entre borda e nuvem. Além disso, o sistema M-FOG objetiva contribuir com a adição de uma implementação distribuída de um algoritmo cujo modelo é estado da arte em detecção de novidades em fluxos contínuos de dados.

Referências

ABDALLAH, A.; MAAROF, M. A.; ZAINAL, A. Fraud detection system: A survey. **Journal of Network and Computer Applications**, v. 68, p. 90 – 113, 2016. ISSN 1084-8045. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1084804516300571>.

ABDALLAH, Z. S. et al. Anynovel: detection of novel concepts in evolving data streams: An application for activity recognition. **Evolving Systems**, v. 7, n. 2, p. 73–93, 2016. ISSN 18686486.

AGGARWAL, C. C. et al. A framework for clustering evolving data streams. **Proceedings - 29th International Conference on Very Large Data Bases,VLDB 2003**, p. 81–92, 2003.

Apache Flink. **Apache Flink**. 2020. Disponível em: <https://flink.apache.org/>.

Apache Hadoop. **The Apache™ Hadoop® project develops open-source software for reliable,scalable,distributed computing.** 2020. Disponível em: <https://hadoop.apache.org/>.

Apache Spark. **Apache Spark™ - Unified Analytics Engine for Big Data**. 2020. Disponível em: <https://spark.apache.org/>.

BIFET, A. et al. MOA: massive online analysis. **J. Mach. Learn. Res.**, v. 11, p. 1601–1604, 2010. Disponível em: <http://portal.acm.org/citation.cfm?id=1859903>.

BONOMI, F. et al. Fog computing and its role in the internet of things. In: **Proceedings of the first edition of the MCC workshop on Mobile cloud computing**. [s.n.], 2012. p. 13–16. ISBN 9781450315197. Disponível em: <http://www.lispmob.org>.

CARBONE, P. et al. **Apache Flink™: Stream and Batch Processing in a Single Engine**. [S.l.], 2015. v. 36, n. 4. Disponível em: <http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-198940>.

CASSALES, G. W. et al. Idsa-iot: An intrusion detection system architecture for iot networks. In: **2019 IEEE Symposium on Computers and Communications (ISCC)**. [s.n.], 2019. p. 1–7. ISBN 978-1-7281-2999-0. ISSN 1530-1346. Disponível em: <https://ieeexplore.ieee.org/document/8969609/>.

COSTA, J. D. **Detecção De Novidade Em Fluxos Contínuos De Dados Multirrótulo**. 127 p. Tese (Master) — UNIVERSIDADE FEDERAL DE SãO CARLOS, 2019. Disponível em: <https://repositorio.ufscar.br/handle/ufscar/12197>.

COULL, S. et al. Intrusion detection: A bioinformatics approach. In: IEEE. **19th Annual Computer Security Applications Conference,2003. Proceedings.** [S.l.], 2003. p. 24–33.

Da Silva, T. P. et al. A fuzzy multiclass novelty detector for data streams. **IEEE International Conference on Fuzzy Systems**, IEEE, v. 2018-July, p. 1–8, 2018. ISSN 10987584.

DASTJERDI, A. V.; BUYYA, R. Fog computing: Helping the internet of things realize its potential. **Computer**, IEEE, v. 49, n. 8, p. 112–116, Aug 2016. ISSN 1558-0814.

DEAN, J.; GHEMAWAT, S. MapReduce: Simplified data processing on large clusters. **OSDI 2004 - 6th Symposium on Operating Systems Design and Implementation**, p. 137–149, 2004. ISSN 23487852.

DOMINGOS, P.; HULTEN, G. Mining high-speed data streams. In: **Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York,NY,USA: Association for Computing Machinery, 2000. (KDD '00), p. 71–80. ISBN 1581132336. Disponível em: <https://doi.org/10.1145/347090.347107>.

FARIA, E. R. et al. Evaluation methodology for multiclass novelty detection algorithms. **Proceedings - 2013 Brazilian Conference on Intelligent Systems,BRACIS 2013**, p. 19–25, 2013.

_____. Novelty detection in data streams. **Artificial Intelligence Review**, Springer, v. 45, n. 2, p. 235–269, Feb 2015. ISSN 1573-7462. Disponível em: <https://doi.org/10.1007/s10462-015-9444-8>.

FARIA, E. R. d.; CARVALHO, A. C. Ponce de L. F.; GAMA, J. Minas: multiclass learning algorithm for novelty detection in data streams. **Data Mining and Knowledge Discovery**, v. 30, n. 3, p. 640–680, May 2015. ISSN 1573-756X. Disponível em: <https://doi.org/10.1007/s10618-015-0433-y>.

FONTUGNE, R. et al. Mawilab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In: **ACM CoNEXT '10**. Philadelphia,PA: [s.n.], 2010. p. 1–12.

FOUNDATION, A. S. **Apache Storm**. 2020. Disponível em: <https://storm.apache.org/>.

GABER, M. M.; ZASLAVSKY, A.; KRISHNASWAMY, S. Mining data streams: A review. **SIGMOD Rec.**, Association for Computing Machinery, New York,NY,USA, v. 34, n. 2, p. 18–26, jun 2005. ISSN 0163-5808. Disponível em: <https://doi.org/10.1145/1083784.1083789>.

GAMA, J.; RODRIGUES, P. P. Data stream processing. In: _____. **Learning from Data Streams: Processing Techniques in Sensor Networks**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 25–39. ISBN 978-3-540-73679-0. Disponível em: <https://doi.org/10.1007/3-540-73679-4_3>.

_____. **Knowledge Discovery from Data Streams**. [S.l.]: Chapman and Hall/CRC, 2010. ISBN 9781439826119.

HAYAT, M. Z.; HASHEMI, M. R. A dct based approach for detecting novelty and concept drift in data streams. In: IEEE. **2010 International Conference of Soft Computing and Pattern Recognition,SoCPaR 2010**. [S.l.], 2010. p. 373–378. ISBN 9781424478958.

IEEE Communications Society. **IEEE Std 1934-2018: IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing.** IEEE, 2018. 176 p. ISBN 9781504450171. Disponível em: <https://ieeexplore.ieee.org/document/8423800>.

JÚNIOR, J. D. C. et al. Novelty detection for multi-label stream classification. In: IEEE. **2019 8th Brazilian Conference on Intelligent Systems (BRACIS)**. [S.l.], 2019. p. 144–149. ISBN 9781728142531.

KAMBOURAKIS, G.; KOLIAS, C.; STAVROU, A. The Mirai botnet and the IoT Zombie Armies. In: **MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)**. IEEE, 2017. v. 2017-Octob, p. 267–272. ISBN 978-1-5386-0595-0. Disponível em: <http://ieeexplore.ieee.org/document/8170867/>.

KREPS, J. **Questioning the Lambda Architecture – O'Reilly**. 2014. 10 p. Disponível em: <https://www.oreilly.com/radar/questioning-the-lambda-architecture/>.

LOPEZ, M. E. A. **A monitoring and threat detection system using stream processing as a virtual function for Big Data**. Tese (Theses) — Sorbonne Université ; Universidade federal do Rio de Janeiro, Jun 2018. Disponível em: <https://tel.archives-ouvertes.fr/tel-02111017>.

MARZ, N.; WARREN, J. **Big Data: Principles and best practices of scalable real-time data systems**. [S.l.]: New York; Manning Publications Co., 2015.

MASUD, M. et al. Classification and novel class detection in concept-drifting data streams under time constraints. **IEEE Trans. on Knowledge and Data Engineering**, IEEE, v. 23, n. 6, p. 859–874, June 2011. ISSN 1041-4347.

MAWI Working Group Traffic Archive. **Index of /mawi/samplepoint-F**. 2020. Disponível em: <http://mawi.wide.ad.jp/mawi/samplepoint-F/>.

MELL, P.; GRANCE, T. The NIST definition of cloud computing: Recommendations of the National Institute of Standards and Technology. In: NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. **Public Cloud Computing: Security and Privacy Guidelines**. National Institute of Standards and Technology, 2012. p. 97–101. ISBN 9781620819821. Disponível em: <http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf>.

PERNER, P. Concepts for novelty detection and handling based on a case-based reasoning process scheme. In: **Advances in Data Mining. Theoretical Aspects and Applications**. [S.l.]: Springer, 2007. p. 21–33. ISBN 978-3-540-73435-2.

_____. Concepts for novelty detection and handling based on a case-based reasoning process scheme. **Engineering Applications of Artificial Intelligence**, v. 22, n. 1, p. 86 – 91, 2009. ISSN 0952-1976. Disponível em: <http://www.sciencedirect.com/science/article/pii/S095219760800105X>.

SHI, W. et al. Edge computing: Vision and challenges. **IEEE Internet of Things Journal**, Institute of Electrical and Electronics Engineers Inc., v. 3, n. 5, p. 637–646, oct 2016. ISSN 23274662. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7488250>.

SILVA, T. P. da. **Abordagem Fuzzy para Detecção de Novidade em Fluxo Contínuo de Dados**. 89 p. Tese (Master) — Universidade Federal de São Carlos, 2018. Disponível em: <https://repositorio.ufscar.br/handle/ufscar/10544>.

SINGH, S.; MARKOU, M. An approach to novelty detection applied to the classification of image regions. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 16, n. 4, p. 396–407, 2004.

SOKOLOVA, M.; LAPALME, G. A systematic analysis of performance measures for classification tasks. **Inf. Process. Manage.**, Pergamon Press,Inc., USA, v. 45, n. 4, p. 427–437, jul. 2009. ISSN 0306-4573. Disponível em: <https://doi.org/10.1016/j.ipm.2009.03.002>.

SOUZA, V. M. et al. Data stream classification guided by clustering on nonstationary environments and extreme verification latency. In: SIAM. **Proceedings of the 2015 SIAM International Conference on Data Mining**. Society for Industrial and Applied Mathematics Publications, 2015. p. 873–881. ISBN 9781510811522. Disponível em: <https://doi.org/10.1137/1.9781611974010.98>.

SPINOSA, E. J.; CARVALHO, A. P. de Leon F. de; GAMA, J. a. Cluster-based novel concept detection in data streams applied to intrusion detection in computer networks. In: **Proceedings of the 2008 ACM Symposium on Applied Computing**. New York,NY,USA: Association for Computing Machinery, 2008. (SAC '08), p. 976–980. ISBN 9781595937537. Disponível em: <https://doi.org/10.1145/1363686.1363912>.

_____. Novelty detection with application to data streams. **Intell. Data Anal.**, IOS Press, NLD, v. 13, n. 3, p. 405–422, ago. 2009. ISSN 1088-467X.

TANENBAUM, A. S.; STEEN, M. V. **Distributed systems: principles and paradigms**. [S.l.]: Maarten van Steen, 2018. ISBN 978-90-815406-2-9.

VALLIM, R. M. et al. Online behavior change detection in computer games. **Expert Systems with Applications**, v. 40, n. 16, p. 6258 – 6265, 2013. ISSN 0957-4174. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0957417413003576>.

VIEGAS, E. et al. Bigflow: Real-time and reliable anomaly-based intrusion detection for high-speed networks. **Future Generation Computer Systems**, Elsevier, v. 93, p. 473 – 485, 2019. ISSN 0167-739X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X18307635>.

VIJAYA, P.; MURTY, M. N.; SUBRAMANIAN, D. Leaders–subleaders: An efficient hierarchical clustering algorithm for large data sets. **Pattern Recognition Letters**, Elsevier, v. 25, n. 4, p. 505 – 513, mar 2004. ISSN 0167-8655. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167865503002824>.

WANG, H. et al. Mining concept-drifting data streams using ensemble classifiers. In: **Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York,NY,USA: Association for Computing Machinery, 2003. (KDD '03), p. 226–235. ISBN 1581137370. Disponível em: <https://doi.org/10.1145/956750.956778>.

ZAHARIA, M. et al. Apache spark: A unified engine for big data processing. **Communications of the ACM**, v. 59, p. 56–65, 11 2016.

_____. Apache spark: A unified engine for big data processing. **Communications of the ACM**, v. 59, n. 11, p. 56–65, 2016. ISSN 15577317.

ZHANG, J. et al. Novel fault class detection based on novelty detection methods. In: _____. **Intelligent Computing in Signal Processing and Pattern Recognition: International Conference on Intelligent Computing,ICIC 2006 Kunming,China,August 16–19,2006**. Berlin,Heidelberg: Springer Berlin Heidelberg, 2006. p. 982–987. ISBN 978-3-540-37258-5. Disponível em: <https://doi.org/10.1007/978-3-540-37258-5_124>.