UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Adonai Gabriel Loreto Peres Gonçalves

UTILIZAÇÃO DE ALGORITMOS DE AGRUPAMENTO PARA DETECÇÃO DE ANOMALIAS EM REDES DE COMPUTADORES

496
Santa Maria, RS
2021

Adonai Gabriel Loreto Peres Gonçalves

UTILIZAÇÃO DE ALGORITMOS DE AGRUPAMENTO PARA DETECÇÃO DE ANOMALIAS EM REDES DE COMPUTADORES

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da Computação**. Defesa realizada por videoconferência.

ORIENTADOR: Prof. João Vicente Ferreira Lima

496
Santa Maria, RS
2021

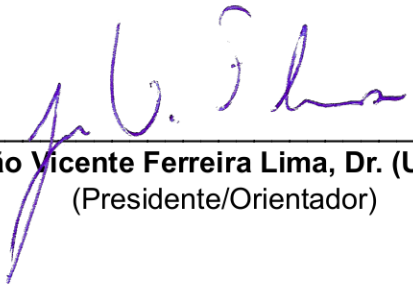
End. Eletr.: agoncalves@inf.ufsm.br

Adonai Gabriel Loreto Peres Goncalves

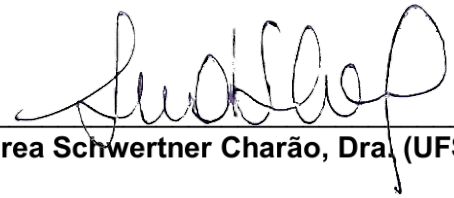
UTILIZAÇÃO DE ALGORITMOS DE AGRUPAMENTO PARA DETECÇÃO DE ANOMALIAS EM REDES DE COMPUTADORES

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da Computação**.

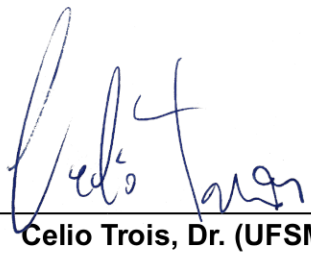
Aprovado em 2 de Setembro de 2021:



João Vicente Ferreira Lima, Dr. (UFSM)
(Presidente/Orientador)



Andrea Schwertner Charão, Dra. (UFSM)



Celio Trois, Dr. (UFSM)

Santa Maria, RS
2021

RESUMO

UTILIZAÇÃO DE ALGORITMOS DE AGRUPAMENTO PARA DETECÇÃO DE ANOMALIAS EM REDES DE COMPUTADORES

AUTOR: Adonai Gabriel Loreto Peres Gonçalves
ORIENTADOR: João Vicente Ferreira Lima

Anomalias em redes de computadores são desvios súbitos e inesperados que se repetem frequentemente em tráfego de dados. Elas podem indicar um pico de usuários, um defeito no sistema ou também um ciberataque. Um dos métodos para detectar essas anomalias é a utilização de algoritmos de agrupamento. Estes algoritmos visam agrupar um conjunto de dados de modo que cada grupo seja distinguível em relação aos outros. Com o intuito de mitigar os efeitos de ataques de agentes maliciosos e melhor compreender o processo de detecção de anomalias de rede, o presente trabalho apresenta um estudo e comparação de três algoritmos de agrupamento (k-Means, MCL e k-Shape) na análise de arquivos de *log* para detecção e classificação de anomalias, ramo da análise de rede em que os algoritmos MCL e k-Shape ainda não haviam sido utilizados. Após o treinamento e seleção dos melhores hiperparâmetros, concluiu-se que, considerando as três implementações utilizadas, apesar do algoritmo MCL ter obtido o melhor resultado em detecção de eventos benignos e o k-Shape ter obtido a melhor acurácia, o algoritmo k-Means é a melhor opção, pois obteve uma acurácia similar ao k-Shape e um tempo de execução mais de dez vezes menor.

Palavras-chave: Algoritmos de agrupamento. Anomalias de rede. Tráfego de dados. Redes de computadores. Análise de arquivos de log.

ABSTRACT

USING CLUSTERING ALGORITHMS TO DETECT ANOMALIES IN COMPUTER NETWORKS

AUTHOR: Adonai Gabriel Loreto Peres Gonçalves
ADVISOR: João Vicente Ferreira Lima

Network anomalies are frequent, unexpected and sudden deviations in data traffic. They may indicate a user spike, a system malfunction, or a cyberattack. One of the methods for anomaly detection is the use of clustering algorithms. These algorithms aim to group a dataset so that each cluster is distinguishable in relation to the others. With the intent of mitigating malicious agents' attacks and better comprehending the network anomaly detection process, this paper presents a study of three clustering algorithms (k-Means, MCL and k-Shape), being used for detecting and classifying the anomalies, a network analysis branch in which the MCL and k-Shape algorithms have not been used before. After training and selecting the best hyperparameters, it was concluded that, considering the three implementations used, despite the MCL algorithm having obtained the best result in detecting benign events and the k-Shape having obtained the best accuracy, the k-Means algorithm is the best option, as it achieved an accuracy similar to k-Shape and a runtime more than ten times shorter.

Keywords: Clustering algorithms. Network anomalies. Data traffic. Computer networks. Log file analysis.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

CPU	Central Processing Unit (Unidade Central de Processamento)
DL	Deep Learning (Aprendizado Profundo)
DoS	Denial of Service (Negação de Serviço)
$DDoS$	Distributed Denial of Service (Negação de Serviço Distribuído)
GPU	Graphics Processing Unit (Unidade de Processamento Gráfico)
IA	Inteligência Artificial
IDS	Intrusion detection System (Sistemas de Detecção de Intrusão)
IP	Internet Protocol
MCL	Markov Cluster Algorithm
ML	Machine Learning (Aprendizado de Máquina)
MLN	Machine Learning for Networking (Aprendizado de Máquina para Rede)

SUMÁRIO

1 INTRODUÇÃO

Na última década, a quantidade de dados sendo criados aumentou significativamente. Mais de 30.000 gigabytes de dados são gerados a cada segundo e a taxa de criação de dados está acelerando rapidamente (MARZ; WARREN, 2015). São diversos os tipos de dados sendo gerados, onde dados de tráfego de rede, especificamente arquivos de *log*, são o foco deste trabalho. Essa quantidade massiva de dados trafegando na rede torna essencial a criação de softwares inteligentes para analisar esses dados, a fim de identificar anomalias.

Essas anomalias podem ser geradas por um simples pico de requisições de usuários, mas também por ciberataques de agentes maliciosos. Entre os ciberataques, um dos mais populares atualmente são os ataques distribuídos de negação de serviço (DDOS), que é tipicamente realizado através do sobrecarregamento da máquina alvo por requisições supérfluas de várias origens, impedindo requisições legítimas. Existem vários outros tipos de ataques, como *Botnet*, *força bruta*, *heartbleed*, infiltração interna, entre outros (SHARAFALDIN; LASHKARI; GHORBANI, 2018).

Com o objetivo de identificar e classificar esses ataques, utiliza-se algoritmos de agrupamento, aliados à redes neurais, na análise de arquivos de *log* em fluxo de rede. Algoritmos de agrupamento, como o nome sugere, permitem o processamento e seleção de dados, de acordo com suas características ou similaridades a fim de criar grupos específicos (DENYSENKO, 2015).

Continuando as investigações e utilizando uma metodologia semelhante a Wiethan (2019), o presente trabalho[1] tem o propósito de apresentar diferentes soluções para detecção e classificação de anomalias de rede a partir da realização de um estudo e comparação de três algoritmos de agrupamento na análise de arquivos de *log*, ramo da análise de rede em que dois deles ainda não obtiveram representação, disponibilizados e usados como referência para este tipo de trabalho.

Para a configuração e seleção dos melhores hiperparâmetros, os três algoritmos passaram por algumas etapas no desenvolvimento do modelo. Onde foi feita a formulação do problema, caracterização da base de dados, análise dos dados, contrução do modelo, validação do modelo e comparação dos algoritmos. Onde o algoritmo k-Means se destacou no tempo de execução, principalmente pelo uso de paralelismo, e detecção de ataques DDOS. O MCL obteve o melhor resultado em detecção de eventos benignos e o k-Shape obteve a melhor acurácia.

Este trabalho está disposto da seguinte forma. O Capítulo 2, apresenta estudos sobre os conceitos de anomalias em redes, citando exemplos de anomalias e métodos para sua detecção. Além disso, apresenta alguns algoritmos de agrupamento que foram

[1] https://github.com/adonaigoncalves/TCC-Adonai

utilizados na detecção de anomalias e, finalmente, faz menção a trabalhos relacionados. O Capítulo 3, esclarece toda a metodologia aplicada no desenvolvimento do sistema de detecção de anomalias com a utilização dos algoritmos de agrupamento selecionados. No Capítulo 4, foram tratados os resultados obtidos por meio da comparação dos algoritmos de agrupamento escolhidos. Por fim, o Capítulo 5, contém a discussão e considerações finais acerca do que foi desenvolvido no trabalho.

2 ANOMALIAS E AGRUPAMENTOS

Este capítulo apresenta as noções básicas sobre anomalias em redes como também estratégias utilizadas para a sua detecção. Serão descritas definições, tipos de anomalias e os três algoritmos de agrupamento, selecionados pelos diferentes propósitos em que foram criados e métodos de classificação, para detecção de anomalias em tráfego de rede. Por fim, serão apresentadas diversas soluções para classificação de tráfego de rede e detecção de anomalias propostas em trabalhos relacionados.

2.1 ANOMALIAS EM TRÁFEGO DE REDE

As redes de computadores apresentam características em seu fluxo de dados que geram de certa forma uma padronização de seu tráfego, algo que pode ser denominado de perfil da rede (SHON; MOON, 2007). Anomalias em redes são marcadas por desvios repentinos e acentuados causados no tráfego em decorrência de inúmeras circunstâncias tais como: uso exorbitante de recursos da rede, erros em equipamentos, problemas com *hardware* ou *software* e ataques maliciosos. Este amplo conjunto de circunstâncias acabam tornando complexa a tarefa de desenvolver técnicas para detectá-las. Estas anomalias causam inúmeros empecilhos nas redes tais como: problemas de confiabilidade, falta de segurança e instabilidade da rede. Para identificar estas anomalias, podem ser feitas detecções através das alterações no padrão do tráfego em relação ao perfil da rede (CHANDOLA; BANERJEE; KUMAR, 2009).

2.1.1 Ataques Responsáveis por Anomalias

As anomalias podem apresentar diversas causas e consequências, que dificultam a identificação e resolução dos problemas gerados por elas. De acordo com Thottan e Ji (2003) as anomalias podem ser ocasionadas por diferentes problemas, que levam à classificação delas em dois grupos. O primeiro agrupa as anomalias onde não há a presença de agentes maliciosos. O segundo são as anomalias geradas por ataques, comumente criadas por agentes que pretendem romper as barreiras de segurança da rede.

Como exemplos de causas de anomalias pertencentes ao primeiro grupo, temos:

- Pico de usuários: uma grande quantidade de clientes passa, subitamente, de maneira não orquestrada e não maliciosa, a enviar requisições a um servidor, podendo fazer com que ele chegue a interromper as suas operações (ZARPELãO, 2010).

- Nós instáveis: um nó da rede entra em estado de falha por tempo indefinido, enviando pacotes aleatoriamente para vários pontos da rede. Um exemplo de anomalia do tipo *babbling node* ocorre quando uma placa de rede com problemas envia uma grande quantidade de pacotes de controle ou sinalização para toda a rede (AL-KASASSBEH; ADDA, 2009).

- Tempestade de *broadcasts*: uma enorme quantidade de pacotes de *broadcast* começa a trafegar pela rede, causando congestionamentos que podem levar à interrupção das operações da rede. Falhas em dispositivos iniciam o envio de pacotes de *broadcast* para outros ativos da rede, fazendo com que estes outros ativos também enviem *broadcasts* e formem uma reação em cadeia, causando tempestades de *broadcasts* (AL-KASASSBEH; ADDA, 2009).

- Erros em configurações: dispositivos como *firewalls*, se configurados incorretamente, podem barrar a entrada de uma grande quantidade de pacotes na rede, causando mudanças repentinas nos níveis de tráfego monitorados. Erros na configuração de servidores podem levá-los a não responder adequadamente às requisições enviadas pelos clientes, causando congestionamentos (THOTTAN; JI, 2003).

No segundo grupo de anomalias, temos os seguintes exemplos:

- Ataque de força bruta: um ataque de força bruta consiste em um agressor (agente malicioso) submetendo várias senhas na expectativa de adivinhar a combinação correta. Ele verifica sistematicamente todas as possíveis senhas até que a correta seja encontrada (KNUDSEN; ROBSHAW, 2011).

- *Heartbleed*: um bug na biblioteca de criptografia *OpenSSL*, que é uma implementação usada popularmente pelo protocolo *TLS* (Transport Layer Security). O *heartbleed* podia ser explorado independentemente se a instância do *OpenSSL* vulnerável estava sendo executada em um servidor *TLS* ou cliente. Foi resultado de uma validação de *input* (entrada) imprópria na implementação da extensão *heartbeat* do *TLS*. Uma situação onde mais dados podiam ser lidos do que deveria ser permitido, devido à uma falta de verificação de limite. O nome deste tipo de ataque é derivado dessa extensão (DURUMERIC et al., 2014).

- *Botnet*: uma *botnet* é um número de dispositivos conectados à Internet, cada um executando um ou mais *bots*. Onde *bot*, diminutivo de *robot* (robô), é um dispositivo infectado por *malware* que atende aos comandos do agente malicioso que o infectou. *Botnets* podem ser usadas para realizar ataque de negação de serviço distribuídos (DDOS), roubo de dados, envio de spam e permitir que o agressor acesse o dispositivo e suas conexões (FEILY; SHAHRESTANI; RAMADASS, 2009).

- Ataque de negação de serviço (*denial of service*, DoS): um ataque de negação de serviço acontece quando um agressor busca indisponibilizar o dispositivo ou recurso de rede aos seus pretendidos usuários por romper, temporariamente ou indefinidamente, os serviços de um *host* conectado à Internet. Negação de serviço é tipicamente realizado por inundar o dispositivo ou recurso alvo com requisições supérfluas, numa tentativa de sobrecarregar o sistema e impedir requisições legítimas de serem realizadas (COLE; KRUTZ; CONLEY, 2005).

- Ataque de negação de serviço distribuído (Distributed denial of service, DDoS): em um ataque de negação de serviço distribuído, a inundação de tráfego no dispositivo da vítima é originada de várias fontes diferentes. Isso torna efetivamente impossível parar o ataque por simplesmente bloquear uma única fonte (COLE; KRUTZ; CONLEY, 2005).

2.1.2 Anomalias e Segurança

A detecção de anomalias tem inúmeras aplicabilidades em âmbitos como detecção de intrusão para cibersegurança, vigilância militar para atividades inimigas e detecção de fraude para cartões de crédito (BHUYAN; BHATTACHARYYA; KALITA, 2014). No meio tecnológico em que vivemos, é imprescindível que haja segurança nas redes de computadores para preservar os dados tanto de instituições como de indivíduos. As redes de computadores se transformaram em um grande meio de transporte de informações e, devido a isso, temos uma quantidade massiva de dados que trafega por elas. Consequentemente, é inquestionável a existência de inúmeras informações de valor que despertam o interesse de agentes maliciosos. Assim, quando esses agentes mal intencionados adquirem esse acesso impróprio, podem fazer uso de informações da rede para prejudicar os usuários, administradores ou corporações.

Manter essas informações seguras é um processo complexo, com componentes tecnológicos e humanos, que envolve análises de metodologias e comportamentos. A confiabilidade de uma rede está diretamente associada à capacidade dela de lidar com a prevenção, detecção e recuperação de anomalias. Segue abaixo a explicação destas capacidades de acordo com Pinheiro (2007):

- Prevenção
  - Proteção de *hardware*: usualmente chamada de segurança física, nega acessos físicos não autorizados à infraestrutura da rede, tem como principal objetivo prevenir roubos de dados, desligamento de equipamentos e demais danos que podem vir a serem exercidos fisicamente no local;

  - Proteção de arquivos e dados: provida pelos serviços de autenticação, monitoramento de acesso e sistemas antivírus. Na operação de autenticação, é feita a análise da identidade do usuário, o monitoramento de acesso disponibiliza apenas os serviços indicados ao usuário e os sistemas antivírus asseguram a proteção do sistema contra agentes maliciosos;
  - Proteção de perímetro: ferramentas de *firewall* e roteadores se responsabilizam por esse aspecto, mantendo a rede segura contra tentativas de violação (interna e externa à rede).

- Detecção

  - Alertas: sistemas de detecção de intrusões (IDS) avisam os responsáveis pela segurança sobre possíveis sinais de invasão ou alteração suspeita no perfil da rede que possa revelar um padrão de ataque. Esses avisos podem ser emitidos via *e-mail*, mensagem no terminal de gerência, *SMS*, etc.;
  - Auditoria: regularmente é necessário ser feita uma análise dos componentes críticos do sistema por alterações suspeitas. Esse processo pode ser desempenhado por ferramentas que buscam, por exemplo, modificações no tamanho dos arquivos de senhas, usuários inativos, etc.

- Recuperação

  - Cópia de segurança dos dados (*Backup*): é cópia de segurança dos dados de um sistema, sendo necessário mantê-los sempre atualizados em mídia confiável e separados física e logicamente dos servidores;
  - Aplicativos de *Backup*: ferramentas que oferecem a restauração rápida e confiável dos dados atualizados em caso da perda das informações originais do sistema;
  - *Backup* do *Hardware*: a existência de *hardware* reserva, fornecimento autônomo de energia, linhas de dados excedentes, etc., podem ser justificados levando-se em conta o custo da indisponibilidade dos sistemas.

2.1.3 Sistemas de Detecção de intrusão

O principal foco deste trabalho é a detecção de anomalias, desta forma, os IDS atuam como recursos técnicos para realizar procedimentos na análise de rede com o propósito de detectar acessos não autorizados, que podem ter por objetivo a execução de ataques. De acordo com Ryza et al. (2017), embora seja bastante evidente quando um

sistema está sendo inundado de tráfego de rede, detectar um abuso pode ser tão difícil como procurar uma agulha em um imenso palheiro de solicitações de rede. Analisar essa grande concentração de dados que passa por uma rede, em tempo hábil para detecção de um ataque, é algo impossível para um ser humano. Devido a esses motivos, cientistas da tecnologia desenvolvem meios de detecção de intrusão fundamentados em variados métodos e algoritmos.

Os sistemas de detecção de intrusão segundo Alves et al. (2016) podem ser separados em dois grupos:

1. Detecção de intrusão baseada em assinatura: esses sistemas comparam o tráfego obtido com um banco de dados preexistente de padrões de ataque conhecidos (i.e. assinaturas). Os fornecedores desses sistemas atualizam ativamente suas listas de assinaturas (semelhante à programas antivírus).

2. Detecção de intrusão com base em anomalias: esses sistemas usam estatísticas para gerar uma linha de base das redes em diferentes períodos de tempo. Eles foram propostos para detectar ataques desconhecidos. Esses sistemas usam ML para criar um modelo que simula atividades regulares e, posteriormente, compara o novo comportamento com o modelo existente.

Os IDS identificam invasões em diversos lugares e podem ser categorizados com base em onde eles operam. Sistemas de detecção de intrusão baseado em rede (NIDS), são estrategicamente inseridos (em um ou múltiplos locais) para vigiar todo o tráfego da rede. Sistemas de detecção de intrusão baseado em hospedeiro (HIDS), analisam todos os dispositivos da rede conectados à Internet-intranet da organização, eles podem detectar tráfego malicioso gerado de dentro (por exemplo, quando um *malware* procura se espalhar para outros sistemas a partir de um *host* em uma organização).

Por fim, há também uma categorização dos IDS com base em suas ações. Ativo, também conhecido como sistema de detecção e prevenção de intrusões, cria alertas e registra entradas juntamente com comandos para modificar a configuração para proteger a rede. Passivo, somente detecta a atividade maliciosa e cria um alerta ou arquivo de *log*, mas não realiza nenhuma ação.

2.2 ALGORITMOS DE AGRUPAMENTO

Agrupamento é um método não supervisionado, que permite que os algoritmos processem e escolham os dados de acordo com suas características peculiares ou similaridades, a fim de criar grupos específicos (i.e. agrupamento). Para ser preciso, o objetivo dos algoritmos de agrupamento é criar grupos, com o mínimo de diferença entre os dados do mesmo agrupamento (DENYSENKO, 2015).

Selecionar um algoritmo de agrupamento apropriado para uma base de dados pode ser difícil, pois existem diferentes tipos de algoritmos que trabalham com diferentes tipos de dados. No entanto, selecionar os parâmetros adequados para cada algoritmo se mostrou o maior desafio. Alguns fatores importantes que afetam essa seleção incluem os aspectos do agrupamento, as características da base de dados, o número de desvios ou exceções, entre outros. No presente trabalho serão explorados três algoritmos de agrupamento: k-Means, MCL (Markov Cluster Algorithm) e k-Shape.

2.2.1 k-Means

O termo 'k-Means' foi utilizado pela primeira vez por James MacQueen em 1967 (MACQUEEN, 1967), apesar do algoritmo ter sido proposto por Stuart Lloyd em 1957 e publicado somente em 1982. E em 1965, Edward W. Forgy publicou essencialmente o mesmo método, e por isso este algoritmo é algumas vezes referido como o algoritmo de LloydForgy. O algoritmo k-Means, resumido em Algoritmo 1, é um algoritmo para segregar N dados em I espaços dimensionais dentro de K grupos (MACKAY, 2005). Cada grupo é associado a um ponto central (centroide) e os pontos mais próximos ou similares do centroide são atribuídos ao mesmo grupo.

O primeiro passo é a inicialização de K centroides. No artigo original (MACQUEEN, 1967) é recomendada a escolha desses pontos aleatoriamente. Mesmo assim, a escolha dos centroides iniciais pode influenciar o resultado dramaticamente, como ilustrado nas Figuras 2.1 e 2.2, podendo convergir em soluções locais otimizadas. Este algoritmo pode ser representado da seguinte forma:

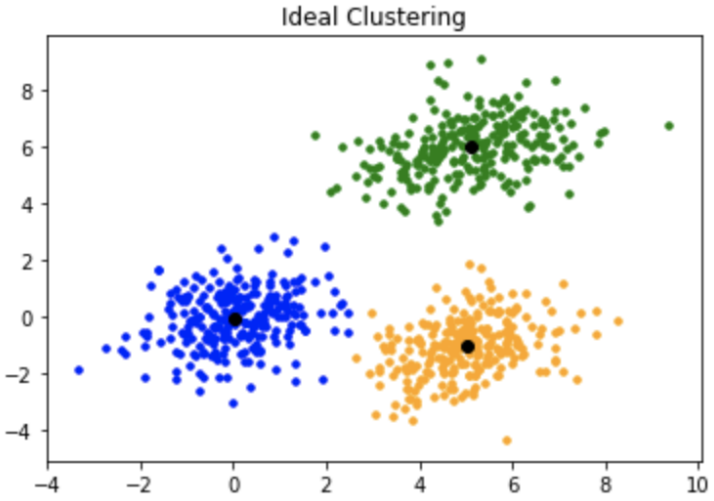
Algoritmo 1 Algoritmo K-Means

```
1: Especificar o número de K grupos
2: Inicializar aleatoriamente K centroides
3: repeat
4:     Atribuir cada ponto ao seu centroide mais próximo
5:     Computar o novo centroide (média) de cada grupo
6: until As posições dos centroides não mudarem
```

Figura 2.1 – Inicialização imprópria dos centroides.

Fonte: Adaptado de (savyakhosla, 2020)

Figura 2.2 – Inicialização ideal dos centroides.



Fonte: Adaptado de (savyakhosla, 2020)

2.2.2 MCL

MCL é um algoritmo de agrupamento não supervisionado rápido e escalável para gráfico, baseado na simulação de fluxo estocástico (processo aleatório, que acontece a partir de probabilidades) em gráfico, como ilustrado na Figura 2.3. O algoritmo foi desenvolvido por Stijn van Dongen no Centro de Matemática e Ciência da Computação na Holanda (Stijn van Dongen, 2000; DONGEN, 2000).

A ideia do algoritmo vem do princípio do passeio aleatório, que descreve um caminho que consiste de uma sucessão de passos aleatórios. Por exemplo, o caminho de um animal buscando alimento, o preço flutuante de ações, entre outros, mesmo que eles possam não ser verdadeiramente aleatórios na realidade. Passeios aleatórios em um gráfico são calculados usando Cadeias de Markov (Markov chains).

Uma Cadeia de Markov é uma modelo estocástico que descreve a sequência de possíveis eventos em que a probabilidade de cada evento depende somente do estado alcançado no evento anterior. Um exemplo de Cadeia de Markov seriam jogos de tabuleiro cujo movimento das peças de cada jogador depende somente de um lançamento de dados, como Banco Imobiliário, Imagem & Ação, entre outros.

Inicialmente, o MCL precisa de uma matriz de fluxo estocástico M construído usando uma função de distância arbitrária. Então são aplicadas as operações de expansão e inflação a cada iteração, até que não aconteça nenhuma mudança significativa nos valores da matriz (i.e. convergência). Para a implementação do Algoritmo MCL, resumido em Algoritmo 2, é necessário conhecer essas duas operações importantes:

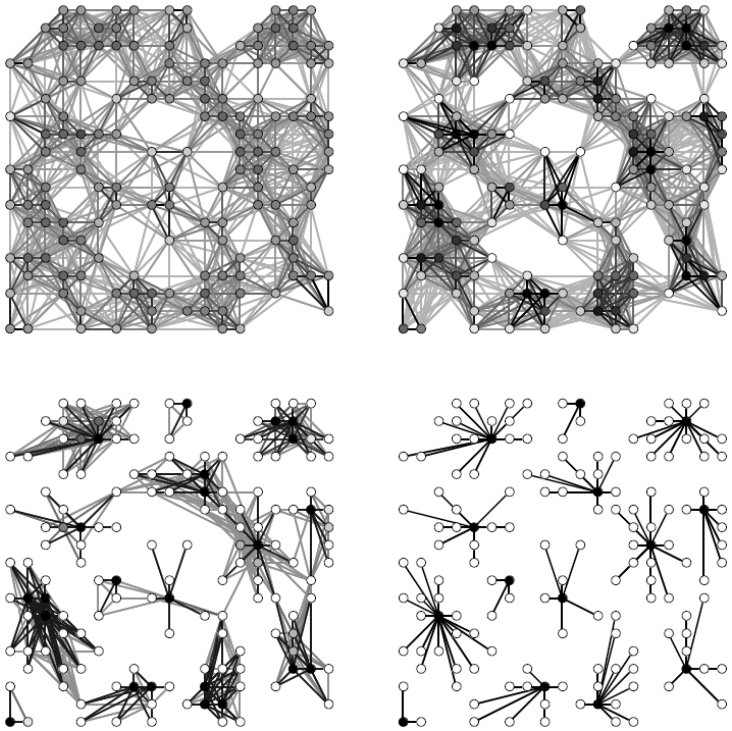
- Inflação: para cada vértice os valores de transição são mudados para que os valores altos vizinhos sejam ampliados e valores baixos sejam reduzidos. Isto pode ser alcançado por elevar os valores da coluna a uma potência não negativa e então re-normalizá-los. Este operador, conhecido como parâmetro *r*, é responsável por ampliar e reduzir o fluxo, influenciando o número de agrupamentos.

- Expansão: esta operação é alcançada por elevar a matriz M a um parâmetro *e*, ajudando a tornar os vértices vizinhos mais distantes alcançáveis. Este operador é responsável por permitir que o fluxo conecte as diferentes regiões do gráfico.

Algoritmo 2 Algoritmo MCL

```
1: Inicializar a matriz de transição M
2: repeat
3:     Expansão, utilizando o parâmetro e
4:     Inflação, utilizando o parâmetro r
5: until M convergir
```

Figura 2.3 – Sucessivos estágios de simulação de fluxo do processo MCL.



Fonte: Adaptado de (DONGEN, 2000)

2.2.3 k-Shape

k-Shape é um algoritmo, assim como o algoritmo k-Means mencionado anteriormente, baseado em centroides, desenvolvido para realizar o agrupamento de séries temporais, i.e. coleção de de observações feitas sequencialmente ao longo do tempo, por meio do algoritmo SBD (Shape-based Distance). SBD é baseado em relação cruzada, i.e. medida de similaridade entre dois sinais em função de um atraso aplicado a um deles, com normalização de coeficientes, que compara a similaridade de forma, ao invés de distância quantitativa, entre duas séries temporais (HE, 2018), como ilustrado na Figura 2.4.

O algoritmo k-Shape, resumido em Algoritmo 3, realiza dois passos a cada iteração (PAPARRIZOS; GRAVANO, 2016):

1. No passo de atribuição, o algoritmo atualiza as associações dos agrupamentos por comparar cada série temporal com todas os centroides computadas e atribuir cada série temporal ao agrupamento com o centroide mais próxima.

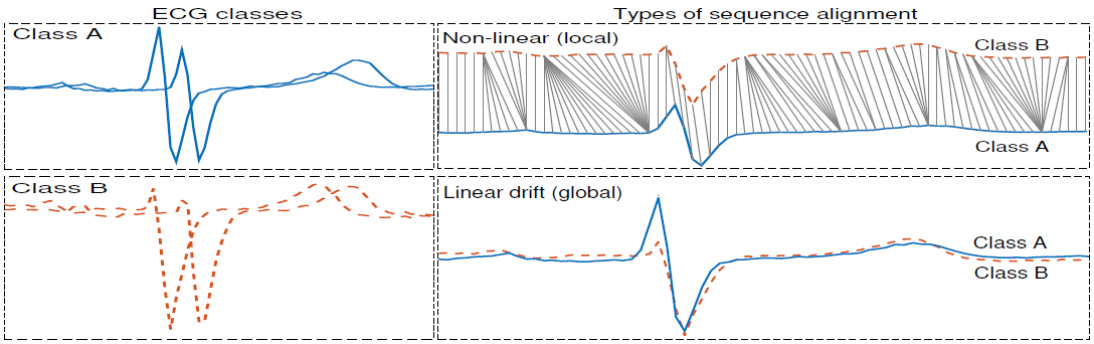
2. No passo de ajuste, os centroides dos agrupamentos são atualizados para refletir as mudanças nas associações dos agrupamentos do passo anterior.

O algoritmo repete esses dois passos até que não ocorra mudança nas associações dos agrupamentos (i.e. convergir) ou o número máximo de iterações permitidas seja alcançado. k-Shape espera como entrada a série temporal X e o número de agrupamentos k que se deseja produzir.

Algoritmo 3 Algoritmo k-Shape

1: Atribuição aleatória dos agrupamentos k na série temporal X
2: **repeat**
3: Ajuste, computando cada centroide de agrupamento
4: Atribuição, utilizando a função *SBD*
5: **until** Convergir ou o número de iterações alcance o máximo

Figura 2.4 – Sequências de eletrocardiograma e tipos de alinhamento das duas classes da base de dados de (CHEN et al., 2015)



Fonte: Adaptado de (PAPARRIZOS; GRAVANO, 2016)

2.3 TRABALHOS RELACIONADOS

Existem diversos trabalhos na área de análise de dados de rede, identificação de anomalias e detecção de ataques. Esta seção apresenta alguns desses trabalhos, considerando os mais relevantes para a realização deste trabalho. No segundo parágrafo, são apresentados os trabalhos que mais influenciaram na elaboração da metodologia deste trabalho. No terceiro parágrafo, são apresentados trabalhos que utilizaram o algoritmo k-Means para a análise e classificação do tráfego de rede. No quarto parágrafo, são apresentados os trabalhos que deram origem aos outros dois algoritmos utilizados no presente trabalho, MCL e k-Shape. E, por fim, no quinto parágrafo, são apresentados trabalhos que utilizaram a mesma base de dados (Canadian Institute for Cybersecurity, 2018) utilizada no presente trabalho para treinamento de técnicas de ML e algoritmos de agrupamento.

A metodologia empregada no presente trabalho tem como base o modelo descrito em Wang et al. (2017), onde é resumido um fluxo de trabalho básico para explicar como aplicar ML no domínio de redes. Então é fornecida uma pesquisa seletiva dos mais recentes avanços representativos de ML empregado em redes e uma descrição detalhada de como executar cada etapa do fluxo de trabalho. Utilizando a mesma metodologia, em Wiethan (2019) foi feito um estudo sobre detecção de anomalias de rede, através da implementação e análise de três modelos de classificadores baseados no aprendizado profundo, que foram testados em execuções de validação cruzada e em dados de uma base de dados paralela a base de seu treinamento. No tema de aprendizagem profunda, em Fadlullah et al. (2017) é abordado o acréscimo significante de pesquisas dessa questão em sistemas de tráfego de rede. É fornecida uma visão geral das arquiteturas e algoritmos de aprendizagem profunda de ponta relevantes ao controle de tráfego de rede. Além disso, é discutido, em detalhe, a ideia de roteamento inteligente baseado em aprendizagem profunda.

Um sistema de detecção de ataques DDoS é aplicado por Han et al. (2017), com uma abordagem original, baseado na entropia de informações (i.e. forma de medir o grau médio de incerteza) coletadas do tráfego de rede. Essa implementação utiliza o *framework Spark*, aplicando três variações do algoritmo K-Means implementados em seu trabalho. Seguindo nesta mesma linha, em Haas (2019) foi apresentado o desenvolvimento de um sistema baseado na arquitetura Lambda, utilizando o algoritmo de ML k-Means para classificação dos dados, capaz de processar e analisar o fluxo de dados do tráfego de rede, com o propósito de identificar anomalias de rede causadas por ataques de DDoS, força bruta e varredura de portas sobre determinados protocolos. Em Denysenko (2015) foi desenvolvido um sistema, com o auxílio dos algoritmos de agrupamento k-Means, k-Modes e k-Prototypes, para conduzir uma análise de agrupamentos para detectar possíveis anomalias em arquivos de *log* em servidor Apache.

Na tese de Dongen (2000) foi desenvolvido o algoritmo MCL, utilizado no presente trabalho, onde os tópicos principais foram a teoria matemática por trás do algoritmo, a sua posição na análise de agrupamentos e agrupamento de gráficos, questões relativas à escalabilidade, implementação e uma avaliação comparativa no geral. No trabalho de Paparrizos e Gravano (2016) é apresentado o algoritmo k-Shape, desenvolvido para realizar o agrupamento de séries temporais. Para demonstrar a robustez do k-Shape, foram executadas avaliações experimentais contra métodos de agrupamento particionais, hierárquicos e espectrais, os quais foram superados em termos de precisão.

Em Atefinia e Ahmadi (2020) foi usado um módulo agregador para integrar quatro arquiteturas de rede, os autores visaram obter uma maior taxa de precisão que qualquer outra proposta descrita nos trabalhos relacionados de seu artigo. Os modelos foram implementados em *Python* com auxílio da biblioteca *Scikit-learn* (PEDREGOSA et al., 2011) utilizando a base de dados Canadian Institute for Cybersecurity (2018). Utilizando a mesma base de dados, os autores de Basnet et al. (2019) conduziram experimentos com vários

frameworks de aprendizado profundo (como *Keras*, *PyTorch*, *TensorFlow*, entre outros) para detectar anomalias de tráfego de rede e classificar tipos de ataques.

Todos os trabalhos que foram citados durante esta seção, num total de dez, são apresentados no Quadro 2.1, na mesma ordem que foram mencionados no texto. Pode ser observado que a metodologia do presente trabalho é baseada nos três primeiros trabalhos. O três trabalhos seguintes forneceram informações sobre a utilização do algoritmo k-Means em detecção de anomalias de rede. O dois próximos trabalhos forneceram o embasamento teórico para a utilização dos algoritmos MCL e k-Shape. E os dois últimos trabalhos forneceram informações sobre a base de dados utilizada no presente trabalho, assim como um comparativo de desempenho entre os modelos desenvolvidos por eles e o presente trabalho. Portanto, nenhum dos trabalhos utilizou os algoritmos MCL e k-Shape para detecção de anomalias de rede.

Quadro 2.1 – Trabalhos relacionados.

Autor/Ano	**Metodologia**	**Atuação**
Wang et al. (2017)	Resume um fluxo de trabalho básico para explicar como aplicar ML no domínio de redes	Metodologia e fluxo de trabalho para ML
Wiethan (2019)	Realiza a detecção de anomalias de rede, através da implementação e análise de três modelos de classificadores baseados no aprendizado profundo	Detecção de anomalias por DL
Fadlullah et al. (2017)	Fornecido uma visão geral das arquiteturas e algoritmos de aprendiza-gem profunda de ponta relevantes ao controle de tráfego de rede	Tráfego de rede com DL
Han et al. (2017)	Analisa o fluxo de dados com base na entropia das informações, aplicando variações do algoritmo k-Means no processamento	Detecção de ataques DDoS
Haas (2019)	Desenvolve um sistema baseado na arquitetura Lambda, utilizando o algoritmo k-Means para classificação dos dados	Detecção de ataques DDOS, força bruta e *scanport*
Denysenko (2015)	Desenvolve um sistema, com o auxílio dos algoritmos de agrupamento k-Means, k-Modes e k-Prototypes, para detectar anomalias em arquivos de *log*	Detecção de anomalias com k-Means
Dongen (2000)	Apresenta o algoritmo MCL, explicando a teoria matemática por trás do algoritmo	Desenvolvimento de algoritmo de agrupamento
Paparrizos e Gravano (2016)	Apresenta o algoritmo k-Shape, desenvolvido para realizar o agrupamento de séries temporais	Desenvolvimento de algoritmo de agrupamento
Atefinia e Ahmadi (2020)	Integra quatro arquiteturas de rede, os autores visaram obter uma maior taxa de precisão que qualquer outra proposta	Detecção de anomalias com a mesma base de dados
Basnet et al. (2019)	Conduz experimentos com vários *frameworks* de aprendizado profundo (como *Keras*, *PyTorch*, *TensorFlow*, entre outros) para detectar anomalias de tráfego de rede	Detecção de anomalias com a mesma base de dados

Fonte: O autor.

2.4 SUMÁRIO

O presente capítulo detalhou alguns tópicos relacionados à anomalias de rede e métodos para a sua detecção. Após apresentar determinados tipos de ataques que causam anomalias no tráfego de rede e sistemas de detecção de intrusão, abordou-se a definição de algoritmo de agrupamento e quais deles foram usados nesta proposta. Por fim, descreveu-se alguns trabalhos relacionados que serviram de base para a metodologia do presente trabalho, que aplicaram diferentes métodos para a análise do tráfego de rede, inclusive métodos para detecção de anomalias que utilizaram a mesma base de dados desta proposta. Todavia, nenhum deles aplicou os algoritmos MCL e k-Shape para a classificação de anomalias de rede.

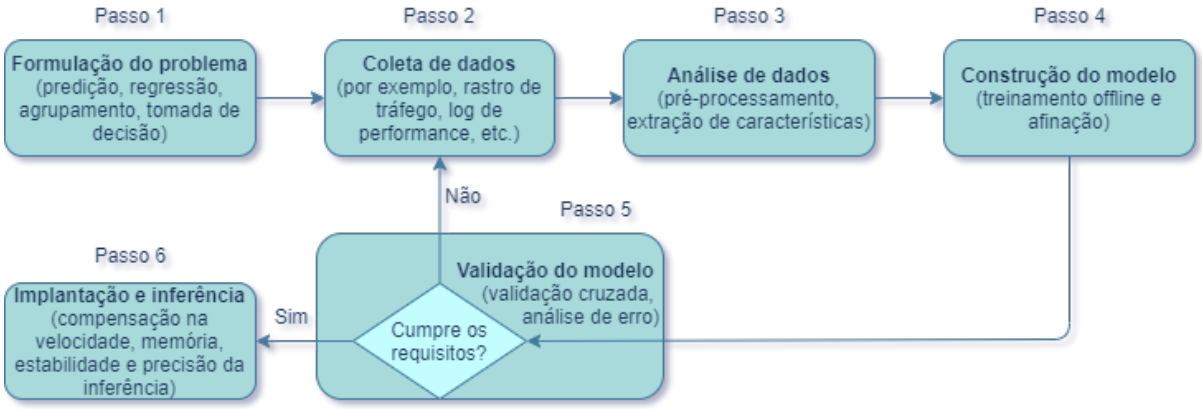
3 METODOLOGIA

Neste capítulo será descrito o processo de desenvolvimento do modelo de treinamento dos algoritmos de agrupamento, com uma descrição do fluxo de trabalho de aprendizado de máquina para rede (*Machine Learning for Networking*, MLN). Além disso, será feita a definição do problema, uma introdução a base de dados utilizada, explicação detalhada dos algoritmos de agrupamento selecionados, coleta e análise dos dados, treinamento e teste dos algoritmos e uma comparação entre os algoritmos.

3.1 FLUXO DE TRABALHO PARA MLN

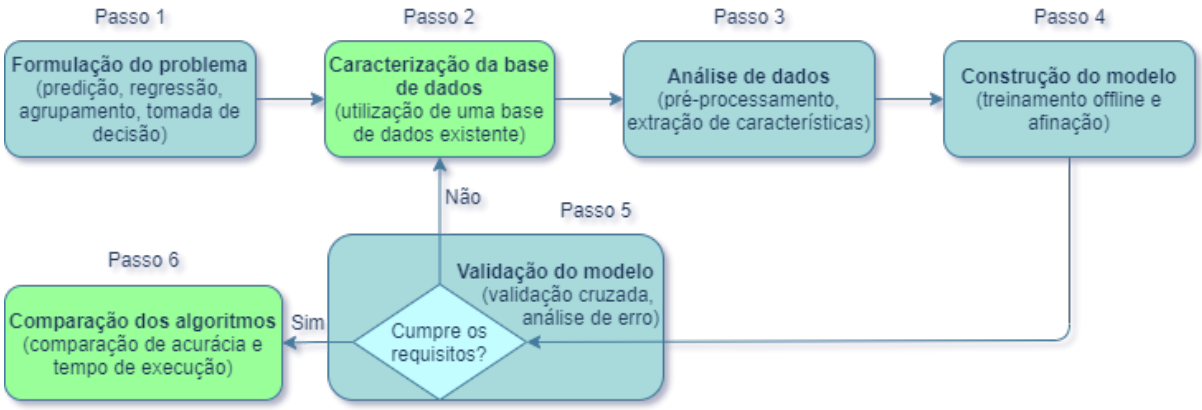
O fluxo de trabalho de ML para rede é descrito em Wang et al. (2017) em seis passos. No presente trabalho serão executados os cinco primeiros passos, como ilustrado na Figura 3.1, entretanto o segundo passo foi renomeado, visto que não será feita a coleta de dados mas a utilização de uma base de dados existente. Além disso, não será executado o passo de implantação e inferência, este será substituído por um passo de comparação dos algoritmos, como ilustrado na Figura 3.2. De acordo com esta descrição, cada passo deste fluxo de trabalho será explicado, à medida que é relacionado com os passos empregados no presente trabalho.

Figura 3.1 – Diagrama original dos passos do MLN.



Fonte: Adaptado de (WANG et al., 2017)

Figura 3.2 – Diagrama dos passos do MLN aplicado.



Fonte: Adaptado de (WANG et al., 2017)

3.1.1 Formulação do problema

Visto que o processo de treinamento de aprendizado de máquina é, muitas vezes, custoso, é importante abstrair e formular o problema no primeiro passo de MLN. Um problema-alvo pode ser classificado em uma das categorias de aprendizado de máquina, como classificação, agrupamento e tomada de decisão. Isso ajuda a decidir que tipo, a quantidade de dados a serem coletados e o modelo de aprendizado a ser escolhido. No presente trabalho, será feito uso do método de agrupamento, pois serão utilizados algoritmos de aprendizado não supervisionado, onde os algoritmos agrupam o conjunto de dados conforme suas semelhanças, de acordo com algum critério.

3.1.2 Caracterização da base de dados

Originalmente, o objetivo desse passo é coletar uma quantidade de dados representativos da rede, sem vieses. Os dados da rede (por exemplo, vestígio de tráfego e arquivos de *log*) são coletados a partir de diferentes camadas de rede, dependendo da necessidade da aplicação. No contexto de MLN, os dados são frequentemente coletados em duas fases. Na fase *offline*, coletar dados suficientes e selecionar corretamente os parâmetros é importante para a análise dos dados e treinamento do modelo. Na fase *online*, o estado da rede em tempo real e informações de desempenho são muitas vezes usados como entrada ou sinais de *feedback* para o modelo de aprendizagem.

Neste trabalho, a base de dados utilizada foi a CSE-CIC-IDS2018 (Canadian Institute for Cybersecurity, 2018), desenvolvida por uma parceria entre o *Communications*

Security Establishment (CSE) e o *Canadian Institute for Cybersecurity* (CIC) (SHARAFALDIN; LASHKARI; GHORBANI, 2018). De acordo com Leevy e Khoshgoftaar (2020), é a base de dados para detecção de intrusão em *Big Data*, disponível publicamente, mais recente e que cobre uma grande variedade de tipos de ataques. Nesta base de dados, foi utilizado o conceito de perfil para gerar os conjuntos de dados de maneira sistemática, os quais contêm descrições detalhadas de intrusões e modelos de distribuição abstratos para aplicações, protocolos, ou entidades de rede de baixo nível. Foram construídos duas classes distintas de perfil:

- Perfil-B: Encapsula o comportamento de entidade de usuários usando várias técnicas de aprendizado de máquina e análise estatística (como k-Means, Random Forest(BREIMAN, 2001), SVM(EVGENIOU; PONTIL, 2001) e J48(IHYA et al., 2019)). Os recursos encapsulados são repartições de tamanhos de pacote de um protocolo, número de pacotes por fluxo, certos padrões no conteúdo útil, tamanho do conteúdo útil e a distribuição de tempo de requisição de um protocolo.

- Perfil-M: Tenta descrever um cenário de ataque de uma maneira não ambígua. No caso mais simples, humanos podem interpretar esses perfis e subsequentemente executá-los. Idealisticamente, agentes autônomos junto com compiladores seriam empregados para interpretar e executar estes cenários.

No perfil-M, foram utilizadas as seguintes ferramentas para a realização dos ataques:

- Ataque de força bruta: *Patator*[1] foi a ferramenta utilizada para a realização destes ataques. Ela foi escrita em Python e, de acordo com Canadian Institute for Cybersecurity (2018), parece ser mais confiável e flexível que outras.

- *Heartbleed*: *Heartleech*[2] foi a ferramenta utilizada para a realização destes ataques. Para explorar essa vulnerabilidade, foi compilada a versão 1.0.1f da *OpenSSL*, que é uma versão vulnerável.

- *Botnet*: Nesta base de dados, foi utilizada a ferramenta *Zeus* e *Ares*, onde *Zeus*[3] é um pacote de cavalo de troia (*Trojan horse*) que pode ser executado em versões do Microsoft Windows. Ela é utilizada para roubo de informações bancárias, *keylogging* (registro das teclas pressionadas em um teclado) e instalação de *ransomware* (*malware* que restringe o acesso ao sistema infectado com uma espécie de bloqueio, cobrando resgate em criptomoedas). Já o *Ares*[4] é uma ferramenta de acesso remoto, que permite o acesso ao terminal do dispositivo, fazer captura de tela, upload e download de arquivos, *keylogging*, entre outros.

[1] https://github.com/lanjelot/patator
[2] https://github.com/robertdavidgraham/heartleech
[3] https://github.com/codingplanets/ZBOT-Botnet
[4] https://github.com/sweetsoftware/Ares

- Ataque de negação de serviço (*denial of service*, DoS): *Slowloris*[5] foi a ferramenta utilizada nesse tipo de ataque. Ela permite que uma única máquina derrube o servidor web de outra máquina com um mínimo de uso de banda e efeito colateral em serviços e portas não relacionados.
- Ataque de negação de serviço distribuído (*Distributed denial of service*, DDoS): O *High Orbit Ion Cannon* (HOIC)[6] e o *Low Orbit Ion Cannon* (LOIC)[7] foram as ferramentas utilizadas para realizar este tipo de ataque nesta base de dados. O LOIC é uma ferramenta de código aberto de teste de estresse de rede e ataque de negação de serviço, escrito em C#. Já o HOIC é uma ferramenta desenvolvida pelo grupo hacktivista *Anonymous* e projetada para substituir o LOIC, o HOIC é capaz de atacar até 256 URLs ao mesmo tempo.
- Ataques Web: a aplicação *Damn Vulnerable Web App* (DVWA)[8] foi utilizada nesta base de dados para realizar ataques web. A DVWA é uma aplicação web vulnerável criada para ajudar profissionais de segurança a testar suas habilidades e ferramentas, ajudar desenvolvedores web a entender melhor o processo de proteção de aplicações web e ajudar professores ou estudantes a ensinar ou aprender segurança de aplicações web em ambiente de sala de aula. A ferramenta possui vários tipos de ataque, como por exemplo, força bruta, injeção de comandos, injeção de SQL, *upload* de arquivos, inclusão de arquivos, entre outros, assim como vários níveis de dificuldade para cada tipo de ataque.
- Infiltração da rede por dentro: neste tipo de ataque, uma aplicação vulnerável deve ser explorada, como Adobe Acrobat Reader 9 e Dropbox. Primeiro a vítima recebe um documento malicioso por *e-mail*. Então, depois de uma exploração bem sucedida usando o *framework Metasploit*[9], uma ferramenta para realização de testes de invasão, um *backdoor*, i.e. um método para escapar uma autenticação ou criptografia em um sistema de um dispositivo, é executado no dispositivo da vítima. Nesse momento, é possível realizar diferentes ataques na rede da vítima (COLE; KRUTZ; CONLEY, 2005).

3.1.3 Análise de dados

Cada problema de rede tem as suas próprias características e é impactado por diversos fatores, porém somente alguns fatores têm um maior efeito nas métricas de desem-

[5] https://github.com/gkbrk/slowloris
[6] https://sourceforge.net/projects/high-orbit-ion-cannon/
[7] https://github.com/NewEraCracker/LOIC
[8] https://dvwa.co.uk/
[9] https://github.com/rapid7/metasploit-framework

penho da rede em questão. Este passo se empenha em extrair as características efetivas de um problema de rede através da análise do histórico de amostras de dados, os quais podem ser considerados como um processo de engenharia de recursos na comunidade de aprendizado de máquina.

A base de dados disponibilizada pelo Canadian Institute for Cybersecurity (2018) e utilizada no presente trabalho foi organizada em dias. Para cada dia, foram registrados dados brutos incluindo o tráfego de rede (Pcaps) e *logs* de evento (*Logs* de evento do Windows e Ubuntu) por máquina. No processo de extração de características dos dados brutos, foi utilizado o CICFlowMeter-V3 e extraídas mais de 80 características de tráfego e guardadas em um arquivo CSV (valores separados por vírgulas) por máquina (Canadian Institute for Cybersecurity, 2018). No presente trabalho, foram selecionados três dos dez dias de ataques realizados pelo Canadian Institute for Cybersecurity (2018), i.e. 14, 15 e 21 de fevereiro de 2018, para a análise e classificação de anomalias de rede. As classes de eventos selecionadas são descritas na Seção 3.2.

De acordo com Sharafaldin, Lashkari e Ghorbani (2018), a classe *RandomForestRegressor*[10] do pacote *Scikit-learn* (PEDREGOSA et al., 2011) pode ser utilizada para seleção dos melhores atributos. Primeiro é calculada a importância de cada atributo em toda a base de dados, então o resultado final é alcançado pela multiplicação da média padronizada de cada valor de cada atributo com o seu valor de importância correspondente. Este processo foi aplicada no presente trabalho e, após a sua aplicação, foram selecionados os atributos mais significativos para cada classe de evento da base de dados, o que resultou em 18 atributos que são descritos no Quadro 3.3.

Como o número de amostras de cada classe de evento é variado, ilustrado no Quadro 3.2, e para que os algoritmos de agrupamento não considerem as classes com maior número de amostras como mais importantes, foi necessário balancear essas quantidades. Foram utilizadas as classes *RandomUnderSampler*[11] e *RandomOverSampler*[12] do pacote *imblearn*[13] para realizar este processo. A classe *RandomUnderSampler* reduz o número de amostras de acordo com alguma estratégia, neste caso foi utilizada a estratégia de redução das amostras majoritárias. Já a classe *RandomOverSampler* amplia o número de amostras de acordo com alguma estratégia, e neste caso foi utilizada a estratégia de redução das amostras minoritárias. Após o balanceamento, cada classe de evento teve seu número de registros transformado para 41508, totalizando 290556 registros em relação aos 3145725 registros ilustrados no Quadro 3.2.

Para que não ocorra *overfitting* foi utilizado o método *random.shuffle*[14] do pacote *numpy*[15] para embaralhar as amostras balanceadas. Para deixar a aleatoriedade do emba-

[10] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
[11] https://imbalanced-learn.org/dev/references/generated/imblearn.under_sampling.RandomUnderSampler.html
[12] https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.RandomOverSampler.html
[13] https://imbalanced-learn.org/dev/references/index.html
[14] https://numpy.org/doc/stable/reference/random/generated/numpy.random.shuffle.html
[15] https://numpy.org/doc/stable/reference/index.html

ralhamento determinística, foi utilizada uma *seed* constante, deixando o embaralhamento constante. E, para realizar a padronização dos valores das amostras, foi utilizada a classe *StandardScaler*[16] do pacote *Scikit-learn* (PEDREGOSA et al., 2011), que substitui cada valor de cada atributo pela subtração do valor pela média dos valores do atributo, dividido pelo desvio padrão dos valores do atributo. Esse processo reduz a variância entre os valores de uma mesma classe, deixando as amostras numa mesma escala e mais parecidos com valores normalmente distribuídos.

3.1.4 Construção do modelo

A construção do modelo envolve a seleção do modelo, treinamento e ajuste. Um modelo ou algoritmo de aprendizado adequado precisa ser escolhido de acordo com o tamanho da base de dados, características típicas do cenário de rede, categoria do problema, entre outros. Então, o histórico de dados será utilizado no treinamento do modelo com ajuste de hiperparâmetro, o que levará um longo período de tempo na fase *offline*.

No presente trabalho[17] serão utilizados três algoritmos de agrupamento (k-Means, MCL e k-Shape) para classificar os parâmetros selecionados dos arquivos CSV e então serão ajustados conforme as características de cada algoritmo e tipos de agrupamentos. Foram utilizadas implementações desses algoritmos disponíveis na *internet*. Onde a implementação do algoritmo k-Means[18] está disponível no pacote *Scikit-learn* (PEDREGOSA et al., 2011), do algoritmo k-Shape[19] no pacote *tslearn* (TAVENARD et al., 2020) e do algoritmo MCL[20] em um repositório do *Github* (GuyAllard, 2017). Abaixo segue uma breve descrição, com os valores ilustrados no Quadro 3.1, de alguns dos hiperparâmetros utilizados durante a configuração dos algoritmos utilizados neste estudo:

- k-Means:
  - *n_clusters*: Controla o número de agrupamentos a serem formados, assim como o número de centroides a serem gerados.
  - *init*: Método para a inicialização dos centroides. Onde é possível selecionar entre *'k-means++'*, que seleciona os centroides de maneira inteligente para acelerar a convergência, *'random'*, que seleciona os centroides de maneira aleatória, ou selecionar manualmente os centroides;
  - *max_iter*: Número máximo de iterações para cada execução;

[16] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
[17] https://github.com/adonaigoncalves/TCC-Adonai
[18] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
[19] https://tslearn.readthedocs.io/en/stable/gen_modules/clustering/tslearn.clustering.KShape.html
[20] https://github.com/guyallard/markov_clustering

  - *random_state*: Define a geração dos números aleatórios para a inicialização dos centroides. Pode ser usado um número inteiro para deixar a aleatoriedade determinística;
  - *n_jobs*: Número de *threads* do *OpenMP* a serem usadas na computação dos centroides, onde 'None' significa o uso de todos os núcleos do processador.

- MCL:
  - *iterations*: Número máximo de iterações para cada execução;
  - *expansion*: Fator de expansão dos agrupamentos, explicado na Seção 2.2.2;
  - *inflation*: Fator de inflação dos agrupamentos, explicado na Seção 2.2.2.

- k-Shape:
  - *n_clusters*: Controla o número de agrupamentos a serem formados, assim como o número de centroides a serem gerados.
  - *init*: Método para a inicialização dos centroides. Onde é possível selecionar entre *'random'*, que seleciona os centroides de maneira aleatória, ou selecionar manualmente os centroides;
  - *max_iter*: Número máximo de iterações para cada execução;
  - *random_state*: Define a geração dos números aleatórios para a inicialização dos centroides. Pode ser usado um número inteiro para deixar a aleatoriedade determinística.

Quadro 3.1 – Valores dos hiperparâmetros descritos acima.

Hiperparâmetro	**k-Means**	**MCL**	**k-Shape**
n_clusters	7	—	7
init	'kmeans++'	—	'random'
max_iter / *iterations*	100	100	100
random_state	0	—	0
n_jobs	1	—	—
expansion	—	2	—
inflation	—	1.7	—

Fonte: O autor.

3.1.5 Validação do modelo

A validação *offline* é um passo indispensável em um fluxo de trabalho MLN para estimar se o algoritmo de aprendizado funciona de modo aceitável. Durante este passo, a validação cruzada é normalmente usada para testar a precisão em geral do modelo a fim de mostrar se o modelo está sobreajustado (*overfitting*) ou insuficiente (*under-fitting*). Isto fornece uma boa orientação de quão otimizado está o modelo, por exemplo, aumentando o volume de dados e reduzindo a complexidade do modelo quando ele está sobreajustado. Analisar amostras erradas ajuda a descobrir a causa de erros para determinar se o modelo e os recursos são adequados ou os dados são representativos o suficiente para um problema.

No presente trabalho[21] foi utilizada a validação cruzada, uma técnica para avaliar a capacidade de generalização de um modelo, a partir de um conjunto de dados. O conceito central das técnicas de validação cruzada é o particionamento do conjunto de dados em subconjuntos mutuamente exclusivos, e posteriormente, o uso de alguns destes subconjuntos para a estimação dos parâmetros do modelo (ZHANG, 2011). A classe *cross_val_score* disponível no pacote *Scikit-learn* (PEDREGOSA et al., 2011) foi utilizada para a fase de validação do modelo, pois possibilita a aplicação automatizada de várias execuções da validação em um conjunto de dados.

3.1.6 Comparação dos algoritmos

Para realizar a comparação entre os três algoritmos selecionados, foram verificados o tempo de execução de cada algoritmo sob o mesmo conjunto de dados e o grau de precisão dos algoritmos na classificação das anomalias de rede encontradas na base de dados utilizada. Para verificar o tempo de execução dos algoritmos, foi utilizada a classe *perf_counter* do pacote *time*. Essa classe retorna o valor (em frações de segundo) de um contador de desempenho, i.e. um relógio com a maior resolução disponível para medir uma duração curta. Foi calculada a média do tempo de dez execuções para cada algoritmo. Os valores obtidos podem ser observados no Capítulo 4. A máquina utilizada para a execução desses testes possui um processador Intel Core i5-7400 de 3.0Ghz, 16GB de memória RAM de 2400Mhz e um SSD de 480GB de 530MB/s de velocidade de leitura.

Para verificar a precisão dos algoritmos, foi utilizada a classe *classification_report* do pacote *Scikit-learn* (PEDREGOSA et al., 2011). Essa classe compara os valores encontrados pelos algoritmos com os valores reais, disponibilizados pela coluna *"Label"* da base de dados. Ela retorna um relatório com algumas métricas de classificação, como *precision*, *recall*, *f1-score* e *accuracy*. *Precision* é dado pela razão $tp/(tp+fp)$, onde "tp" é o número

[21] https://github.com/adonaigoncalves/TCC-Adonai

de verdadeiros positivos (quando o algoritmo considera uma amostra como pertencente à uma classe e isso condiz com a realidade) e "fp" é o número de falsos positivos (quando o algoritmo considera uma amostra como pertencente à uma classe e isso não condiz com a realidade). Já o *recall* é dado pela razão $tp/(tp + fn)$, onde "tp" é o número de verdadeiros positivos e "fn" é o número de falsos negativos (quando o algoritmo considera uma amostra como não pertencente à uma classe e na realidade a amostra pertence). E o *f1-score* é dado pela razão $(2 * precision * recall)/(precision + recall)$. Por fim, a *accuracy*, ou acurácia, é a razão $(tp + tn)/n$, onde "tp" é o número de verdadeiros positivos, "tn" é o número de verdadeiros negativos (quando o algoritmo considera uma amostra como não pertencente à uma classe e na realidade a amostra realmente não pertence) e "n" é o número total de amostras de todas as classes.

3.2 AVALIAÇÃO

Para a avaliação deste trabalho, foram selecionados três dos dez dias disponibilizados pelo Canadian Institute for Cybersecurity (2018) como base de dados de treinamento. Esses dados foram concatenados, os melhores atributos foram selecionados, ilustrado no Quadro 3.3 e tratados de forma que pudessem ser utilizados como entradas dos algoritmos de agrupamento. Foi então realizado um tratamento no desbalanceamento das classes, pois a base de dados contém uma quantidade desequilibrada de registros por classe como indicado no Quadro 3.2. Os dados são embaralhados para evitar o *overfitting* e então padronizados para reduzir a variância entre os valores de uma mesma classe.

Depois da preparação dos dados de treinamento, são feitas medições de tempo de execução e precisão da classificação dos algoritmos. Nesta etapa, os algoritmos são executados diversas vezes para que se perceba uma consistência no tempo de execução, visto que o *hardware* da máquina pode causar oscilações no resultado. Então é feita verificação da precisão dos algoritmos, onde foram utilizadas métricas populares na classificação de algoritmos de ML, como *accuracy*, *precision*, *recall* e *f1-score*.

Quadro 3.2 – Classes rotuladas da base de dados e quantidade de registros.

Classe	**Quantidade de registros**	**Classe**	**Quantidade de registros**
Benign	2024536	DoS attacks-GoldenEye	41508
DDOS attack-HOIC	686012	DoS attacks-Slowloris	10990
FTP-BruteForce	193360	DDOS attack-LOIC-UDP	1730
SSH-Bruteforce	187589		

Fonte: O autor.

Quadro 3.3 – Melhores atributos selecionados.

Nome da Classe	**Descrição**
Flow_Duration	Duração do fluxo em microssegundos.
TotLen_Fwd_Pkts	Tamanho total do pacote na direção direta.
Fwd_Pkt_Len_Mean	Tamanho médio do pacote na direção direta.
Bwd_Pkt_Len_Min	Tamanho mínimo do pacote na direção inversa.
Bwd_Pkt_Len_Std	Tamanho de desvio padrão do pacote na direção inversa.
Flow_IAT_Mean	Tempo médio entre dois pacotes enviados no fluxo.
Flow_IAT_Std	Desvio padrão do tempo entre dois pacotes enviados no fluxo.
Flow_IAT_Min	Tempo mínimo entre dois pacotes enviados no fluxo.
Fwd_IAT_Mean	Tempo médio entre dois pacotes enviados na direção direta.
Fwd_IAT_Min	Tempo mínimo entre dois pacotes enviados na direção direta.
Bwd_IAT_Mean	Tempo médio entre dois pacotes enviados na direção inversa.
Fwd_PSH_Flags	Número de vezes que o sinalizador PSH foi definido em pacotes que viajam na direção direta (0 para UDP).
Fwd_Pkts/s	Número de pacotes diretos por segundo.
SYN_Flag_Cnt	Número de pacotes com SYN.
ACK_Flag_Cnt	Número de pacotes com ACK.
Pkt_Size_Avg	Tamanho médio do pacote.
Subflow_Fwd_Byts	O número médio de bytes em um subfluxo na direção direta.
Init_Fwd_Win_Byts	O número total de bytes enviados na janela inicial na direção direta.

Fonte: Adaptado de (Canadian Institute for Cybersecurity, 2018)

4 RESULTADOS E DISCUSSÃO

Descrita a metodologia e o modo de avaliação utilizados, uma série de execuções foram realizadas com propósito de classificar e comparar os algoritmos selecionados. Este capítulo ilustra os resultados da análise da validação cruzada dos algoritmos e resultados acerca da comparação do desempenho desses algoritmos.

4.1 RESULTADOS DA VALIDAÇÃO CRUZADA

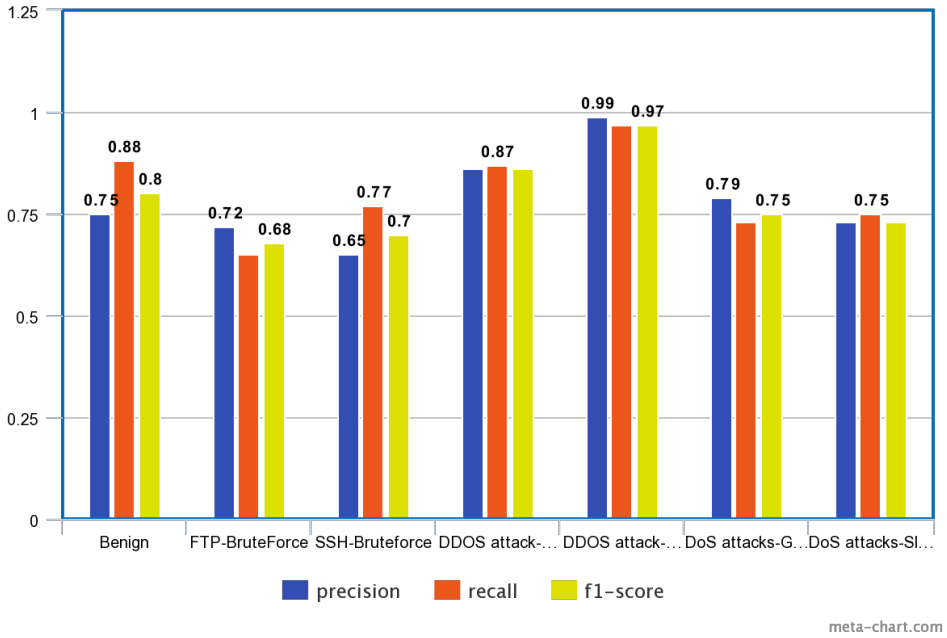
Com o treinamento dos algoritmos e configuração dos melhores hiperparâmetros, alcançados através de diversos testes, foram obtidos os resultados demonstrados no Quadro 4.1, Figura 4.1, Figura 4.2 e Figura 4.3. Como pode ser visto na Figura 4.1, o algoritmo k-Means apresentou a maior eficácia na detecção de ataques DDOS, alcançando 99% de *precision* e 97% de *recall* para a ferramenta HOIC e 86% de *precision* e 87% de *recall* para a ferramenta LOIC. Enquanto o algoritmo MCL apresentou a maior eficácia na detecção de eventos benignos, i.e. usuários comuns, alcançando 85% de *precision* e 88% de *recall*, ilustrado na Figura 4.2. Já o algoritmo k-Shape apresentou a maior eficácia na detecção de ataques de força bruta, alcançando 72% de *precision* e 85% de *recall* para ataques pelo protocolo FTP e 85% de *precision* e 87% de *recall* para ataques pelo protocolo SSH, ilustrado na Figura 4.3.

Quadro 4.1 – *Precision*, *recall* e *f1-score* dos três algoritmos.

	k-Means			**MCL**			**k-Shape**		
Classe	***precision***	***recall***	***f1-score***	***precision***	***recall***	***f1-score***	***precision***	***recall***	***f1-score***
Benign	0.75	0.88	0.80	**0.85**	**0.88**	**0.86**	0.72	0.71	0.71
FTP-BruteForce	0.72	0.65	0.68	0.75	0.69	0.71	**0.72**	**0.85**	**0.77**
SSH-Bruteforce	0.65	0.77	0.70	0.68	0.70	0.68	**0.85**	**0.87**	**0.85**
DDOS attack-LOIC-UDP	**0.86**	**0.87**	**0.86**	0.80	0.77	0.78	0.86	0.87	0.86
DDOS attack-HOIC	**0.99**	**0.97**	**0.97**	0.90	0.89	0.89	0.89	0.91	0.89
DoS attacks-GoldenEye	0.79	0.73	0.75	0.71	0.72	0.71	0.71	0.74	0.72
DoS attacks-Slowloris	0.73	0.75	0.73	0.63	0.68	0.65	0.73	0.68	0.70

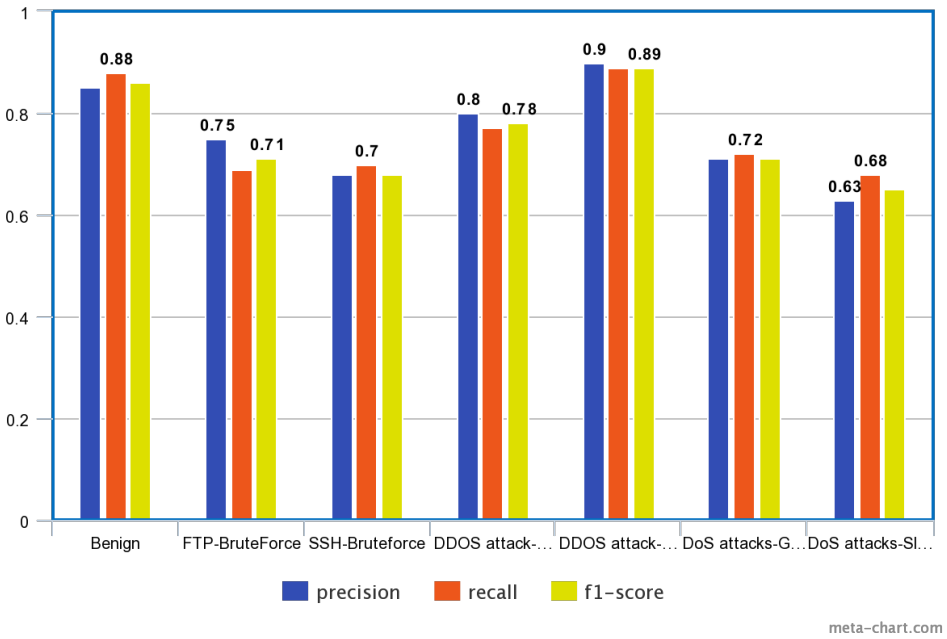
Fonte: O autor.

Figura 4.1 – *Precision*, *recall* e *f1-score* do k-Means.



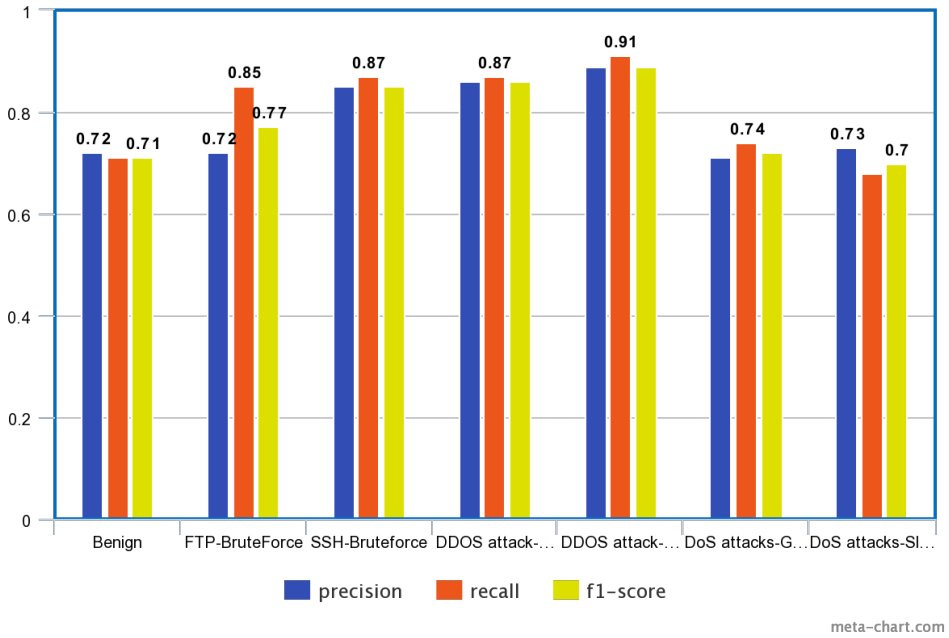
Fonte: O autor.

Figura 4.2 – *Precision*, *recall* e *f1-score* do MCL.



Fonte: O autor.

Figura 4.3 – *Precision*, *recall* e *f1-score* do k-Shape.



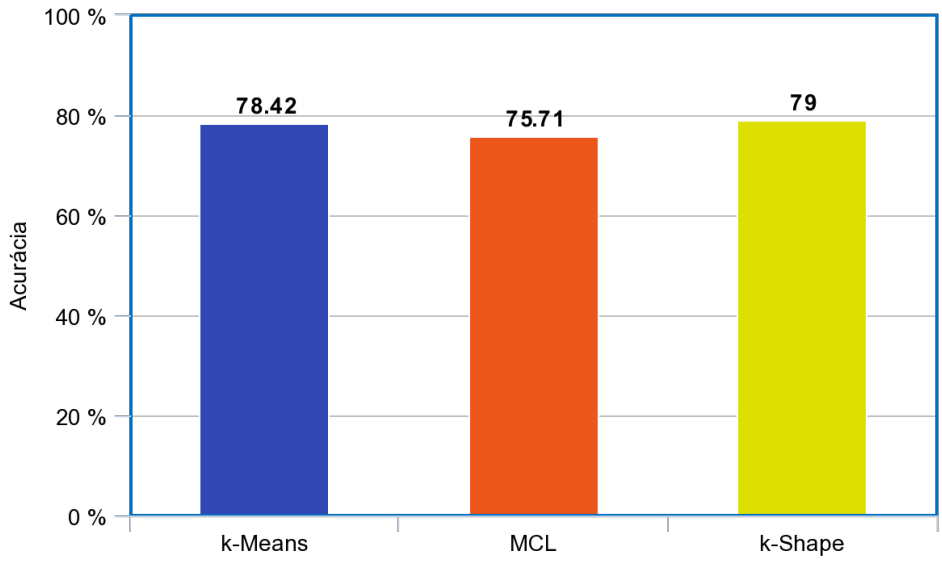
Fonte: O autor.

4.2 RESULTADOS DA COMPARAÇÃO DE DESEMPENHO DOS ALGORITMOS

Com o intuito de selecionar o melhor algoritmo para detecção de anomalias, entre os escolhidos, foram feitas dez execuções de cada algoritmo para coletar a média de tempo de execução dos algoritmos na detecção de anomalias, ilustrado nas Figuras 4.4. Como foi utilizado um valor fixo para o parâmetro *random_state*, explicado na Seção 3.1.4, dos algoritmos k-Means e k-Shape, não foi observada nenhuma mudança nas suas acurácias. Entretanto, como o MCL não possui um parâmetro desse tipo, foi feita a média de acurácia entre as execuções, ilustrado nas Figuras 4.5.

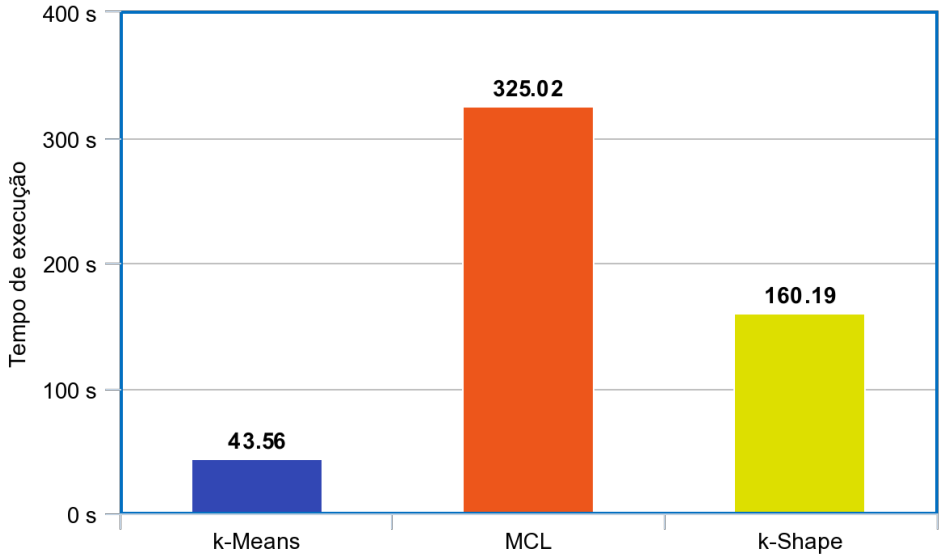
Como pode ser observado na Figura 4.4, o algoritmo k-Shape obteve a maior acurácia entre os três algoritmos, com 79% em relação aos 78.72% alcançado pelo k-Means e 75.71% do MCL. Em questão de acurácia, os três algoritmos obtiveram uma acurácia bastante similar. Já em questão de tempo de execução, ilustrado na Figura 4.5, o algoritmo k-Means é consideravelmente mais rápido que os outros dois algoritmos. Com 43.56 segundos de tempo de execução, em relação aos 325.05 segundos do MCL e os 160.19 segundos do k-Shape e, considerando o resultado dos testes de acurácia, pode-se afirmar que o algoritmo k-Means é o melhor algoritmo, entre os três selecionados, para detecção de anomalias em rede.

Figura 4.4 – Gráfico de acurácia média dos algoritmos.



Fonte: O autor.

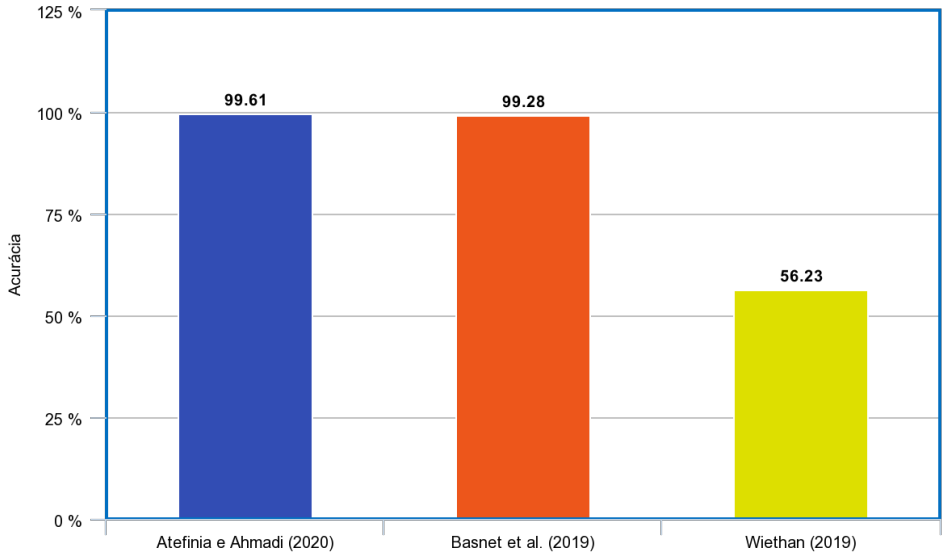
Figura 4.5 – Gráfico de tempo de execução médio dos algoritmos.



Fonte: O autor.

Na Figura 4.6 abaixo, é ilustrada a acurácia média dos trabalhos relacionados que utilizaram a mesma base de dados que esta proposta.

Figura 4.6 – Acurácia média obtida pelos trabalhos relacionados.



Fonte: O autor.

5 CONSIDERAÇÕES FINAIS

Esse trabalho teve como propósito estudar diferentes estratégias de detecção de anomalias em redes de computadores com algoritmos de agrupamento. Utilizou a metodologia de Wang et al. (2017), que descreve o fluxo de trabalho típico do aprendizado de máquina para redes. Os treinamentos foram feitos sobre dados estatísticos retirados de arquivos de *log* de rede no formato PCAP e salvos em arquivos no formato CSV.

Foi realizada uma revisão bibliográfica sobre sistemas de detecção de intrusão, anomalias em redes, aprendizado de máquina, algoritmos de agrupamento e trabalhos relacionados para fornecer embasamento às técnicas propostas no presente trabalho. Foram explorados conceitos básicos de anomalias de rede, sistemas de detecção de intrusão, algoritmos de agrupamento e detecção de anomalias descobertos em trabalhos relacionados.

Na etapa de desenvolvimento, foram utilizados três algoritmos de agrupamento com características distintas em seus treinamentos e tratamentos de dados. Esta abordagem visou analisar a capacidade de generalização dos modelos, a fim de comparar a acurácia e desempenho dos algoritmos selecionados. Foram feitos testes para escolha dos melhores hiperparâmetros para o treinamento dos algoritmos, assim como uma preparação da base de dados para o treinamento. Foram feitos testes de validação cruzada para medir o desempenho dos algoritmos, e por fim, testes de tempo de execução e acurácia dos algoritmos, com o intuito de selecionar o melhor algoritmo entre eles para detecção de anomalias em rede.

Na etapa de discussão, pôde-se notar que os três algoritmos obtiveram uma acurácia similar, porém o k-Means se destacou no tempo de execução, mesmo sem utilizar o seu suporte à múltiplas *threads*, e detecção de ataques DDOS. O algoritmo MCL obteve o melhor resultado em detecção de eventos benignos e, como não precisa de um valor predefinido de número de agrupamentos, pode ser utilizado na detecção de falsos positivos ou anomalias benignas em trabalhos futuros. O algoritmo k-Shape obteve a melhor acurácia entre os algoritmos selecionados, e se receber algumas otimizações e um suporte à múltiplas *threads*, como no k-Means, ele pode vir a superar o k-Means no futuro.

Como trabalhos futuros, sugere-se a implementação de novos modelos de aprendizado de máquina para análise de rede e detecção anomalias de rede sem a presença de agentes maliciosos. Seria interessante a execução e análise do sexto passo original da metodologia aplicada no desenvolvimento do presente trabalho. Esses algoritmos também poderiam ser implementados, juntamente com um sistema de detecção de intrusão e redes neurais, para realizar a análise em tempo real de uma rede monitorada.

REFERÊNCIAS BIBLIOGRÁFICAS

AL-KASASSBEH, M.; ADDA, M. Network fault detection with wiener filterbased agent. **Elsevier - Journal of Network and Computer Applications**, v. 32, p. 824–833, 2009.

ALVES, F. C. C. et al. Uma revisão sobre as publicações de sistemas de detecção de intrusão. **Revista de Informática Teórica e Aplicada**, v. 23, p. 67–99, 2016.

ATEFINIA, R.; AHMADI, M. Network intrusion detection using multi-architectural modular deep neural network. **The Journal of Supercomputing - Springer**, v. 77, p. 3571–3593, 2020.

BASNET, R. B. et al. Towards detecting and classifying network intrusion traffic using deep learning frameworks. **Journal of Internet Services and Information Security**, v. 9, p. 1–17, 2019.

BHUYAN, M. H.; BHATTACHARYYA, D. K.; KALITA, J. K. Network anomaly detection: Methods, systems and tools. **IEEE - Communications Surveys & Tutorials**, v. 16, p. 303–336, 2014.

BREIMAN, L. **Machine Learning - Random Forests**. Netherlands: Kluwer Academic Publishers, 2001. 5–32 p.

Canadian Institute for Cybersecurity. **CSE-CIC-IDS2018 on AWS**: Ids 2018. University of New Brunswick, 2018. Acesso em 24 maio 2021. Disponível em: <https://www.unb.ca/cic/datasets/ids-2018.html,https://registry.opendata.aws/cse-cic-ids2018/>.

CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly detection: A survey. **ACM computing surveys (CSUR)**, v. 41, p. 1–72, 2009.

CHEN, Y. et al. **The UCR Time Series Classification Archive**. 2015. Acesso em maio 2014. Disponível em: <www.cs.ucr.edu/~eamonn/time_series_data/>.

COLE, E.; KRUTZ, R.; CONLEY, J. W. **Information Theory, Inference, and Learning Algorithms**. USA: Wiley Publishing, 2005. 557–571 p.

DENYSENKO, M. **Detecting Anomalies in Log Data from Apache Web Server by Means of Cluster Analysis**. 2015. 134 f. Dissertação (Mestrado em Sensoriamento Remoto) — Merseburg University of Applied Sciences, Ucrânia, 2015.

DONGEN, S. M. van. **Graph clustering by flow simulation**. 2000. 175 f. Tese (Doutorado em Matemática e Ciência da Computação) — Universidade de Utrecht, Utrecht, 2000.

DURUMERIC, Z. et al. The matter of heartbleed. **2014 Internet Measurement Conference**, p. 475–488, 2014.

EVGENIOU, T.; PONTIL, M. Support vector machines: Theory and applications. **Machine Learning and Its Applications, Advanced Lectures**, p. 249–257, 2001.

FADLULLAH, Z. M. et al. State-of-the-art deep learning: Evolving machine intelligence toward tomorrows intelligent network traffic control systems. **IEEE - Communications Surveys & Tutorials**, v. 19, p. 2432–2455, 2017.

FEILY, M.; SHAHRESTANI, A.; RAMADASS, S. A survey of botnet and botnet detection. **2009 Third International Conference on Emerging Security Information, Systems and Technologies**, p. 268–273, 2009.

GuyAllard. **Markov Clustering**: Mcl. Github, 2017. Acesso em 24 maio 2021. Disponível em: <https://github.com/guyallard/markov_clustering>.

HAAS, A. **Um Sistema por Processamento de Fluxos Aplicado à Análise e Monitoramento da Rede**. 2019. 92 f. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Santa Maria, Santa Maria, 2019.

HAN, D. et al. A ddos attack detection system based on spark framework. **Computer Science and Information Systems**, v. 14, p. 769–788, 2017.

HE, H. **Applications of Reference Cycle Building and K-shape Clustering for Anomaly Detection in the Semiconductor Manufacturing Process**. 2018. 76 f. Dissertação (Master of Science in Mechanical Engineering) — Massachusetts Institute of Technology, USA, 2018.

IHYA, R. et al. J48 algorithms of machine learning for predicting user's the acceptance of an e-orientation systems. **International Conference on Smart City Applications**, v. 4, p. 1–8, 2019.

KNUDSEN, L. R.; ROBSHAW, M. J. B. **The Block Cipher Companion**. Berlin, Heidelberg: Springerg, 2011. 109–126 p.

LEEVY, J. L.; KHOSHGOFTAAR, T. M. A survey and analysis of intrusion detection models based on cse-cic-ids2018 big data. **Journal of Big Data**, v. 7, p. 1–19, 2020.

MACKAY, D. J. **Information Theory, Inference, and Learning Algorithms**. Reino Unido: Cambridge University Press, 2005. 284–292 p.

MACQUEEN, J. Tornadoes and downbursts in the context of generalized planetary scales. **Berkeley Symposium**, v. 5, p. 281–297, 1967.

MARZ, N.; WARREN, J. **Big Data: Principles and Best Practices of Scalable Real-Time Data Systems**. New York: Manning Publications Co., 2015. 22–44 p.

PAPARRIZOS, J.; GRAVANO, L. k-shape: Efficient and accurate clustering of time series. **ACM SIGMOD Record**, v. 45, p. 69–76, 2016.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PINHEIRO, J. M. dos S. Ameaças e ataques aos sistemas de informação: Prevenir e antecipar. **Cadernos UniFOA**, v. 5, p. 11–21, 2007.

RYZA, S. et al. **Advanced analytics with spark: patterns for learning from data at scale**. USA: OReilly Media, Inc, 2017.

savyakhosla. **ML | K-means++ Algorithm**. https://www.geeksforgeeks.org/, 2020. Acesso em 01 jul 2021. Disponível em: <https://www.geeksforgeeks.org/ml-k-means-algorithm/>.

SHARAFALDIN, I.; LASHKARI, A. H.; GHORBANI, A. A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. **4th International Conference on Information Systems Security and Privacy (ICISSP)**, v. 4, p. 108–116, 2018.

SHON, T.; MOON, J. A hybrid machine learning approach to network anomaly detection. **Elsevier - Information Sciences**, v. 177, p. 3799–3821, 2007.

Stijn van Dongen. **MCL - a cluster algorithm for graphs**: Introduction. https://micans.org, 2000. Acesso em 01 jul 2021. Disponível em: <https://micans.org/mcl/>.

TAVENARD, R. et al. Tslearn, a machine learning toolkit for time series data. **Journal of Machine Learning Research**, v. 21, p. 1–6, 2020.

THOTTAN, M.; JI, C. Anomaly detection in ip networks. **IEEE - TRANSACTIONS ON SIGNAL PROCESSING**, v. 51, p. 2191–2204, 2003.

WANG, M. et al. Machine learning for networking: Workflow, advances and opportunities. **IEEE Network**, v. 32, p. 92–99, 2017.

WIETHAN, W. da S. **UTILIZAÇÃO DE APRENDIZADO PROFUNDO PARA DETECÇÃO DE ANOMALIAS EM REDES DE COMPUTADORES**. 2019. 38 f. Monografia (Trabalho de Conclusão de Curso) — Curso de Graduação em Ciência da Computação, Universidade Federal de Santa Maria, Santa Maria, 2019.

ZARPELãO, B. B. **Detecção de Anomalias em Redes de Computadores**. 2010. 159 f. Tese (Doutorado em Engenharia Elétrica) — Universidade Estadual de Campinas, Campinas, SP, Brasil, 2010.

ZHANG, F. **Cross-validation and regression analysis in high-dimensional sparse linear models**. 2011. 91 f. Tese (Doutorado em Filosofia) — Stanford University, California. USA, 2011.