DISSERTAÇÃO DE MESTRADO PROFISSIONAL

Detecção de Ataques de Negação de Serviço em SGBDs A Partir de Logs Internos Usando Abordagens Supervisionada e Não Supervisionada

Danilo Anderson de Moura Chagas

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

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

DISSERTAÇÃO DE MESTRADO PROFISSIONAL

Detecção de Ataques de Negação de Serviço em SGBDs A Partir de Logs Internos Usando Abordagens Supervisionada e Não Supervisionada

Danilo Anderson de Moura Chagas

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

Banca Examinadora

Prof. Dr. Vinícius Gonçalves Pereira, Ph.D, FT/UnB _____
Orientador

Prof. Rafael Rabelo Nunes, Ph.D, FT/UnB _____
Examinador Interno

Prof. José Rodrigues Torres Neto, Ph.D, INF/UFPI _____
Examinador Externo

FICHA CATALOGRÁFICA

CHAGAS, DANILO A. M.
Detecção de Ataques de Negação de Serviço em SGBDs A Partir de Logs Internos Usando Abordagens Supervisionada e Não Supervisionada [Distrito Federal] 2024.
xvi, 68 p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2024).
Dissertação de Mestrado Profissional - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Negação de Serviço	2. SGBD
3. Supervisionado	4. Não Supervisionado
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

CHAGAS, D.A.M. (2024). *Detecção de Ataques de Negação de Serviço em SGBDs A Partir de Logs Internos Usando Abordagens Supervisionada e Não Supervisionada*. Dissertação de Mestrado Profissional, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 68 p.

CESSÃO DE DIREITOS

AUTOR: Danilo Anderson de Moura Chagas
TÍTULO: Detecção de Ataques de Negação de Serviço em SGBDs A Partir de Logs Internos Usando Abordagens Supervisionada e Não Supervisionada.
GRAU: Mestre em Engenharia Elétrica ANO: 2024

Danilo Anderson de Moura Chagas
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

DEDICATÓRIA

Dedico este que é o resultado de um trabalho árduo, iniciado do zero, àqueles que são minha fonte de inspiração e razão de tudo o que faço: minha esposa, Laís, e nossos filhos, Jonathan e Daniel.

AGRADECIMENTOS

Agradeço, primeiramente, ao Senhor e Salvador da minha vida, Jesus Cristo. Ele é aquele que foi me tirar do monturo e me trouxe para caminhos abundantes. “Porque dele e por ele, e para ele, são todas as coisas; glória, pois, a ele eternamente”. Ao professor Rafael Rabelo por ter me incentivado a tentar entrar no PPEE, algo que eu não tinha cogitado até então, e que se mostrou a escolha certa. Ao meu co-orientador, professor Geraldo Filho, que com sua calma, paciência e imenso conhecimento me guiou por um caminho que leva a um trabalho de excelência. Ao meu orientador e exemplo de ser humano, professor Vinícius Gonçalves: levarei seus ensinamentos acadêmicos e, principalmente, pessoais, humanos, para toda a minha vida. Foi um prazer e uma honra estar em sua presença nesta caminhada, na qual me ensinou que amor, compaixão e empatia andam juntos com firmeza e exigência na busca por primor. Aos meus pais, que lutaram e se doaram por mim e minhas irmãs a vida inteira, sempre se atendo fielmente aos princípios de respeito e ética. Finalmente, à minha esposa, Laís, minha companheira, auxiliadora, pesquisadora, professora, exortadora. Sem você, eu não teria desejado ser uma pessoa melhor, um profissional melhor, um aluno melhor. Você é o motivo de eu querer evoluir e conquistar coisas novas. Amo-te para sempre!

RESUMO

A taques de Negação de Serviço (*Denial-of-Service* – DoS) impõem ameaças ao cumprimento dos propósitos de uma organização, uma vez que resultam em sérios problemas relacionados à disponibilidade dos sistemas de informação. Os ataques DoS têm sido extensivamente estudados na literatura, entretanto os trabalhos existentes geralmente focam nas camadas de rede e transporte ou em protocolos como o HTTP. Banco de dados, infraestrutura crítica para provimento de serviços, possui mecanismos de gravação de informações (*logs*) de consultas SQL e sessões, o que gera grandes volumes de dados. Embora os bancos de dados sejam vulneráveis ao DoS, eles não são totalmente cobertos por ferramentas comerciais ou por pesquisas sobre a detecção de tais ataques. As técnicas de Aprendizado de Máquina (AM) são altamente eficazes na identificação de padrões em grandes quantidades de dados, tais como os *logs* SQL de banco de dados. Assim, este trabalho desenvolveu a aplicação de AM na detecção de ataques DoS a banco de dados a partir dos *logs* de consultas nele executadas. Faz uso de duas abordagens complementares de AM diferentes: supervisionado e não supervisionado. Como resultado, a classificação de registros obteve um F1-*score* de 94,44% e a Detecção de Anomalias atingiu um F1-*score* de 75,75%, indicando a efetividade das abordagens desenvolvidas.

ABSTRACT

Denial-of-Service (DoS) attacks impose threats to the accomplishment of an organization's purposes once they result in serious issues related to the availability of information systems. DoS attacks have been extensively studied in the literature, especially in their most dangerous form, the Distributed Denial-of-Service (DDoS). However, existing works usually focus on the network and transport layers or protocols like HTTP. Database, a critical infrastructure for service provision, has mechanisms for recording information (logs) of SQL queries and sessions, which generates large volumes of data. Although databases are vulnerable to DDoS, they are not entirely covered by commercial tools or research on detecting such attacks. Machine Learning (ML) techniques are highly effective in identifying patterns in large amounts of data, such as database SQL logs. Thus, this work developed the application of ML to detect DDoS attacks on a database from the logs of queries executed on it. It makes use of two complimentary approaches of ML: supervised and unsupervised. As a result, the classification of records obtained an F1-score of 94.44% and the Anomaly Detection achieved an F1-score of 75.75%, which indicates the effectiveness of the developed approaches.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Gregos

ζ	Limiar para transformar pontuação de anomalia, contínua, em variável binária

Siglas

ADC	*Application Delivery Controller* (infraestrutura equivalente ao *proxy*, que serve para disponibilizar páginas *web*)
AM	Aprendizado de Máquina (*Machine Learning*)
AP	Aprendizado Profundo (*Deep Learning*)
botnet	*robot networks* (redes zumbis para perpetrarem ataques de DDoS)
DA	Detecção de Anomalias (*Anomaly Detection*
DBMS	*Database Management System* (o mesmo que SGBD)
DDoS	*Distributed Denial-of-Service* (DoS Distribuído - várias origens)
DNS	*Domain Name System* (protocolo de camada 7 para resolução de nomes)
DoS	*Denial-of-Service* (Ataque de Indisponibilização de Ambientes)
FN	Falso Negativo
FP	Falso Positivo
HTTP	*Hypertext Transfer Protocol* (protocolo camada 7 de entrega de páginas *web*)
HTTPS	*Hypertext Transfer Protocol Secure* (versão do HTTP com criptografia)
IDS	*Intrusion Detection System* (sistema de detecção de ataques de intrusão)
IPS	*Intrusion Prevention System* (sistema de prevenção de ataques de intrusão)
LSTM	*Long Short-Term Memory* (algoritmo de AP)
MLP	*Multilayer Perceptron* (algoritmo de AP)
MSE	*mean of squared errors* (função para cálculo de erro entre previsão e observado)
NLP	*Natural Language Processing* (procesamento de linguagem natural
NMS	*Network Monitoring System* (sistema de monitoramento de ativos em rede)
OHE	*One-Hot Encoding* (algoritmo de codificação de variáveis categóricas)
OSI	*Open Systems Interconnect* (modelo teórico de redes de computadores)
PCA	*Principal Component Analysis* (algoritmo de redução de dimensionalidade)
ReLU	*Rectified Linear Unit* (função de ativação de uma RN)
RN	Rede Neural (algoritmo de AP)
RNR	Rede Neural Recorrente (algoritmo de AP)
ROC-AUC	*Reciever Operator Characteristics - Area Under the Curve* (gráfico para determinação da qualidade do classificador)
SGBD	Sistema Gerenciador de Bancos de Dados (infraestrutura que gerencia os bancos de dados)
SQL	*Structured Query Language* (linguagem para consultas a bancos de dados)
SQLIA	*SQL Injection Attack* (ataque a SGBD por meio de injeção de consultas maliciosas)
SSE	*sum of squared errors* (função para cálculo de erro entre previsão e observado)
Sigmoid	Função de ativação de uma RN cujo resultado retornado está entre 0 e 1
Tbps	*Terabits* por segundo (unidade de volume de tráfego)
TCP/IP	*Transmission Control Protocol/Internet Protocol* (modelo prático e pilha de protocolos de comunicação)
VN	Verdadeiro Negativo
VP	Verdadeiro Positivo
XGBoost	*Extreme Gradient Boosting* (algoritmo de AM supervisionado)

1 INTRODUÇÃO

As organizações, privadas ou públicas, ofertam serviços para serem consumidos pelo seu público-alvo, seja de forma manual ou informatizada. Com a evolução de processos e tecnologias, serviços passaram a ser disponibilizados de maneira informatizada e *on-line*. Essa oferta aumentou bastante após o surgimento da pandemia de COVID-19 (8), no qual o mundo se viu restrito em seus deslocamentos, mas continuando a necessitar de tais serviços. Como efeito colateral, com uma quantidade maior de serviços presentes no mundo virtual, muitas vezes não configurados de forma correta ou por equipes sem o conhecimento aprofundado da infraestrutura, aumentaram-se os riscos de Segurança da Informação (9).

O provimento de serviços *on-line* enfrenta desafios de Segurança da Informação, pois sistemas chave para o funcionamento ou existência do ente provedor estão disponibilizados em espaço público (10). Este espaço é acessível a toda natureza de usuários, desde os legítimos – aqueles a quem realmente o serviço é útil – até os ilegítimos, que têm intenções maliciosas para com a organização. Estes últimos são os responsáveis pelos ataques cibernéticos.

Ataques cibernéticos possuem diversas classificações, mas todos os seus tipos têm o intuito de violar um dos três pilares da Segurança da Informação: Confidencialidade, Integridade e Disponibilidade (5, 11). Dentre esses ataques, os de Negação de Serviço (*Denial-of-Service* – DoS) abalam diretamente o pilar Disponibilidade. São ataques cujo intuito exclusivo é exaurir os recursos da infraestrutura subjacente aos serviços, tornando-os indisponíveis para os usuários (12).

A infraestrutura necessária para disponibilizar serviços na internet possui diversos componentes, como *links* de internet, roteadores, *firewalls*, *proxies*, *Application Delivery Controllers* (ADCs), servidores *web*, servidores de aplicação e Sistemas Gerenciadores de Bancos de Dados (SGBDs ou DBMSs – *Database Management Systems*). Cada um desses componentes está sujeito a apresentar vulnerabilidades que podem ser exploradas por atacantes no intuito de perpetrar um ataque cibernético (5, 13, 14).

Todo *software* disponibilizado na Internet faz acesso a algum tipo de banco de dados, sendo estes locais nos quais são armazenados os dados gerados por seus usuários. Um banco de dados é definido por Elmasri e Navathe como "uma coleção de dados relacionados" e os dados, por sua vez, são fatos que possuem significado implícito para usuários e para o serviço provido (15). O termo banco de dados é comumente utilizado com o sentido de SGBD. Estes, no entanto, possuem sentido mais amplo, pois armazenam um conjunto de bancos de dados. Além disso, são responsáveis por executar tarefas de manipulação dos bancos de dados, como o armazenamento e recuperação dos dados neles gravados (13, 15). Os SGBDs não ficam disponíveis diretamente para acesso por meio da Internet, mas os dados neles gravados são recuperados por meio dos *softwares* tornados públicos por quem disponibiliza o serviço. Costumam, portanto, estar numa localização mais interna à infraestrutura, também conhecida por *backend* (13). Apesar desta localização teoricamente mais protegida, são componentes que também estão vulneráveis a ataques cibernéticos.

Os ataques mais comuns aos SGBDs – e, consequentemente, os mais estudados na literatura – são os de SQL *Injection* (SQLIA - SQL *Injection Attack*) (13, 14). SQLIA visa o acesso não autorizado ao banco de dados, recuperação de dados sensíveis ou a indisponibilidade de tais serviços (13, 16, 17, 18) Este último

propósito, a indisponibilidade do serviço, classifica-se como DoS, sendo ele também abordado na literatura (19, 20).

Aprendizado de Máquina (AM), também conhecido pelo termo inglês *Machine Learning*, é uma área da Inteligência Artificial que capacita computadores a aprender a partir de dados sem uma programação específica para tal. Esta técnica tem sido aplicada em desafios que são complexos para métodos convencionais, onde ainda é necessário criar algoritmos para sua resolução (21). Algumas aplicações práticas do AM incluem reconhecimento de voz e imagem, tradução de textos em linguagens diferentes, recomendação de músicas ou filmes e determinação de preços sazonais. A detecção e a proteção contra ameaças cibernéticas, incluindo ataques DoS e variantes, também se valem desse método (5, 22, 23).

Cada componente de infraestrutura bem estabelecida apresenta um comportamento padrão quando do provimento de seus serviços. Os usuários dos sistemas, por exemplo, não sofrem um aumento de quantidade abrupto e nem há um uso exacerbado e repentino de recursos (24, 25). Um comportamento que foge a esses padrões chama-se uma Anomalia.

A Detecção de Anomalias (DA), do inglês *Anomaly Detection*, é um processo importante da Segurança da Informação, pois visa determinar os comportamentos padrão e, a partir daí, identificar atividades que fujam a esse padrão. Este comportamento anormal representa uma Anomalia (26, 27, 28) e é também conhecido como um *outlier* ou uma *novelty*.

Nos trabalhos acadêmicos observados, nota-se um enfoque majoritário na análise nas camadas mais baixas dos modelos de referência de redes de comunicação, mais comumente o modelo OSI (*Open Systems Interconnect*) e o modelo TCP/IP (*Transmission Control Protocol/Internet Protocol*) (18, 29, 30, 31). Normalmente, essa análise se dá utilizando-se captura de pacotes de rede por meio de *softwares* como o *tcpdump* ou o *Wireshark* (32). Essa captura permite uma investigação em baixo nível e exige uma análise mais criteriosa, visto que pacotes podem estar fragmentados e necessitam ser recompostos para dar continuidade à avaliação (32). Além deste inconveniente, possuem informações anteriores à execução dos comandos e rotinas nos elementos da infraestrutura. Ou seja, não conseguem representar o quanto cada execução impactou efetivamente no desempenho do elemento.

SGBDs relacionais registram cada sessão aberta e cada consulta realizada em tabelas de *log*. Estas tabelas contêm dados valiosos sobre o número de execuções de consulta, o número de registros retornados, consumo de memória, tempo de uso, tempo de espera do processador e as sessões abertas (33). Suas informações incluem o comando SQL executado, parâmetros de desempenho da consulta, o usuário que a executou e o estado da sessão. Assim, o SGBD registra parâmetros quantitativos independentemente de sua natureza, sejam eles consultas e sessões legítimas ou com propósitos maliciosos – embora, neste caso, tenham uma sintaxe coerente. Este fato transforma os *logs* internos de um SGBD em fontes ricas de informações para detectar atividades anormais de tal componente.

Diante dos conceitos apresentados e de um cenário real de um ataque perpetrado a um SGBD, surgiram algumas questões para o desenvolvimento deste trabalho:

1. Como o Aprendizado de Máquina, tanto em abordagens supervisionadas quanto não supervisionadas, pode ser aplicado para detectar ataques DoS/DDoS em SGBDs?
2. É possível utilizar os *logs* internos de um SGBD como fonte eficaz para a detecção de ataques

DoS/DdoS, visto que possuem informações mais ricas sobre o funcionamento desta infraestrutura?

3. É possível identificar ataques de DDoS por meio de atividades anômalas em SGBDs a partir de seus *logs* internos?

4. Qual é o desempenho comparativo de diferentes algoritmos de Aprendizado de Máquina na detecção de ataques DoS/DDoS em SGBDs, e como esses algoritmos podem ser otimizados para melhor precisão e eficiência?

5. De que forma a detecção de ataques DoS/DDoS em SGBDs contribui para a melhoria da segurança da informação em organizações?

Aliado ao uso do AM e da DA, nosso trabalho realizou a junção desses métodos e técnicas aos *logs* internos de um SGBD como ferramenta para detecção de ataques DDoS (variante do DoS) a esse elemento. Portanto, este trabalho usa AM para detectar ataques DDoS em bancos de dados. Nossa abordagem utiliza tabelas de *log* do SGBD com todos os dados da sessão para treinar algoritmos de AM supervisionados, que são usados para classificá-los em termos de sua legitimidade em relação à disponibilidade de serviço. Empregamos esses mesmos dados para treinar modelos de AM não supervisionado na forma de DA para detecção de atividades anômalas e, portanto, ataques. Frisa-se que as consultas (*queries*) em si não são analisadas neste trabalho – não há necessidade disso, já que todas as informações necessárias podem ser extraídas das sessões registradas. Este fato agrega ao trabalho uma característica de Segurança da Informação chamada princípio do menor privilégio (34), pois somente é acessado aquilo que é necessário para a detecção.

Deve-se notar que esta classificação usa informações sobre estados de sessão gerados sempre que uma consulta SQL é executada no SGBD. Neste trabalho, realizamos um estudo com dados de sessões legítimas e ataques em um SGBD Oracle de uma instituição pública brasileira do sistema judiciário. Nosso trabalho se concentra em determinar se é possível detectar ataques de DoS (incluindo DDoS) em SGBDs a partir dos dados coletados dos logs internos do SGBD. Acrescentamos que está fora do escopo deste trabalho lidar com sua prevenção. Três técnicas de AM supervisionado e três de AM não supervisionado foram investigadas. XGBoost apresentou os melhores resultados para a abordagem supervisionada, com detecção precisa de ataques DDoS direcionados a bancos de dados, sem Falsos Positivos e com uma baixa taxa de Falsos Negativos, atingindo um F1-*score* de 94,44% para nosso conjunto de dados de teste. Para a DA, o *Autoencoder* apresentou resultados ligeiramente melhores que os demais algoritmos investigados, com um F1-*score* de 75,75%. Neste caso, porém, cabe ressaltar que uma escolha de um limiar diferente, a depender das necessidades de taxas de Falsos Positivos e Falsos Negativos, pode alterar a ordem dos modelos.

1.1 JUSTIFICATIVA

Os SGBDs são componentes fundamentais para o provimento de serviços, tanto para o público interno como externo de uma organização. Normalmente, são acessados por meio de aplicações desenvolvidas pela equipe da organização ou adquiridas de terceiros. Podem ser utilizados, também, para provimento de análises de dados utilizando-se ferramentas apropriadas, conhecidas como ferramentas de *Analytics* ou de

Inteligência de Negócio (*Business Intelligence* – BI).

Os SGBDs, assim como servidores de aplicação, *firewall*, *proxies* (e assemelhados), que são (ou podem ser) parte integrante de uma aplicação, apresentam vulnerabilidades de segurança. Os impactos da exploração dessas vulnerabilidades nos SGBDs podem ser catastróficos para a organização, trazendo danos à sua imagem e danos financeiros, por exemplo (35). Assim, faz-se necessário que haja cuidados para tratar possíveis incidentes de Segurança da Informação em tais componentes. A mitigação de um ataque e/ou de suas consequências é uma das formas de tratamento da vulnerabilidade.

Uma das vulnerabilidades a que estão sujeitos os SGBDs é a ataques DoS e suas variantes. Ataques desta natureza são difíceis de serem identificados por meio de capturas de pacotes, fonte de dados mais comumente utilizada para detecção de tal tipo de ataque. Isso se deve ao fato de que consultas legítimas podem ter sido encaminhadas ao banco de dados na mesma janela de tempo investigada. Nestes casos, a análise de pacotes é ineficaz, pois os pacotes obedecem fielmente ao que ditam os protocolos de comunicação. Seria necessário avaliar o impacto que essas consultas exercem nos SGBDs, ou seja, quanto de seus recursos estão sendo consumidos com as consultas aparentemente legítimas. Esta análise somente é possível utilizando-se os *logs* internos de um SGBD, pois neles encontram-se dados sobre a utilização de seus recursos computacionais.

O uso de AM Supervisionado e Não Supervisionado, este por meio da DA, é fundamental para identificação de padrões de ataques nos SGBDs. Algoritmos específicos aplicados a tais dados podem trazer informações que determinem a ocorrência do ataque ou atividade anormal que pode indicar um ataque. Em ambos os casos, os administradores passam a ter maior visibilidade sobre o SGBD no que tange à Segurança Cibernética. Assim, face à detecção de uma ataque, ficam cientes e aptos a usar de suas ferramentas disponíveis para bloquear as atividades prejudiciais.

1.2 OBJETIVOS

Esta dissertação tem como objetivo desenvolver e avaliar uma solução que faça uso de AM para analisar *logs* de sessões de consultas a bancos de dados com o intuito de identificar ataques de DDoS a SGBDs ou atividades anormais que contêm tal ataque. O trabalho faz uso de abordagem de AM Supervisionado e, posteriormente, DA.

1.2.1 Objetivos Específicos

Face aos desafios apresentados pela pesquisa e ao *dataset* obtido e utilizado no trabalho, foram definidos três objetivos específicos, ambos considerando a identificação de ataques de DDoS a SGBDs:

- Realizar uma Análise Exploratória de *dataset* obtido de um SGBD efetivamente em produção e que sofreu um ataque DDoS. Este SGBD está operacional em um órgão da cúpula do Poder Judiciário Federal e os *logs* foram coletados em tempo de ataque, sendo propriamente rotulados;
- Face ao *dataset* devidamente rotulado, utilizar-se uma abordagem supervisionada, por meio do em-

prego de algoritmos de classificação, para avaliar se é possível a identificação de ataques de DDoS a SGBD a partir da fonte de dados obtida;

- Para aplicações de mundo real, em que a maioria dos dados não se apresenta rotulado, determinar, com o mesmo *dataset*, a possibilidade de detecção de ataques a SGBD fazendo uso de abordagem de AM Não Supervisionado por meio da detecção de atividades anômalas.

Desta forma, o trabalho engloba tanto um método tradicional e com resultados muito favoráveis, demonstrados pelas métricas discriminadas no Capítulo 5, como uma abordagem mais maleável, que não depende de uma prévia classificação e conhecimento dos dados analisados. Portanto, o trabalho demonstra a viabilidade da adoção de ambas as abordagens para detecção de ataques de DDoS a este componente tão crítico de uma infraestrutura de serviços informatizados.

1.3 PUBLICAÇÕES RESULTANTES DESTA PESQUISA

Os principais resultados obtidos deste trabalho foram publicados nos Anais do XXVIII Workshop de Gerência e Operação de Redes e Serviços, Qualis B4:

- Chagas, Danilo A M et al. *Machine Learning for Detection of Distributed Denial-of-Service Attacks from Queries Executed in DBMS. In*: Anais do XXVIII Workshop de Gerência e Operação de Redes e Serviços. SBC, 2023. (2)

1.4 ORGANIZAÇÃO DO TRABALHO

Esta dissertação está organizada como se segue. O Capítulo 2 apresenta a fundamentação teórica por trás do Aprendizado de Máquina e dos ataques de Negação de Serviço. O Capítulo 3 discorre a respeito dos trabalhos relacionados e como se comparam com o nosso trabalho, identificando suas limitações e consequentes oportunidades para o nosso trabalho. O Capítulo 4 apresenta as abordagens de detecção de ataque DoS utilizando AM supervisionado e detecção de anomalias desenvolvidas em nosso trabalho. Apresenta, também, características do *dataset* obtido. O Capítulo 5 apresenta a avaliação dos resultados conseguidos com ambas as abordagens. Ao final, o Capítulo 6 apresenta as conclusões acerca da pesquisa e possibilidades de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados os conceitos utilizados neste trabalho, iniciando-se com um conceito de fundamental importância: a compreensão da mecânica de ataques de DoS. Tais ataques, mais comuns em ativos de infraestrutura que não os SGBDs, podem também afetar a infraestrutura dos servidores de bancos de dados. Como forma de habilitar sua detecção, são apresentados conceitos de aplicação do AM, tanto em sua forma supervisionada (classificação) como não supervisionada (por meio da DA) e como ele se aplica à análise dos *datasets* existentes e alvo deste trabalho. Por fim, são apresentadas métricas utilizadas para avaliação dos resultados obtidos na pesquisa.

2.1 ATAQUES DE NEGAÇÃO DE SERVIÇO

Ataques de DoS têm a intenção de exaurir os recursos da infraestrutura de um serviço computacional, tornando-o indisponível para usuários legítimos (12, 36). Assim, o atacante força à organização a violação de um dos pilares da Segurança da Informação, a Disponibilidade (37, 38, 39). Alguns ataques de DoS tornaram-se notórios ao público, especialmente aqueles concernentes a serviços utilizados por uma grande parcela de usuários. Um destes ataques foi direcionado ao GitHub, serviço responsável por armazenamento, versionamento e trabalho colaborativo em códigos-fonte de aplicações desenvolvidas no mundo todo em 2018 ilustra um DoS massivo em sua forma distribuída (DDoS), demonstrando seu potencial dano. Este ataque teve um volume de 1,35 Tbps, tornando-se o maior ataque DDoS já visto até então (37, 40). Outro ataque de alta carga foi submetido à Microsoft em 2022, o qual atingiu um volume de 3,47 Tbps em um minuto (39).

Ataques de DoS podem ser direcionados desde aos protocolos de camadas mais baixas do modelo de referência OSI e do modelo da pilha de protocolos TCP/IP – como camadas de rede e transporte –, até às mais altas – camada de aplicação (32, 41). Os ataques DoS às camadas de rede e transporte, respectivamente camadas 3 e 4 dos citados modelos, são mais comuns e, por isso, mais frequentemente investigados. Para ataques à camada de aplicação, os ataques mais frequentemente realizados – e igualmente mais estudados – são variantes de ataques ao protocolo HTTP (38, 42). Porém, é necessário fazer uma distinção mais precisa acerca do termo DoS.

Ataques de DoS podem ser classificados, quanto à origem, em clássico (ou única origem) e distribuído. O DoS clássico é um ataque no qual se tem somente uma origem. Ou seja, um atacante somente – e, portanto, somente um IP de origem (12, 36, 39, 43). Esta característica faz com que seja de fácil detecção e fácil remediação (36, 39). Para a variante distribuída, que é formalmente conhecido como Ataque Distribuído de Negação de Serviço (*Distributed Denial-of-Service* – DDoS), têm-se diversas origens de ataque (12, 36, 39, 43, 44). Para desempenhar este ataque, o atacante utiliza-se de redes zumbis, conhecidas por "*robot networks*" ou simplesmente *botnets*, em que diversas máquinas estão espalhadas por diversas localidades e infectadas por *malware*. Essas *botnets* são, então, comandadas pelos atacantes para realizar o ataque coordenado a um destino específico (44, 39). Estes ataques (DDoS) serão estudados neste trabalho

e estão representados na Figura 2.1.

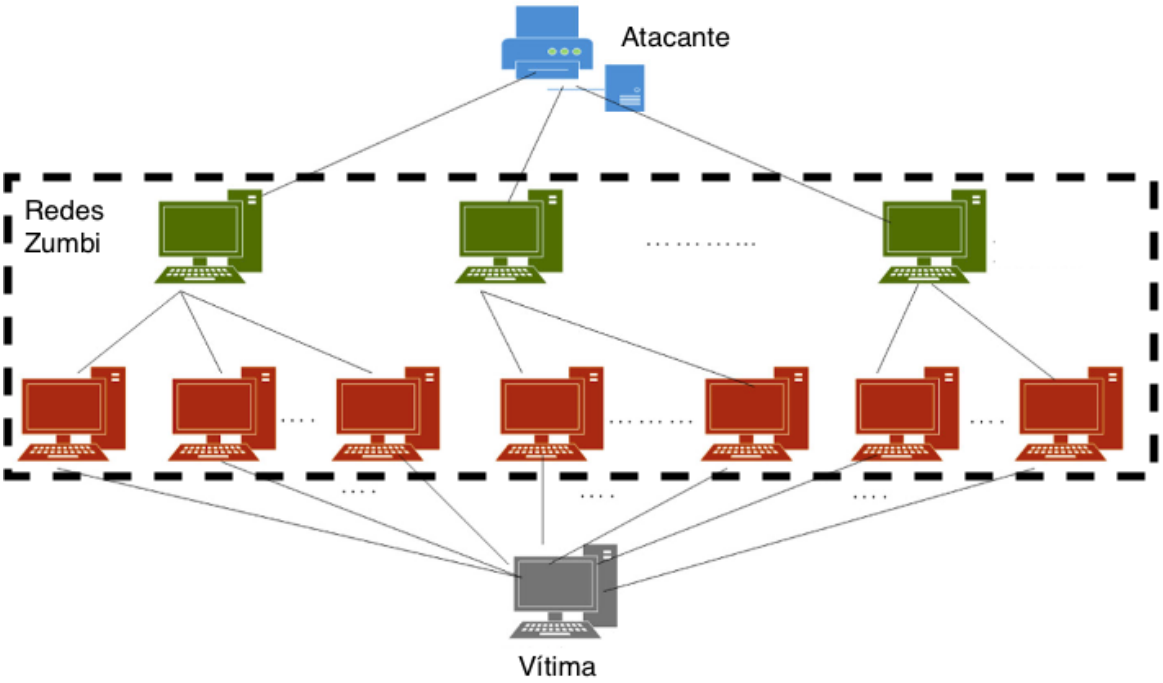


Figura 2.1: Estrutura de um ataque DDoS. Adaptado de (1)

Algumas literaturas apresentam a designação de três tipos diferentes de DDoS: baseado em volume (também conhecido por volumétrico), baseado em protocolo e baseado em Camada de Aplicação (39). Outras tratam em duas categorias maiores: volumétricos e não-volumétricos (20, 45). Porém, para ambas as vertentes, alguns conceitos são comuns e outros se sobrepõem.

A definição de ataques volumétricos é pacificada: têm o intuito de exaurir toda a banda disponível nos *links* de comunicação existentes ao enviar um volume muito alto de dados. Frente à grande carga imputada, a infraestrutura atacada, então, deixa de responder às requisições posteriores, pois sua capacidade foi exaurida completamente para atender a essa demanda excessiva (39). Esses ataques utilizam protocolos como o UDP (*User Datagram Protocol*) e ICMP (*Internet Control Message Protocol*), especialmente em ataques do tipo UDP *flooding* e ICMP *flooding*.

Porém, é possível que ataques volumétricos sejam direcionados não somente a protocolos das camadas 3 e 4, mas a protocolos de camadas mais altas. Neste ponto, temos uma sobreposição com a definição de ataques baseados em protocolo e baseados em Camada de Aplicação, a mais alta tanto do modelo de referência OSI quanto do modelo padrão da indústria, o TCP/IP. Como exemplo desses ataques, temos o ataque de amplificação de DNS (39), no qual um cliente do protocolo *Domain Name System* (DNS) - este, um protocolo de camada 7 - recebe um grade volume de respostas a consultas DNS. Estas são executadas por um atacante que se utiliza da personificação de DNS (DNS *spoofing*) para executar uma grande quantidade de consultas DNS, que são respondidas ao ente personificado, ou seja, o destinatário do ataque (36, 39).

Outro ataque volumétrico a um protocolo de camada 7 é o HTTP *flood*, direcionado a servidor *web* e/ou aplicação, nos quais algum tipo de página *web* é disponibilizada para o público. Essa disponibilização se

dá por meio do protocolo *Hypertext Transfer Protocol* (HTTP) ou sua variante segura, o *Hypertext Transfer Protocol Secure* (HTTPS). Neste ataque, o atacante envia uma quantidade de requisições HTTP/HTTPS muito acima da capacidade do provedor do serviço. Desta forma, o servidor *web* ou de aplicação atinge o limite de requisições que ele pode suportar e para de responder às demais requisições (20, 45, 46).

Quanto aos ataques não-volumétricos, encontramos em seu rol tanto ataques baseados em protocolo e baseados em Camada de Aplicação. São ataques que exploram características dos protocolos que disponibilizam serviços (46, 47). Caracterizam-se, também, por apresentar volume comum, característico de uma atividade normal do ambiente, mas também de usar características presentes em determinados protocolos (36, 42). Por conta dessas características, são mais difíceis de serem detectados. Um exemplo de ataque não-volumétrico é o ataque HTTP *slow* POST, no qual o atacante envia o corpo da mensagem HTTP POST a uma taxa extremamente baixa. A mensagem, em si, está correta e, portanto, o servidor HTTP aguarda a conclusão do envio (1). O atacante, então, executa este mesmo procedimento em várias conexões, o que faz com que o atacado atinja seu limite de conexões configuradas no serviço. Este tipo de ataque também se caracteriza por não haver consumo excessivo de largura de banda de rede e de outros recursos físicos do servidor.

2.2 SISTEMA GERENCIADOR DE BANCOS DE DADOS - SGBD

Bancos de dados são coleções de dados que possuem alguma relação entre si e que possuem um significado implícito para algum sistema e/ou processo. Um SGBD, por sua vez, é um conjunto de ferramentas que possibilita criar e gerenciar bancos de dados. São fundamentais para o provimento de serviços via *web*, pois são utilizados como repositórios dos dados pelas aplicações disponibilizadas (15).

SGBDs relacionais geralmente registram cada sessão aberta e cada consulta realizada em tabelas de *log*. Estas tabelas têm dados valiosos sobre o número de execuções de consulta, o número de registros retornados, consumo de memória, tempo de uso, tempo de espera do processador e as sessões abertas (33). Suas informações incluem o comando SQL executado, parâmetros de desempenho da consulta, o usuário que a executou e o estado da sessão. Assim, o SGBD registra parâmetros quantitativos independentemente de sua natureza, sejam elas consultas e sessões legítimas ou com propósitos maliciosos – embora, neste caso, tenham uma sintaxe coerente. Estes parâmetros podem se tornar bastante úteis para avaliação de desempenho e capacidade da infraestrutura do SGBD (13).

2.2.1 Ataques Mais Comuns a SGBDs

Os ataques mais comuns em SGBDs são Injeção SQL (SQLIA – SQL *Injection Attack*) (46). Neles, o atacante realiza a inserção de uma consulta maliciosa por meio de sistemas disponibilizados na *web* (11, 30). Para que os sistemas aceitem as consultas, os campos de entrada de dados não foram corretamente configurados, permitindo a inserção de padrões diferentes do esperado pelo *software* atacado (30). SQLIA visa o acesso não autorizado ao banco de dados, recuperação de dados sensíveis ou a indisponibilidade de tais serviços (13, 16, 17, 18). Ataques SQLIA que geram indisponibilidade são conhecidos como ataques SQL *Injection Distributed Denial-of-Service* (SIDDoS).

SIDDoS é um tipo moderno de ataque baseado na Camada de Aplicação (11, 30). Nele, o atacante utiliza-se do lado cliente (aplicação disponibilizada) para inserir consultas maliciosas, caracterizando-o como um SQLIA (30). Porém, ao invés de procurar acesso não autorizado ao banco de dados, tem o intuito de causar impacto negativo no desempenho do servidor, tornando-o mais lento até chegar ao extremo de o indisponibilizar (11)

2.2.2 Negação de Serviço em SGBDs

A anatomia de um ataque DoS/DDoS em um SGDB é ilustrada na Figura 2.2. Uma organização pode fornecer seu serviço via *software* disponível publicamente acessado por HTTP/HTTPS. Os usuários utilizam a interface fornecida e enviam solicitações HTTP/HTTPS para o *backend* do *software*, que então forma consultas (*queries*) e as envia para seu banco de dados. O SGDB executa a ação, registra a consulta e seus parâmetros em seus *logs* internos e responde para o *backend* com os dados solicitados.

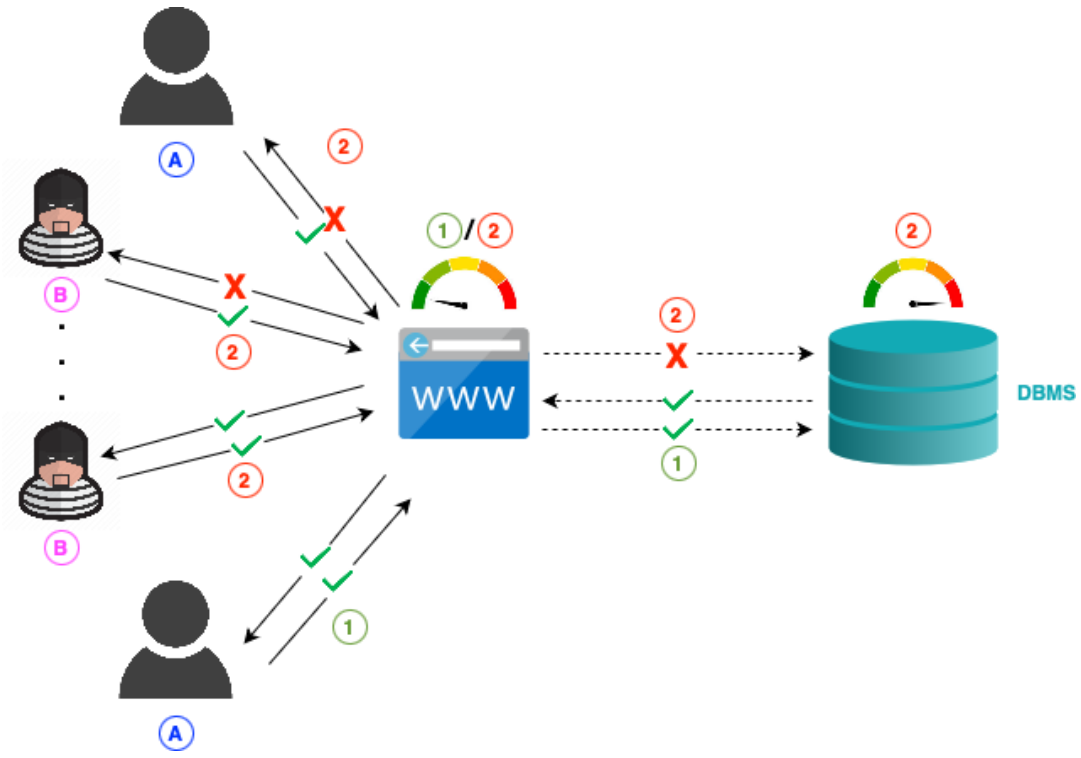


Figura 2.2: Anatomia de um ataque DoS/DDoS a um SGBD. Adaptado de (2)

Sob uso normal (identificado pelo número "1" na figura), usuários legítimos (representados pela letra "A") fazem um acesso e recebem a resposta desejada. Nessa situação, toda a infraestrutura apresenta cargas aceitáveis. No entanto, em um cenário de ataque (identificado pelo número "2"), usuários mal-intencionados (letra "B") enviam um número significativo de solicitações de uma (DoS) ou muitas (DDoS) origens (12, 36). Essas solicitações são enviadas por meio de injeção de consultas maliciosas nas aplicações e, como é de se supor, pretendem causar sobrecarga na infraestrutura do SGDB (um SIDDoS, portanto).

É notável, porém, que o servidor de aplicações (ou servidor HTTP/HTTPS) possivelmente não é afetado por essas solicitações em termos de carga. Esse fato se dá porque a quantidade de solicitações e a banda consumida para sua consecução são normais para o referido servidor. As consultas repassadas ao

banco de dados, por não serem executadas nesse servidor *web*, não causam impacto nele. Este caso, se configura em um ataque de baixa taxa, não volumétrico, no qual o tráfego se assemelha a um legítimo (42, 47).

Como consequência do cenário descrito, a maior parte do processamento exigido pelos usuários mal-intencionados é executada no SGDB. Assim, a infraestrutura do SGDB fica saturada e, posteriormente, não consegue lidar com as solicitações vindouras. Quando tal situação acontece, todos os usuários subsequentes deixam de receber uma resposta do SGDB, a esta altura sobrecarregado, embora a parte HTTP/HTTPS da infraestrutura do *software* ainda esteja responsiva.

2.3 APRENDIZADO DE MÁQUINA

AM é um campo da Inteligência Artificial que permite que computadores aprendam com dados sem serem explicitamente programados para isso. AM tem sido usado para resolver problemas que não podem ser facilmente resolvidos por abordagens tradicionais, ou algoritmos ainda precisam ser desenvolvidos para resolvê-los (21). Como exemplos não exaustivos de seu uso, temos reconhecimento de voz e imagem, classificação de mensagens indesejadas (*spam*), sistemas de recomendação de produtos e precificação de ativos como imóveis ou ações. Outra área que se beneficia enormemente de AM é a detecção e prevenção de ataques cibernéticos, nos quais tanto DoS quanto DDoS são encontrados (5, 22, 23).

AM pode depender de dados coletados de várias partes de uma infraestrutura (48). Os modelos são primeiramente treinados com esses dados coletados (49). Com base neste modelo treinado, é possível identificar padrões (ou relações matemáticas) representando informação. Por exemplo, ataques DDoS tendem a ter o mesmo padrão (ou um muito próximo) dependendo dos dados analisados. No entanto, a eficiência de um algoritmo de AM depende fortemente de quão bem ele pode representar os dados de entrada. Dados de treinamento de baixa qualidade levam a representações imperfeitas e, consequentemente, menor desempenho das técnicas de AM (4).

2.3.1 Aprendizado Supervisionado

No Aprendizado Supervisionado, os dados utilizados para treinar os modelos já possuem uma solução desejada, conhecida como rótulo ou *label* (21) – esta variável rótulo também é conhecida como variável alvo (*target*). Por meio do Aprendizado Supervisionado, pode-se resolver dois tipos de problemas: classificação ou regressão.

A classificação diz respeito a determinar a categoria de um dado contido no conjunto de dados analisados (ou seja, classificá-lo em uma categoria) a partir de suas características. Essas características ou *features* são também conhecidas como variáveis preditoras. A categoria, por sua vez, pode ser binária (0 ou 1, verdadeiro ou falso) ou multiclasse (21). Este último resolve o problema de classificar os dados em mais de uma classe. Por exemplo: a classificação de um carro por seu formato pode indicar carros do tipo *hatchback*, sedã, picape etc. Estas são as suas classes.

A regressão consiste em prever o valor numérico da variável alvo a partir das características da obser-

vação (21). Sua utilização se dá para qualquer problema em que um valor numérico necessita ser previsto. Por exemplo, a partir de algumas características de um carro (consumo, quilometragem, ano de fabricação etc.) prever seu preço de revenda. Pode-se utilizar a regressão para problemas de estimativa de consumo e otimização de energia, previsão de necessidades de manutenção de equipamentos, definição de valores de imóveis, previsão de tráfego de uma rede de computadores etc.

A Regressão Logística, apesar de seu nome denotar ser um algoritmo de regressão, é utilizada para resolução de problemas de classificação (2, 21). Seu uso se dá para estimar a probabilidade de determinada instância de dados pertencer a uma categoria em particular. Embora possa ser utilizada para atacar problemas multiclasse (previsão de mais de duas classes), seu maior uso é para classificação binária – caso idêntico ao tratado no nosso trabalho. Se o percentual predito é maior que 50%, então o modelo classifica aquela instância como pertencente à classe alvo. Esta é também chamada de classe positiva ou *positive class* e recebe o rótulo 1. Abaixo deste patamar, a instância é classificada como 0 (classe negativa ou *negative class*) (21). Assim, adapta-se bem ao objetivo de nosso trabalho, que é determinar se uma instância é um ataque (classe positiva e, portanto, valor 1) ou não ataque (classe negativa ou 0).

Modelos baseados em árvores são bastante utilizados tanto para classificação como para regressão. Consistem em segmentar os dados em regiões de acordo com a média (para variáveis alvo contínuas) ou a moda (variáveis alvo categorizadas) e utiliza-se de medidas de erro de classificação ou regressão. Adiciona-se que são modelos conceitual e computacionalmente simples (3). Dentre esses algoritmos, temos o Floresta Aleatória também conhecido, em inglês, como *Random Forest*.

O Floresta Aleatória particiona as variáveis preditoras em regiões mutuamente exclusivas, gerando árvores de decisão para cada conjunto particionado. Ao final, os resultados de todas as árvores são combinados de forma a se trazer a classificação ou regressão otimizada para o modelo (3, 21). A Figura 2.3 apresenta um exemplo de uma árvore de decisão em um processo de classificação binária. Esta árvore procura identificar um ataque a partir de quatro variáveis do *dataset*: avg_user_count (média de vezes que o usuário promove uma interação com o banco, provocando uma mudança de estado da sessão), avg_mem_consump _per_user (média de memória do SGBD consumida por usuário), avg_cpu_consump_per_user (média de consumo de CPU do SGBD por usuário, int_btw_sessions (intervalo entre sessões abertas no SGBD por usuário). Assim, tem-se uma visão clara do processo de decisão do algoritmo de acordo com os valores das variáveis. Esses limiares examinados são escolhidos pelo algoritmo de acordo com as características dos dados apresentados. Ao final, a instância dos dados é classificada como "Normal"ou "Ataque".

Um algoritmo que tem tido grande sucesso em competições de AM é o *Extreme Gradient Boosting* (XGBoost) (21, 50). Ele alia o método de *Boosting*, que consiste em treinar preditores sequencialmente, cada um corrigindo seu predecessor (21), às Árvores de Decisão. Foi desenhado para ser extremamente rápido, escalável e portável. Assim como as Árvores de Decisão, pode ser utilizado tanto para resolver problemas de regressão como problemas de classificação.

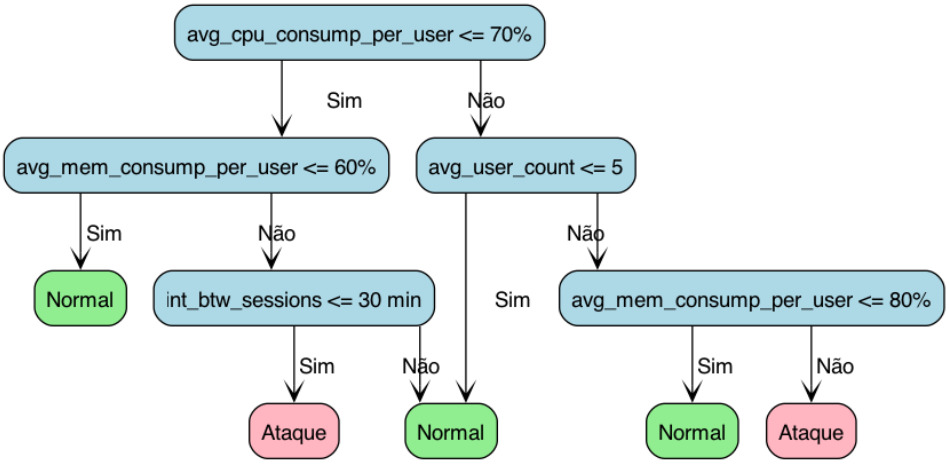


Figura 2.3: Exemplo de Árvore de Decisão de classificação binária.

2.3.2 Aprendizado Não Supervisionado

Embora a maior parte das aplicações de AM seja baseada em aprendizado supervisionado, a maioria dos dados disponíveis no mundo real não apresenta rótulos (21, 51) – estes, condição imprescindível para termos o aprendizado supervisionado. Neste contexto que entra o aprendizado não supervisionado. Neste tipo de aprendizado, o objetivo dos modelos e algoritmos é identificar padrões ou estruturas nos dados sem ter conhecimento prévio de suas categorias ou rótulos (51, 52).

Ao contrário do versão supervisionada, o aprendizado não supervisionado não necessita de ação externa humana para atribuição de rótulos que determinem uma característica alvo de cada instância dos dados. Desta forma, torna-se a melhor opção para analisar dados desconhecidos ou, ainda, não processados previamente (51).

Aprendizado não supervisionado apresenta-se em diferentes formas. Dentre elas, temos Agrupamento (*Clustering*), Redução de Dimensionalidade (*Dimensionality Reduction*), DA, *Autoencoders* (3). Estes tipos são não exaustivos e foram escolhidos por suas características serem compatíveis com o nosso trabalho.

Agrupamento, mais conhecido pelo seu termo em inglês *clustering*, é uma técnica para identificar instâncias similares e, então, associá-las a um grupo (*cluster*) com componentes com as mesmas características (21). Essas características são baseadas em uma medida de distância (por exemplo, distância Euclidiana), de modo que cada grupo contenha elementos em que a distância é pequena (3). Trata-se de uma técnica não supervisionada, na medida em que não há a necessidade de rótulos para realizar esses agrupamentos (3, 21). Esta técnica pode ser utilizada para diversas finalidades: segmentação de imagens, análise de dados, redução de dimensionalidade (3), DA (21, 53, 54) etc.

A Redução de Dimensionalidade consiste em reduzir a quantidade de dimensões do *dataset*, também conhecidas como características, *features* ou variáveis. Essa redução se dá com base na estrutura de dependência dessas variáveis, ou seja, consiste em encontrar poucos fatores que conservem a estrutura de covariância das variáveis (3). Estes poucos fatores podem substituir as variáveis do *dataset*, e, por sua vez, podem ser utilizadas como variáveis preditoras para resolução de problemas de regressão ou de classificação. O algoritmo mais amplamente utilizado é o *Principal Component Analysis – PCA* (3, 21).

2.3.3 Aprendizado Profundo

O Aprendizado Profundo (AP) - do inglês *Deep Learning* -, também conhecido como aprendizado de representação, é uma abordagem considerada um subdomínio do AM. Modelos de AP têm a capacidade de realizar processamento não linear para aprender múltiplos níveis de representações de dados, distinguindo-o de técnicas tradicionais de AM. Esta abordagem tem sido utilizada com sucesso em várias aplicações, como classificação de imagens, visão computacional e segurança cibernética (5).

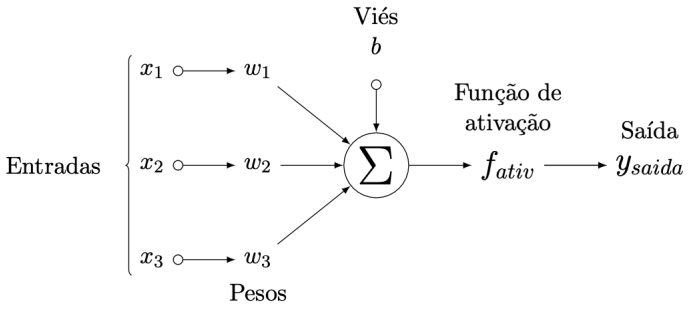
Tem seus primórdios em trabalhos como o de McCulloch-Pitts (MCP) de 1943, em que descreve um modelo matemático baseado nos neurônios biológicos. Este modelo é projetado para imitar funcionalmente o cérebro humano, empregando uma combinação de algoritmos e fórmulas matemáticas (5). Em 1958, Rosenblatt introduziu o "*perceptron*", cuja ideia é atribuir pesos aos dados de entrada iterativamente até se atingir uma precisão predeterminada para a tomada de decisão. Nele, aos valores de entrada são multiplicados pesos w_i, somados a valores b chamados vieses e, após, aplicada, sobre este resultado, uma função conhecida por "função de ativação" (3).

As Redes Neurais (RN) são algoritmos construídos para identificar relações matemáticas entre as variáveis de um *dataset* utilizando um processo que imita o cérebro humano, mais especificamente a interação entre seus neurônios. Têm o intuito de receber dados de entrada, estabelecer tais relações e prover uma saída adequada (3, 55, 56). Os *perceptrons* são um exemplo de uma RN em que há somente uma camada de entrada e uma camada de saída (Figura 2.4a). Temos também as RNs multicamadas e que também são conhecidas por rede do tipo proalimentada (*feed forward*). Possuem *n* camadas escondidas, as quais recebem dados vindo das camadas precedentes como suas entradas e entregam suas saídas para as camadas subsequentes, como se vê na Figura 2.4b.

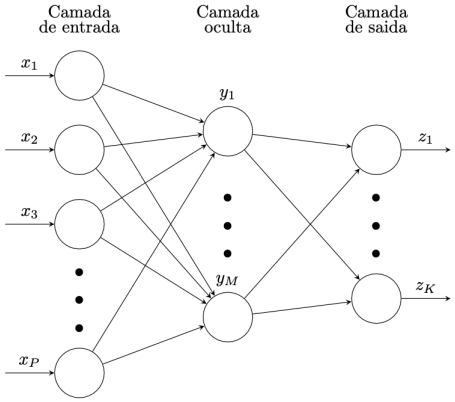
Além dessas duas instâncias de RNs, temos também as Redes Neurais Recorrentes (RNR) - Figura 2.4c, em que os neurônios da camada precedente se conectam aos das camadas antecedentes, criando um ciclo de retroalimentação (3, 5, 55, 56). Por poderem utilizar o equivalente a uma memória em seus neurônios, podem ser usadas para processar sequências de comprimentos variáveis (entradas). Como benefício dessa característica, podem ser aplicadas para resolução de problemas sequenciais, como, por exemplo, de reconhecimento de escrita e voz e séries temporais (3, 24, 57).

As RNRs, porém, são difíceis de implementar, pois são sensíveis aos chamados problemas do gradiente evanescente (*vanishing gradient problem*) e problema do gradiente explosivo (*exploding gradient problem*) (31). Os gradientes obtidos pelo mecanismo de retroalimentação podem decair ou explodir exponencialmente devido a multiplicações de derivadas grandes ou pequenas, cumulativamente. Essa sensibilidade faz com que a RN "esqueça" as entradas iniciais a partir das novas entradas apresentadas (3, 4). A Figura 2.4 apresenta um resumo das RNs.

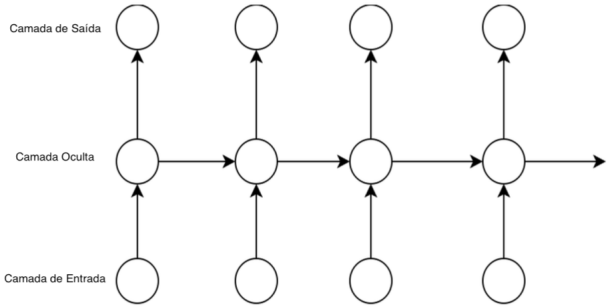
Os ajustes de modelos de RNs baseia-se na minimização dos erros encontrados na camada de saída. Para um problema do tipo regressão, a função de perda examinada é a soma dos quadrados dos erros (*sum of squared errors* – SSE), ao passo que, para classificação, utiliza-se a entropia. Em ambos os casos, o ajuste da RN (de seus pesos e vieses) tem por base um algoritmo chamado de retropropagação (*backpropagation*). Nele, o gradiente da função de perda escolhida é utilizado para atualizar os pesos precedentes, até que um valor mínimo de perda seja alcançado (3, 56). Desta forma, o modelo é otimizado para alcançar os melhores



(a) *Perceptron*



(b) RN Multicamadas



(c) RNR

Figura 2.4: Exemplos de RNs. (a) *Perceptron*, (b) RN multicamadas e (c) RN Recorrente. Adaptado de: (3, 4)

valores possíveis de acordo com a quantidade de neurônios e camadas definidos no modelo.

Os *Autoencoders* são uma classe de RN não supervisionada que tentam recriar, como seu alvo e por meio da retropropagação, os dados de entrada na RN. Consiste em duas partes: *encoder* e *decoder*. O *encoder* recebe dados de entrada, reduz a dimensão desses dados em camadas posteriores (compressão) e os entrega para o *decoder*. Este, por sua vez, reconstrói os dados aumentando sua dimensão (descompressão), de forma a reconstruir os dados entrados inicialmente (5, 55, 56). Podem ser aplicados em casos como reconstrução de imagens, agrupamentos, DA, dentre outros. Esta arquitetura está representada na Figura 2.5.

Portanto, um *Autoencoder* possui duas RNs em cascata, sendo a primeira a rede *encoder* e a segunda, a *decoder*. A *encoder* consiste em receber os dados de entrada x e transformá-los em um sinal codificado h:

$$y = h(x)$$

A parte *decoder* recebe os dados codificados y e aplica uma transformação f para conseguir o sinal reconstruído r:

$$r = f(y) = f(h(x))$$

Ao final, calcula-se o erro e pela diferença entre os dados de entrada x e os dados reconstruídos r:

$$e = x - r$$

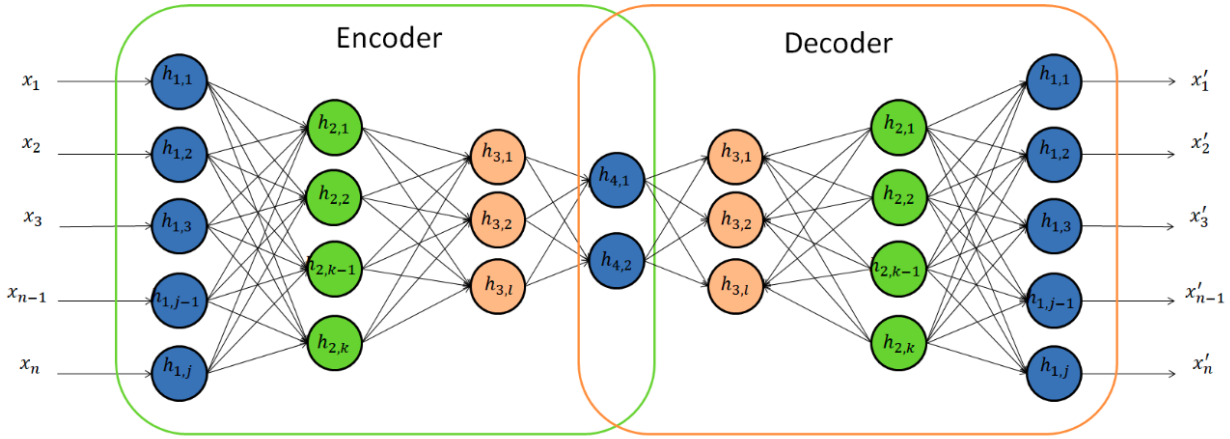


Figura 2.5: Estrutura de um *Autoencoder*. Fonte: (5)

A partir do erro e por meio do mecanismo de retropropagação, a RN aprende ao reduzir a função de erro e propagar os novos pesos e vieses para as camadas ocultas da rede (55).

Existe um tipo de RNR que é capaz de "aprender" dependências de longo prazo e resolver os problemas do gradiente evanescente e do gradiente explosivo. Essa rede chama-se rede com memórias de curto e longo prazos ou *Long Short-Term Memory* (LSTM) e, como seu nome diz, modelam, simultaneamente, memórias de curto e longo prazo (3) ao prover blocos de memória em suas conexões recorrentes (4). Cada bloco de memória, chamado célula (*cell*), está conectado por meio de camadas e a informação contida nas células é regulada por meio de portas (*gates*) – de entrada, de esquecimento e de saída –, por meio de funções de ativação sigmoides e tangentes hiperbólicas (3, 29). Sua arquitetura está representada na Figura 2.6.

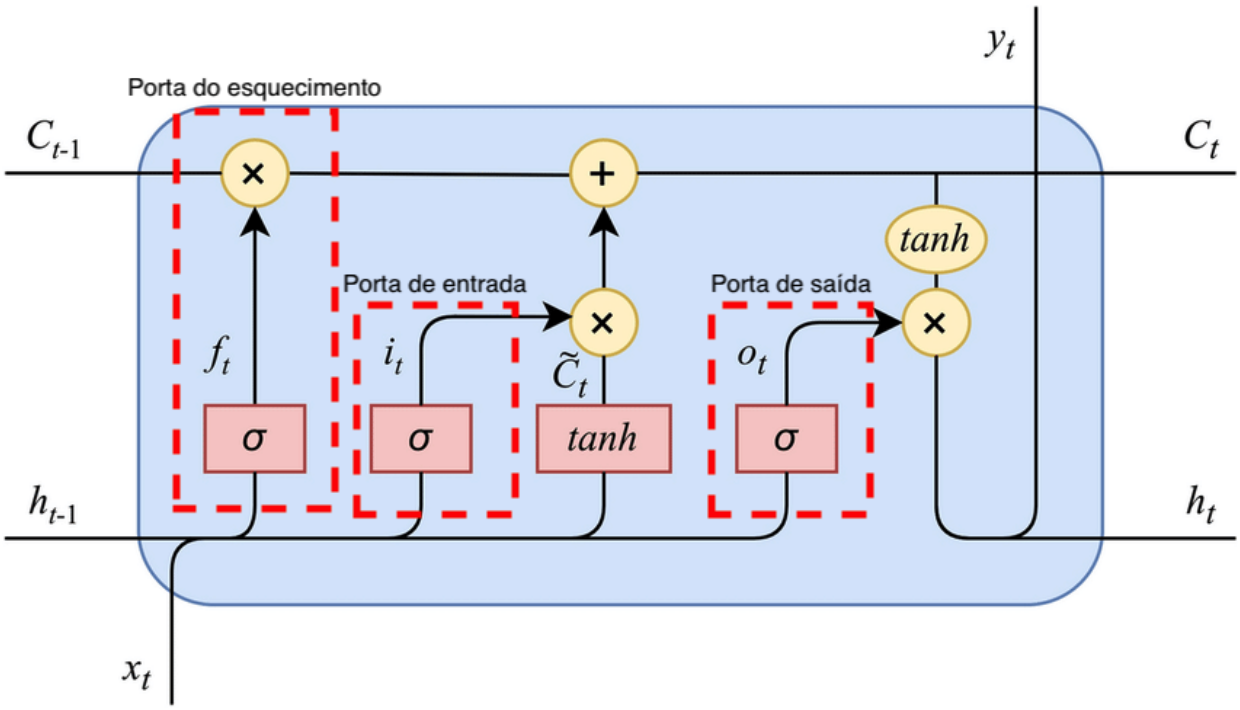


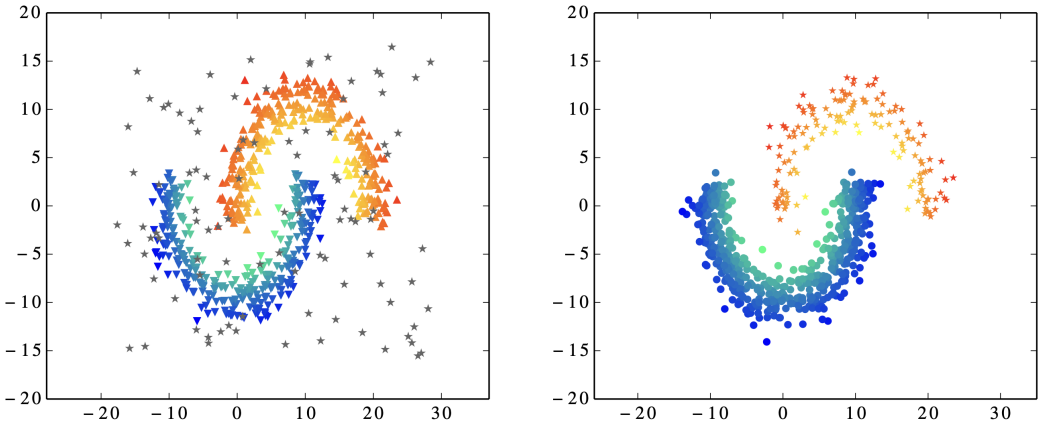
Figura 2.6: Célula de uma rede LSTM. Adaptado de (3)

2.3.4 Detecção de Anomalias

Detecção de Anomalias (DA), também conhecido por *Anomaly Detection* (AD), trata da identificação de padrões ou eventos que se desviam do comportamento normal (24, 25, 28). Envolve modelar o comportamento padrão dos dados durante a fase de treino e rotular aqueles dados que não estão em conformidade com o padrão (58) – estas são as anomalias. Essa "rotulação" ocorre mesmo sem haver uma noção predefinida do que constitui uma anomalia. Por conta desta característica, torna-se bastante útil em aplicações de segurança cibernética (57).

As anomalias, de acordo com a literatura clássica, podem ser classificadas em três tipos distintos: anomalias de ponto (*point anomalies*), anomalias condicionais ou contextuais (*conditional or contextual anomalies*) e anomalias de grupos (*group anomalies*). As anomalias de ponto são instâncias individuais que são consideradas anômalas com relação a outras instâncias individuais. As anomalias condicionais são instâncias individuais anômalas, mas que estão inseridas em um contexto ou condição particular – estas podem ser consideradas normais quando inseridas em outro contexto. Por fim, as anomalias de grupos são um subconjunto de instâncias de dados que são anômalos como um todo quando comparados a outras instâncias de dados (6, 7, 27, 57).

A Figura 2.7 representa os casos de anomalias de pontos e anomalias de grupos. Na figura 2.7a, temos anomalias de pontos, que são aqueles que não pertencem aos grupos de pontos. Na figura 2.7b, temos anomalias de grupos, pois os pontos agrupados em um dos grupos representam anomalias para aquele *dataset*.



(a) *Anomalias de pontos.* (b) Anomalias de grupos.

Figura 2.7: Anomalias de pontos e anomalias de grupos. Adaptado de (6)

A Figura 2.8 demonstra visualmente uma anomalia condicional ou contextual. Nela, estão apresentados dois contextos, sendo que os pontos anômalos são aqueles com menor valor dentro do contexto. Nota-se que há pontos no "Contexto 2" que estão abaixo da anomalia do "Contexto 1", mas que não são considerados anomalias, pois há um ponto de menor valor naquele contexto.

Considerando a disponibilidade de rótulos no *dataset*, a DA pode ser dividida em três configurações: DA Não Supervisionada, DA Semi-supervisionada, DA Supervisionada (57). Estes conceitos encontram paralelo nos conceitos semelhantes de AM já apresentados aqui.

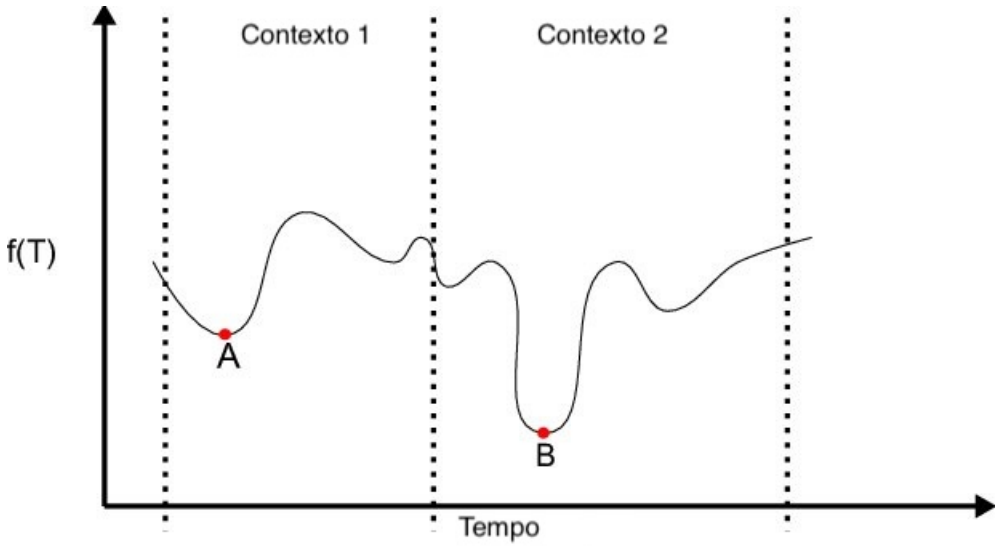


Figura 2.8: Anomalia condicional ou contextual. Adaptado de (7)

As raridades e classes desbalanceadas representam desafios para a DA. Anomalias são tipicamente instâncias raras nos dados – ou seja, os dados tendem a apresentar uma quantidade muito grande de instâncias normais e uma pequena fração de anomalias. Devido a essa característica, torna-se difícil – algumas vezes impossível – reunir quantidade significante de instâncias classificadas previamente como anormais (ou anômalas). Essa característica dificulta o treinamento de modelos que conseguem efetivamente detectar anomalias (27). As classes desbalanceadas, por sua vez, são uma consequência da raridade, visto que se tem uma classe (classe negativa) com um número muito maior de instâncias que a outra (classe positiva ou anomalia).

Considerando-se essa raridade e o fato de se tratar de detecção de ataques a um componente crítico da infraestrutura para provimento de serviços, uma classificação errônea de anomalias é usualmente mais problemática do que uma classificação errônea de observações normais (27). Isso equivale a dizer que uma atividade normal classificada como anômala não traz tantos prejuízos como uma atividade anômala sendo classificada como normal. Neste último caso, um ataque está sendo bem-sucedido, ao passo que no primeiro caso haverá a indicação de um suposto ataque, mas que efetivamente não se concretizou. Este fato demonstra a importância de se detectar com alta eficácia as anomalias.

A existência da DA não está condicionada à aplicação de Inteligência Artificial e de suas subáreas. Ela pode ser obtida por meio de abordagens estatísticas, como domínio de conhecimento e sistemas baseados em regras, por exemplo (59). Porém, o uso de AM permite maior flexibilidade e maior adaptabilidade à evolução dos padrões de dados (53). Desta forma, este trabalho faz uso de modelos de AM para a DA no *dataset* analisado.

O crescente uso de AP, bem como a evolução dos algoritmos deste tipo de aprendizado, tem levado a importantes progressos na DA (57). Estes modelos têm apresentado avanços significativos em comparação aos métodos tradicionais de AM em se tratando de DA. Esse fato se deve à habilidade desses modelos de aprender padrões intrincados a partir de uma grande quantidade de dados (57).

Ao se executar um algoritmo de AM ou de AP para DA, estes não trazem diretamente a informação de

que aquela observação é uma anomalia ou não. Ao invés disso, é computada uma pontuação (ou *anomaly score*) para aquela instância de dados, sendo que esta pontuação define seu grau de desvio em relação a uma instância normal. Quanto maior o valor desta pontuação, mais anômalo o dado se apresenta em relação aos demais contidos no *dataset* (57). Para se transformar essa medida contínua em uma medida binária (normal ou anômalo), um valor de limiar (ou seu termo em inglês *threshold*) deve ser estabelecido. Este pode ser definido como um valor de corte que determina que, a partir daquele valor de pontuação, os dados são classificados como atividade anômala. A fórmula para determinação binária de anomalia, de acordo com o limiar ζ é a seguinte:

$$Anomalia = \begin{cases} Normal, & Pontuação\,(x) < \zeta \\ Anomalia, & Pontuação\,(x) \geq \zeta \end{cases}$$

A definição do limiar mais apropriado encontra diferentes formas na literatura. Pode-se, por exemplo, defini-lo com base no conjunto de validação para alcançar uma taxa de falso positivo pré-definida. Neste caso, o limiar é determinado de tal forma que a fração de pontos de dados de validação que ultrapassam esse limiar se alinhe com a taxa de falso positivo desejada (53). Outra forma de se estabelecer este limiar é de acordo com o percentual de instâncias anômalas do *dataset*, caso este valor seja conhecido. Assim, qualquer observação que esteja em um percentil igual ou superior ao limiar será considerada anômala (60).

O limiar na DA, portanto, não é apenas um valor numérico, mas um parâmetro crítico que influencia a eficácia do processo de detecção. Sua determinação, qualquer que seja a forma com que é escolhida, desempenha um papel fundamental para garantir o equilíbrio entre as taxas de verdadeiros positivos, falsos positivos e falsos negativos (53).

A Floresta de Isolamento (do inglês *Isolation Forest*, também conhecida como *iForest*) é um algoritmo eficiente para DA. É baseado em Árvores de Decisão, como o Floresta Aleatória, no qual são construídas várias árvores de decisão a partir de variáveis escolhidas aleatoriamente. Este algoritmo escolhe, também de forma aleatória, os limiares mínimo e máximo para dividir em dois o *dataset* criado do processo de escolha aleatória de variáveis. O algoritmo estabelece que as anomalias são aqueles pontos que estão distantes das demais obtidas. Na média, esses pontos tendem a ser isolados em menos passos que as instâncias consideradas normais (21). Este algoritmo também reside em uma pontuação de anomalia para determinar se um ponto é anômalo ou não. Sua fórmula é definida como a seguir:

$$s(x, n) = 2^{-}\frac{E(h(x))}{c(n)}$$

Onde $h(x)$ é o tamanho do caminho da observação x, $c(n)$ é o caminho médio de busca bem-sucedida em uma árvore de busca binária e n é o número do nó externo.

O processo está ilustrado na Figura 2.9.

Figura 2.9: Processo de Árvores de Decisão para isolamento de anomalias no algoritmo iForest.

Por suas características de compressão e posterior reconstrução das informações, os *Autoencoders* são muito usados para DA. Essas características, por meio do erro de reconstrução, permitem a comparação dos dados reconstruídos em relação aos dados de entrada no *encoder*. Desta forma, caso os dados reconstruídos pelo *decoder* apresentem um erro maior que um limiar determinado, são considerados anomalias (26, 57).

Os erros de previsão em relação ao esperado para se detectar anomalias se aplicam às LSTMs. Como este é um modelo sequencial (também conhecido como *sequence-to-sequence* ou *seq2seq*), ele utiliza o modelo treinado para prever os novos dados apresentados. O erro da previsão em relação aos dados observados é usado em comparação ao limiar de detecção: caso seja maior que o limiar, os dados são classificados como anomalias (57).

Para avaliar a eficácia de algoritmos de AM e de DA, é necessária a submissão das previsões obtidas a algumas métricas. Porém, de forma a compreender o significado dessas métricas, é necessário tratar do conceito de Matriz de Confusão ou *Confusion Matrix*. A Matriz de Confusão representa o resumo dos resultados obtidos na classificação em relação aos valores da variável alvo (31). Num classificador binário, os valores preditos são "0" ou "1", verdadeiro ou falso. Estes valores são comparados com aqueles referentes ao valor real da variável alvo (conceito este conhecido como *ground truth*). Desta forma, traça-se uma tabela com os valores preditos *versus* os valores reais. Esta representação está presente na Tabela 2.1:

Tabela 2.1: Matriz de Confusão

		Vl. Previsto	
		Positivo	Negativo
Vl. Real	Positivo	VP	FN
	Negativo	FP	VN

A definição dos valores constantes na tabela estão abaixo: (31):

- Verdadeiro Positivo (VP): valor predito positivo e valor real positivo;
- Falso Negativo (FP): valor predito negativo (falso ou "0") e valor real positivo;
- Verdadeiro Negativo (VN): valor predito negativo e valor real negativo;
- Falso Positivo (FP): valor predito positivo e valor real negativo.

Em termos de métrica, as mais comumente utilizadas na literatura para avaliação, em problemas de

aprendizado supervisionado, são Acurácia (*accuracy*), Precisão (*precision*), Sensibilidade (também conhecido por *recall*), Especificidade (*specificity*), F1-*score*, taxa de falsos positivos (*False Positive Rate* – FPR) e ROC-AUC (*Receiver Operator Characteristics - Area Under the Curve*) (2, 26, 31, 61).

- Acurácia: identifica quão bem o modelo previu corretamente a classe positiva. Sua fórmula está descrita abaixo:
$$Acurácia = \frac{VP + VN}{VP + VN + FP + FN}$$
- Precisão: identifica, dentre todas as instâncias previstas como verdadeiras, quais realmente o são. É representada pela seguinte fórmula:
$$Precisão = \frac{VP}{VP + FP}$$
- Sensibilidade: também chamada *recall* ou *true positive rate* (TPR), representa as instâncias positivas corretamente preditas pelo modelo. Sua fórmula é definida como:
$$Sensibilidade = \frac{VP}{VP + FN}$$
- F1-*score* é uma forma comum de se comparar classificadores; combina precisão e sensibilidade em uma única métrica – é a média harmônica de precisão e sensibilidade:
$$F1\text{-}score = 2 \cdot \frac{Precisão \cdot Sensibilidade}{Precisão + Sensibilidade}$$
- A FPR está relacionada às instâncias negativas classificadas incorretamente como positivas e é representada pela seguinte fórmula:
$$FPR = \frac{FP}{FP + VN}$$
- A curva ROC-AUC é uma outra forma de se comparar classificadores e uma das principais, pois tem a habilidade de avaliar o desempenho do modelo de acordo com diferentes limiares de decisão (2, 57, 61). Na prática, traz uma identificação visual de quão bem um modelo pode diferenciar uma classe da outra. Neste gráfico, são traçados os pontos da Sensibilidade (TPR) em relação à taxa de falsos positivos (FPR), também conhecida pela fórmula $1 - Sensibilidade$. Esta métrica é medida em porcentagem, sendo que o valor de 50% representa um classificador aleatório. Quanto mais próximo de 100%, melhor é o resultado do modelo (2, 62).

2.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou conceitos aplicados neste trabalho e que são fundamentais para seu entendimento. Introduziu os conceitos de DoS, em especial sua variante mais perigosa, a Distribuída (DDoS), principal foco desse estudo. Viu-se, também, qual o papel de um SGBD para o provimento de serviços ao público-alvo de uma organização. Como consequência dessa importância, torna-se um ponto crucial

da infraestrutura para aplicação de técnicas de defesa contra ataques cibernéticos. Dentre as técnicas mais avançadas, a aplicação do AM tem ganhado maior importância e espaço tanto nas pesquisas acadêmicas quanto nas implementações nas instituições. Esse crescimento dá-se pelo fato de sua adaptabilidade à quantidade enorme de dados que são gerados e sua facilidade de estabelecer relacionamento entre as variáveis desses dados. Pode ser utilizado tanto para dados previamente classificados como para dados nunca vistos antes. Essa maleabilidade possibilita, então, extrair informações importantes dos dados, como, por exemplo, se um ataque está em andamento ou se uma atividade é suspeita. Ademais, uma correta avaliação dos resultados deve estar presente, por meio de métricas adequadas e do que esses resultados podem representar para a solução.

Para o próximo capítulo, serão apresentados trabalhos que embasaram as ideias contidas neste estudo. Tais trabalhos uniram os conceitos aqui demonstrados de forma a se gerar soluções para aplicação à segurança cibernética, demonstrando a efetividade dessa abordagem heurística.

3 TRABALHOS RELACIONADOS

Neste capítulo, são apresentados trabalhos similares na literatura e de que forma se relacionam com o que foi desenvolvido neste trabalho. O método de pesquisa utilizado para obtenção da bibliografia analisada também é apresentado. Por fim, tem-se uma comparação entre métricas para quantificar os trabalhos correlatos e o trabalho aqui desenvolvido.

Conforme discutido no Capítulo 2, os SGBDs são ferramentas críticas e, portanto, de extrema importância para provimento dos serviços de uma organização para seu público-alvo. Por carregarem tamanha importância, estão sujeitos a ataques de várias naturezas, incluindo aí a ataques DoS e variantes. Faz-se, portanto, necessária a proteção de tal item da infraestrutura. As ferramentas mais maleáveis descritas na literatura empregam algum tipo de AM, seja de maneira supervisionada, seja não supervisionada. Neste último exemplo, a DA tem ganhado espaço e atenção acadêmica – e é neste contexto que se insere nosso trabalho.

Neste capítulo, apresentaremos trabalhos que fazem a detecção de ataques a SGBDs na Seção 3.2. Posteriormente, avançaremos para a análise de trabalhos que detectam DoS e variantes em SGBDs na Seção 3.3 e faremos a discussão desses trabalhos na Seção 3.4.

3.1 MAPEAMENTO SISTEMÁTICO

Para levantamento dos artigos sobre os quais trabalhamos as ideias do nosso trabalho, realizamos pesquisas de artigos em bases de dados amplamente conhecidas, como ACM *Library*, IEEE e, em última instância, Google Scholar. As *strings* de busca que nos permitiram obter os artigos usados neste trabalho foram as seguintes:

- *database* AND dos AND data de publicação: últimos 2 anos
- *database* AND ddos AND data de publicação: últimos 2 anos
- *database* AND siddos AND data de publicação: últimos 2 anos
- dbms AND dos AND data de publicação: últimos 2 anos
- dbms AND ddos AND data de publicação: últimos 2 anos
- dbms AND siddos AND data de publicação: últimos 2 anos
- dbms AND *denial-of-service* AND data de publicação: últimos 2 anos
- *database* AND *denial-of-service* AND data de publicação: últimos 2 anos
- *database* AND *attack* AND *machine learning* AND data de publicação: últimos 2 anos
- *database* AND *attack* AND *anomaly detection* AND data de publicação: últimos 2 anos

- dbms AND *attack* AND *machine learning* AND data de publicação: últimos 2 anos
- dbms AND *attack* AND *anomaly detection* AND data de publicação: últimos 2 anos
- *attack* AND *anomaly detection* AND data de publicação: últimos 2 anos

Escolhemos o marco temporal de dois anos, para as publicações pesquisadas, de forma a conseguirmos levantar os algoritmos e técnicas mais atualizados da literatura para detecções de ataques desta natureza. Após uma análise preliminar, cerca de cento e cinquenta artigos foram escolhidos inicialmente, sendo que foram encontrados poucos artigos (cinco, mais precisamente) que abordassem, simultaneamente, a detecção de ataques de DoS a SGBDs. Diante deste número reduzido de artigos, expandimos as consultas e retiramos a delimitação de tempo de cada uma delas. Desta forma, conseguimos chegar a artigos publicados em anos anteriores que exploravam tais ataques e podemos perceber o quanto a introdução de novas técnicas de AM permitiram à pesquisa evoluir.

Frisa-se, entretanto, que a consulta inicial nos trouxe rica literatura sobre o uso de AM para detecção de ataques em geral. Este fato possibilitou-nos compreender melhor os mecanismos de ataques cibernéticos e, especialmente, de como o AM poderia ser útil para o nosso trabalho. Outros conceitos que são abordados no trabalho puderam ser propriamente compreendidos, como técnicas de treinamento de modelos, balanceamento de classes, funções de ativação, conceitos como *underfitting* e *overfitting* e como a quantidade de dimensões do *dataset* poderiam impactar na capacidade de generalização do modelo, técnicas para validação de novos *datasets* em relação ao modelo treinado etc. Toda esta pesquisa agregou conhecimento que não só pôde ser utilizado no nosso trabalho como tornou-se imprescindível para sua consecução.

De forma a enriquecer e solidificar conhecimento e conceitos utilizados no trabalho, optamos por escolher livros consagrados na literatura, escritos por autores proeminentes. Destes, extraímos conceitos basilares para o trabalho, a partir dos quais se possibilitou a união de conhecimentos para o desenvolvimento da nossa solução.

3.2 DETECÇÃO DE ATAQUES A SGBD

Fonseca et al. (63) avaliam que os mecanismos tradicionais de segurança de bancos de dados muitas vezes não são suficientes diante de ataques sofisticados. Eles fornecem recursos essenciais como autenticação, autorização e auditoria. No entanto, nem sempre abrangem a exploração de vulnerabilidades em aplicações de banco de dados. Explicita, também, que a análise de mecanismos internos ao SGBD (neste caso, mecanismo de auditoria), adiciona uma camada a mais de segurança à infraestrutura. Esta camada complementa os instrumentos de autenticação de um SGBD, bem como estruturas externas, a saber, de rede ou de sistema operacional de detecção de intrusão. Acrescenta que essas ações maliciosas no SGBD podem não ser detectadas por essas estruturas externas, pois podem não ser consideradas maliciosas sob seu ponto de vista.

Os autores do trabalho (63) introduzem um mecanismo que possibilita a detecção de acesso malicioso aos dados. Isso é alcançado realizando uma análise *online* do rastro de auditoria do SGBD. Para tal, traça um perfil de transações válidas, permitindo que o sistema identifique e sinalize sequências não autorizadas

de comandos SQL. Apresentam um algoritmo utilizado para aprender os perfis de transação executados pelos usuários. Esta aprendizagem é fundamental, pois forma a base sobre a qual as atividades maliciosas são detectadas. Isso possibilita proteger tanto aplicações de banco de dados tradicionais de violações de dados (tratando do aspecto “Integridade” da Segurança da Informação) quanto aplicações baseadas na *web* de ameaças como ataques de SQLIA. Por fim, adiciona-se que a abordagem proposta no trabalho utiliza *logs* de mecanismos de auditoria de um SGBD. Este mecanismo contém informações de usuário, sessão, transação, ação executada e *timestamp*.

Medeiros et al. (13) exploram a detecção de ataques SQLIA em seu trabalho. Eles notaram que o SGBD é um local interessante para se adicionar proteção contra SQLIA, pois possui um conhecimento profundo a respeito de cláusulas, predicados e expressões em uma consulta SQL. O trabalho adiciona que nenhum mecanismo fora do banco de dados possuiria tal conhecimento.

No trabalho (13), os autores também identificam um problema prevalente denominado "incompatibilidade semântica". Esse descompasso surge quando os desenvolvedores têm concepções erradas sobre como as consultas SQL são executadas pelos SGBDs. Tais concepções equivocadas podem introduzir inadvertidamente vulnerabilidades na aplicação, tornando-as suscetíveis a vários ataques de SQLIA.

Para enfrentar esses desafios da pesquisa (13), os autores apresentam uma ferramenta projetada para prevenir ataques a SGBDs. Essa ferramenta é implementada dentro do SGBD, possibilitando-a examinar cada consulta executada em busca de potenciais ataques. O sistema passa por uma fase de treinamento, durante a qual gera modelos de consulta para a aplicação, sendo feito por meio da execução de comandos nos aplicativos *web* que servem de *frontend* para o banco de dados.

3.3 APLICAÇÃO DE MACHINE LEARNING PARA DETECÇÃO DE ATAQUES A SGBD

Li et al. (64) retratam que é necessário avançar na garantia da segurança e integridade de grandes quantidades de dados gerados nos ambientes corporativos. Para abordar o tema, eles escolheram o caminho da detecção de operações anormais de acesso a dados no SGBD, fazendo uso de DA contextual. Apesar das rigorosas políticas de controle de acesso, essas anomalias, muitas vezes resultantes de erros operacionais ou ataques maliciosos, continuam a assolar os sistemas. Métodos tradicionais concentram-se principalmente em padrões de ataque conhecidos ou comportamentos que se desviam significativamente da norma. Isso deixa uma lacuna na detecção de operações furtivas que imitam de perto as regulares.

Os pesquisadores de (64) introduziram um sistema chamado de *Unsupervised Contextual Anomaly Detection in Database Systems* (UCAD). A abordagem do UCAD reside em sua capacidade de detectar operações anormais de acesso a dados, comparando a semântica inerente de uma operação com sua intenção contextual mais ampla. Para alcançar isso, o artigo faz a aplicação de um modelo variante dos *Transformers*, que eles nominaram de Trans-DAS. Este modelo criado, ao contrário de tradicionais como LSTM e *Transformer* convencional, não depende de informações de ordem – ou seja, não é um modelo sequencial ou *seq2seq* –, o que, na conjuntura do trabalho, é adequado para a tarefa pretendida. Além de um módulo de DA, foi proposto outro de pré-processamento para limpeza dos dados.

O trabalho (64) utiliza *logs* de sistema (portanto, *logs* internos) como fonte de seus dados. Mais precisamente, *logs* de sessões dos usuários de uma empresa de Internet. A partir desses dados, o modelo discerne a semântica de operações individuais e tenta compreender a intenção contextual das sequências de operações – a semântica das operações de acesso a dados, por sua vez, pode variar amplamente com base em seu contexto. Além disso, os padrões heterogêneos de acesso do usuário significam que diferentes sequências podem ter semânticas idênticas.

A fase de treinamento da ferramenta (64) envolve a construção de vocabulário para a "*tokenização*" de operações de acesso a dados e a remoção de dados de sessão ruidosos para criar um conjunto de dados limpos. Este conjunto de dados então treina o modelo Trans-DAS para entender a informação semântica normal. Durante a fase de detecção online, as sessões de usuário ativas são "*tokenizadas*", e o modelo treinado avalia a semântica das operações em andamento para detectar anomalias.

Hashem et al. (46) propõem um mecanismo para detectar simultaneamente DoS e SQLIA. Ele usa um conjunto de dados publicamente disponível chamado NSL-KDD que contém registros de ataque DDoS e é baseado em captura de pacotes. O sistema proposto tem duas partes: a primeira é responsável por detectar DDoS nas camadas de rede, transporte e aplicação, e a segunda, por detectar SQLIA.

O trabalho (46) utiliza o NSL-KDD, no qual estão presentes ataques diversos, incluindo ataques DDoS e SQLIA. Este *dataset* público é formado a partir da captura de pacotes de rede (65) e se encaixa com os objetivos dos pesquisadores, de fornecer a detecção de ataques em um servidor de rede. Utiliza-se de algoritmos de classificação – portanto, uma abordagem de AM supervisionada – para detecção de DDoS.

Para a parte relativa à detecção de SQLIA do trabalho (46), lança mão fortemente de técnicas de NLP (*Natural Language Processing*), mais precisamente por meio do uso da "*tokenização*" para treinar um modelo de AM para padrões de consulta SQL. Para tal, os autores utilizam-se de técnicas de dividir frases ou sentenças em unidades menores chamadas *tokens*. O sistema, então, verifica cada consulta executada e as classifica em um padrão suspeito ou não.

Sofi et al. (30) destacam a crescente sofisticação dos ataques DDoS e a consequente dificuldade em detectá-los e combatê-los. Os métodos tradicionais de detecção frequentemente falham diante dos ataques DDoS modernos, que exploram endereços IP falsificados para eludir a identificação da fonte. Tais táticas de evasão tornam esses ataques particularmente prejudiciais, pois podem visar não apenas usuários individuais, mas também grandes instituições, incluindo agências governamentais, corporações financeiras e até estabelecimentos de defesa.

Para enfrentar esse desafio, os autores de (30) propõem a criação de um Sistema de Detecção de Intrusões (*Intrusion Detection System* - IDS). Este IDS faz uso de um *dataset* próprio, gerado especificamente para este trabalho, a partir da captura de pacotes. Utiliza, para classificação dos ataques, alguns algoritmos de AM supervisionado, mais notadamente *Naïve Bayes*, MLP, Máquinas de Vetores de Suporte (*Support Vector Machines* – SVM) e Árvores de Decisão. Dentre os ataques tratados está o SIDDoS. Reconhecem, entretanto, a ausência de fontes de dados públicas para os ataques abordados no artigo, que eles denominaram de "modernos".

3.4 DISCUSSÃO SOBRE OS TRABALHOS

Como apresentado acima, identificamos que a maioria dos trabalhos se baseia na análise e detecção de ataques a SGBDs a partir de capturas de pacotes. Alguns trabalhos fazem suposições sobre pacotes HTTP, funcionando de forma semelhante a um *Firewall* de Aplicação *Web* (*Web Application Firewall* - WAF). Outros, criam linhas de base das consultas executadas como uma espécie de assinatura e, aquilo que fugir desse padrão, é considerado atividade maliciosa. Apenas dois trabalhos lidaram diretamente com a captura de ataques a SGBD a partir de seu interior. Portanto, isso indica a necessidade de se explorar mais o assunto.

Nota-se que o artigo (64), apesar de utilizar uma fonte de dados com características semelhantes à utilizada pelo nosso trabalho, faz uso das operações executadas por cada usuário e cada sessão. Ou seja, as consultas e demais operações necessitam ser conhecidas e, portanto, não são anonimizadas. Isso pode trazer um problema de privacidade dos dados, pois administradores ou mesmo ferramentas maliciosas já instaladas poderão ter acesso a informações valiosas e reveladoras do banco de dados. Este problema não ocorre no nosso trabalho, pois lidamos exclusivamente com as sessões estabelecidas pelos usuários, sem haver a necessidade de conhecer suas consultas.

O trabalho (13) faz uso de informações contidas dentro dos SGBDs, informações estas que apresentam aspectos qualitativos maiores do que se retiradas de captura de pacotes. A partir daí, compara as consultas executadas com um modelo de consultas permitidas. A ferramenta proposta, no entanto, altera a estrutura do banco de dados, agindo como uma espécie de *proxy*, o que faria com que as aplicações tivessem que ser alteradas para comportar essa modificação. Consultas inicialmente não previstas para criação dos modelos de consultas permitidas seriam classificadas como anômalas, mesmo havendo a possibilidade de não o ser. Por fim, por não utilizar técnicas de AM, mais avançadas e maleáveis a ponto de resolver o problema relatado, não parece um caminho interessante a se seguir.

A ferramenta proposta por (30) não enfatiza a detecção de ataques a banco de dados, apenas lida com a detecção de SIDDoS. Cria um *dataset* que inclui tal ataque, baseando-se fortemente na assunção do conhecimento de como se dá um SIDDoS. Para se chegar a esta conclusão, analisam os pacotes trafegados e não o que é executado no SGBD. Este fato dá margem a serem criadas outras formas de SIDDoS, com novas sintaxes, por exemplo, que não seriam classificados pela ferramenta como um ataque.

Quanto ao trabalho (63), frisa-se que se utiliza de *logs* internos do SGBD, assim como a nossa abordagem. Dentre esses *logs*, as sessões abertas pelo usuário são levadas em consideração, o que valida nossa abordagem. Porém, um ponto que não está claro é se faz uso de AM ou suas variantes para esta detecção ou não. Ele cita fase de aprendizado (treinamento) e fase de detecção, o que é condizente com abordagens de AM. Porém, acreditamos que não use AM, pois o trabalho apenas cita a definição de um padrão de comportamento correto e, a partir dele, identifica ataques ao banco. Esta descrição se encaixa, também, com a definição de assinaturas. Carece, portanto, da adaptabilidade trazida pelo nosso trabalho e por outros aqui apresentados.

O trabalho (46) emprega técnicas de aprendizado supervisionado a um *dataset* amplamente conhecido e pesquisado, o NSL-KDD. Dentre seus registros, há aqueles referentes a ataques SIDDoS. Propõe-se a detectar, simultaneamente, ataques DDoS e SQLIA por meio da utilização de uma abordagem de AM

Tabela 3.1: Trabalhos relacionados.

Referência	Detecção de DDoS em SGBD	Análise de *logs* do SGBD	Mantém Arquitetura do SGBD	AM Supervisionado	AM Não Supervisionado
Fonseca et al. (63)	✓	✓	✓	✗	✗
Li et al. (64)	✓	✓	✓	✗	✓
Hashem et al. (46)	✓	✗	✓	✓	✗
Sofi et al. (30)	✓	✗	✓	✓	✗
Medeiros et al. (13)	✗	✓	✗	✗	✗
Nosso Trabalho	✓	✓	✓	✓	✓

supervisionada. Carece, entretanto, de uma abordagem que possibilite a detecção dessas atividades maliciosas não somente usando uma fonte de dados rotulada, mas detectar aquelas atividades que fogem à norma ou ao padrão observado na infraestrutura.

A Tabela 3.1 resume os achados dos trabalhos pesquisados e aqui citados. Comparativamente ao nosso trabalho, a maioria das pesquisas analisadas foca na análise de capturas de pacotes e em ataques mais comuns aos SGBDs, como SQLIA – nosso trabalho tem por foco a identificação de SIDDoS, forma nociva de ataque que indisponibiliza a infraestrutura. Os trabalhos que fazem uso de AM, majoritariamente, o fazem utilizando abordagens supervisionadas. Estas abordagens possibilitam uma precisão maior na detecção de ataques, porém necessitam de ser feita uma rotulação (portanto, classificação) manual prévia em um *dataset* de treino.

Nosso trabalho atua em ambas as frentes em se tratando de AM: com uma abordagem supervisionada, por meio de algoritmos de classificação, e uma abordagem não supervisionada por meio de DA. Outro fator que contribui com o nosso trabalho e sua relevância diz respeito ao uso de uma fonte rica de informações das atividades de um SGBD: seus *logs* internos. Neles, estão contidas informações qualitativas do funcionamento de um SGBD, que se configuram como uma fonte de qualidade para a análise de atividades maliciosas a tal elemento da infraestrutura.

Por fim, nosso trabalho não impõe mudança de arquitetura para aplicações, tornando-a bem menos intrusiva. Este fator facilita sua implantação em ambiente de produção, ao mesmo tempo em que adiciona uma camada de segurança a mais à infraestrutura como um todo.

3.5 CONSIDERAÇÕES FINAIS

Este capítulo apresentou trabalhos utilizados como balizadores desta dissertação. Trabalhos que apresentam, individualmente, nuances diferentes do que foi desenvolvido nesta dissertação. Ficam evidentes as várias formas de se abordar o tema de segurança para SGBDs, mas fica claro, também, que a abordagem por nós escolhida tem pontos positivos que diferenciam nosso trabalho.

No próximo capítulo, será apresentada a nossa solução em seus detalhes, desde a escolha do *dataset* até o uso das abordagens de AM, passando, inclusive, pela escolha dos algoritmos.

4 SOLUÇÃO DESENVOLVIDA

Este capítulo apresenta a solução desenvolvida para detecção de ataques de DoS a partir de *logs* coletados em SGBDs. Explica o porquê da escolha da fonte utilizada para obtenção do *dataset* analisado. Explicita, também, o fato pouco usual de os dados terem sido coletados durante o acontecimento de um ataque real de DDoS a um SGBD. Apresenta, também, as arquiteturas e meandros das abordagens escolhidas para este trabalho, tanto supervisionada quanto não supervisionada.

4.1 HISTÓRICO DO PROBLEMA

Em um dia corriqueiro, de atividades normais, notou-se um uso excessivo de parte da infraestrutura de provimento de serviços na *web* de um órgão da cúpula do Poder Judiciário Federal. Componentes como *links* de Internet, *firewall*, *proxy*/ADC, servidor de aplicação/*web* apresentavam carga padrão, dentro do esperado. Porém, face ao aumento de reclamações a respeito de lentidões e indisponibilidades dos sistemas, constatou-se que o servidor de bancos de dados (SGBD) de produção apresentava carga excessiva. O *load average*, métrica usual no mundo *Unix*/*Linux*, de seu sistema operacional estava por volta de 200 – um valor alto, mas aceitável, é de 20.

Foram conduzidas investigações em todos os componentes da infraestrutura no intuito de se chegar ao cerne do problema, qual seria a causa. Uma equipe de resposta a incidentes composta por desenvolvedores, administradores de servidores de aplicação, administradores de banco de dados, administradores de rede e especialistas em segurança cibernética realizou a identificação do ataque. Esta equipe fez uso de ferramentas como *software* de monitoramento de desempenho do SGBD, tabela/visão de sessões do SGBD e *Network Monitoring System* (NMS) ou Sistema de Monitoramento de Rede, dentre outros. No entanto, a atividade maliciosa não foi detectada nas camadas inferiores dos protocolos TCP/IP ou em dispositivos de segurança cibernética, como WAF, IDS, *Intrusion Prevention System* (IPS) ou Sistema de Prevenção de Intrusão. As operadoras de telefonia responsáveis pelos *links* de Internet, que possuem proteção contra DDoS, informaram não haver problemas dessa natureza. A suspeita voltou-se, então, para um DDoS ao SGBD por meio de SQLIA.

Como forma de cessar os ataques, foram adotadas medidas que impedissem ferramentas automatizadas (*botnets*) de realizar as consultas que estavam causando a indisponibilidade. Os sistemas *web* tiveram sua disponibilização suspensa por prazo indeterminado até que medidas mais concretas de defesa fossem adotadas. Concomitantemente a este momento de ataque, uma equipe conduzia uma avaliação de *logs* internos do SGBD, de forma a se tentar extrair informações úteis.

O ataque ocorreu entre os dias 04 e 08 de abril de 2022 e, neste ínterim, *logs* foram coletados, transformando-se em *datasets* de treinamento e teste. Após o término do ataque, os *logs* foram analisados e foi adicionada uma coluna nos *datasets*, esta representando rótulos, com os valores “1” para ataque e “0” para atividade normal – esta característica qualifica nosso problema como um problema de classificação

binária. Desta forma, o cenário para uso na pesquisa estava preparado.

4.2 ARQUITETURA GERAL DA SOLUÇÃO

Optamos por analisar os *logs* internos do SGBD devido a alguns fatores. Apesar de ser possível detectar ataques a partir de outros componentes na infraestrutura, os *logs* do SGBD podem ser mais ricos em termos de informações em relação às que provavelmente seriam obtidas em análise de pacotes. Eles contêm variáveis que registram o desempenho das operações executadas, bem como o consumo de recursos do servidor. Esses parâmetros armazenam informações conhecidas apenas dentro do SGBD e representam uma espécie de impressão digital de cada consulta, uma vez que são o produto de suas execuções (66). Por esse motivo, esses dados não estão presentes em pacotes como solicitações ou respostas HTTP/HTTPS nem em solicitações de banco de dados iniciadas no servidor de aplicação. Assim, entendemos que nosso trabalho traz uma abordagem complementar à disponível no mercado para detectar ataques DDoS em SGBDs, adicionando uma camada de segurança à infraestrutura.

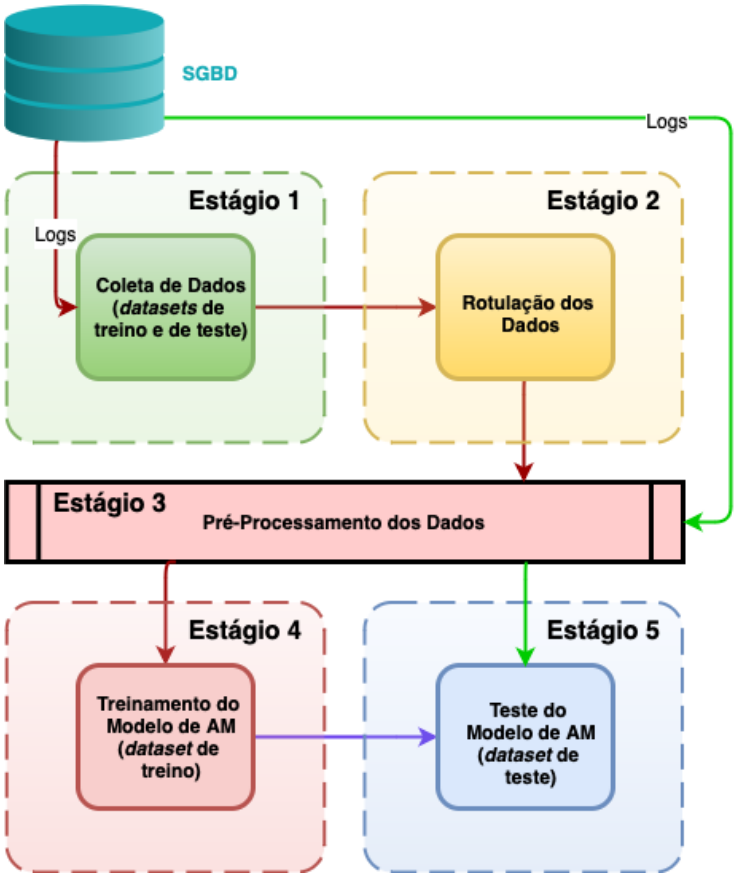


Figura 4.1: Visão geral dos estágios desenvolvidos para detecção de DoS/DDoS em um SGBD. Adaptado de (2)

Destarte, este trabalho utiliza métodos amplamente disseminados em AM aplicados à segurança cibernética. No entanto, eles foram instanciados e adaptados para o contexto em foco, compondo uma abordagem de cinco estágios (Figura 4.1). O primeiro está relacionado à coleta de dados do SGBD, que consiste em *logs* de sessões criadas no banco de dados sempre que ele executa uma consulta. Esses *logs*

compreendem características que identificam cada sessão, incluindo o usuário que a executa, tempo gasto pelo processador, identificação do usuário, endereço IP, tempo de espera, entre outros. Essas informações representam os parâmetros de desempenho das consultas que o SGBD executa, identificando as consultas inequivocamente.

O segundo estágio consiste em rotular os registros do conjunto de dados com base em um ataque DDoS real a um SGBD em produção, no qual os agentes de ataque foram identificados usando diversas ferramentas. Este estágio somente seria necessário para a utilização da abordagem supervisionada, mas foi fundamental para avaliação do desempenho na abordagem não supervisionada.

O terceiro estágio consiste na extração e engenharia de características (*feature engineering*), onde excluímos variáveis desnecessárias e extraímos novas características das informações presentes no conjunto de dados. A codificação de variáveis categóricas também é realizada durante esta fase.

O quarto estágio consiste em treinar o modelo usando uma técnica de validação cruzada para detectar ataques. Vários algoritmos de AM podem ser avaliados nesta fase. Optamos por três algoritmos de classificação (portanto, AM supervisionada) – Regressão Logística, *Random Forest* e XGBoost – e três algoritmos para DA (AM não supervisionada) – Floresta de Isolamento, *Autoencoders* e LSTM (estes dois últimos, algoritmos de DL). Esses algoritmos foram escolhidos, pois têm mostrado resultados promissores na literatura e por apresentarem custos computacionais aceitáveis.

O quinto estágio é a aplicação do modelo treinado a novos dados. Esta estratégia foi preferida, neste estudo em detrimento da divisão treinamento-teste, pois o conjunto de dados de treinamento era relativamente pequeno. Considerando que nosso trabalho utilizou validação cruzada com excelentes resultados na fase de treinamento, essa abordagem fazendo uso de novos dados seria mais realista do que dividir um conjunto de dados já pequeno para duas finalidades distintas. Os cinco estágios serão discutidos em mais detalhes neste trabalho.

O SGBD analisado foi o *Oracle* 11g instalado em *cluster* e hospedado em um equipamento proprietário do fabricante, chamado *Exadata*. Este equipamento conta com poder de processamento muito acima do que a infraestrutura de aplicações necessitaria. Mesmo assim, o DDoS consignado sobre ele tornou seus dados indisponíveis para as aplicações, demonstrando a capacidade prejudicial deste tipo de ataque.

A tabela de *logs* do SGBD (mais especificamente uma *view*) que forneceu os dados para nosso trabalho chama-se V$SESSION (66). Nela, encontram-se parâmetros (ou variáveis) de desempenho para cada sessão aberta por usuário em um banco de dados. A tabela possui campos que identificam unicamente a sessão, seu estado, data e hora de sua criação, tempo de espera em fila para processamento, tempo gasto com processamento, endereço IP que originou a consulta, usuário de banco/aplicação que efetuou a consulta. No total, temos 99 colunas, cada coluna representando uma variável ou dimensão do *dataset*.

Algumas colunas da tabela, como a que representa o estado da sessão, possuem formato texto. Dentre estas, um pequeno conjunto somente pode assumir valores que estejam contidos em uma lista específica. Variáveis com este atributo são conhecidas como variáveis categóricas. As colunas com esta característica foram “TYPE”, “STATUS”, “SERVER”, “PDDL_STATUS”, “PQ_STATUS”, “BLOCKING_SESSION_STATUS”, “FINAL_BLOCKING_SESSION_STATUS”, “WAIT_CLASS#”, “STATE”, “SERVICE_NAME” e “SESSION_EDITION_ID”.

O uso da tradicional análise de pacotes de rede não conseguiu identificar o ataque, o que nos fez mudar o foco da análise. Esta migrou para dentro do SGBD, mais especificamente nas informações das sessões dos usuários dos bancos de dados. Descobriu-se um número excessivo de sessões criadas por diversos usuários, partindo de vários endereços IP distintos num curto espaço de tempo. Estas características são condizentes com um DDoS – mais precisamente, um SIDDoS por ter sido oriundo de consultas injetadas por meio das aplicações *web*. A partir desta análise, os dados foram devidamente rotulados.

Para um algoritmo de AM funcionar, é necessário que ele lide somente com valores numéricos. Desta forma e com base em análise do significado das variáveis, quatro ações foram tomadas inicialmente: as variáveis nativamente numéricas foram mantidas; variáveis textuais sem significado de desempenho foram excluídas; as variáveis de data e hora e as variáveis textuais não descartadas foram transformadas em numéricas; variáveis textuais categóricas foram codificadas utilizando o método *One-Hot Encoding* (OHE). Este método cria uma matriz esparsa a partir dos possíveis valores categóricos da variável. Ou seja, para cada categoria da variável, uma nova coluna é criada e atribuído o valor "1" caso ela esteja presente naquela observação ou "0" em caso contrário (61, 67). Esta técnica de codificação encaixa-se bem com o nosso *dataset*, pois suas categorias não apresentam ordem e nem quantidade exacerbada de categorias.

A codificação OHE pode trazer um problema ao final do processo: crescimento demasiado do número de variáveis ou dimensões do *dataset*. Este crescimento pode acabar causando a chamada "Maldição da Dimensionalidade" ou *curse of dimensionality*, em seu termo em inglês, mais conhecido. Este problema ocorre quando as dimensões de uma fonte de dados crescem demasiadamente, adicionando complexidade aos modelos de AM. Esta adição contribui para um aumento no consumo de memória dos modelos de AM e na dispersão dos dados, fato que torna o desafio para análise do *dataset* maior. Nestes casos, poder-se-ia resolvê-los de duas formas: aumentando o tamanho da fonte de dados (maior quantidade de registros) até atingir uma densidade suficiente para instâncias de treinamento; ou realizar uma redução de dimensionalidade (52) – esta redução acabou sendo utilizada para outra finalidade do nosso trabalho, por meio da aplicação do PCA. Como os *datasets* foram coletados a partir de uma situação real de ataque, que posteriormente cessou, não seria possível tomar o primeiro caminho. Porém, pelo fato de, ao final do processo, os *datasets* terem sido reduzidos a 52 variáveis, optamos por não tomar a segunda medida, o que se provou correto.

Algumas das variáveis textuais do *dataset* continham informações relevantes para a classificação de ataques e detecção de anomalias e foram transformadas em numéricas mediante extração de características. As principais foram:

- CLIENT_INFO, da qual foram extraídas três *features*: IP da sessão, usuário que abriu a sessão e o horário inicial da sessão. A partir do IP, depreendeu-se se este era interno ou externo (IP privado ou válido). A partir do usuário extraído, determinou-se se há um usuário ativo na consulta ou não;
- LOGON_TIME e STATUS, a partir das quais se depreenderam as novas sessões iniciadas e o tempo decorrido entre cada sessão do usuário. A coluna LOGON_TIME também gerou quatorze colunas contendo dados relativos ao tempo, como dia, ano, período do dia etc. Estas somente foram usadas para geração das visualizações.

Porém, houve pequenas diferenças no pré-processamento dos *datasets* para aplicação de algoritmos de

AM supervisionados e de não supervisionados; estas diferenças estão descritas abaixo.

Em relação à análise de dados ausentes, encontramos algumas variáveis com uma taxa de ausência superior a 90% - estas são as que provavelmente reduziriam o desempenho do modelo de AM (67). Nestes casos, descartamos as variáveis para evitar essa perda nas métricas de treinamento. Também analisamos instâncias (ou linhas da tabela) com poucos dados – menos de 10% da linha preenchida – e as descartamos como um todo (67). Esse limite foi escolhido cuidadosamente de forma a descartar apenas uma pequena porção dos dados, que, neste caso, representou menos de 0,5% da contagem de linhas. Ao final do processo de seleção de características, nosso conjunto de dados não exigiu entrada de dados em variáveis faltantes.

Dados desequilibrados podem afetar algoritmos de AM supervisionado, pois os modelos sabem pouco sobre uma das classes. Modelos treinados usando *datasets* desequilibrados tendem a fazer uma boa previsão da classe mais comum, mas uma previsão imprecisa da outra. Embora, em alguns casos, isso possa resultar em boas métricas preditivas, geralmente não é desejável em problemas reais, pois é quase inútil para prever as classes com menos instâncias (23).

Tratamos esse problema aplicando técnicas de balanceamento, mais notavelmente o *oversampling* da classe menor, que consiste em gerar sinteticamente novas linhas dela. A outra abordagem que poderíamos ter usado seria o *undersampling*, ou seja, reduzir a quantidade de dados da classe predominante para igualar à menor. Esta escolha reduziria o número de linhas para uma quantidade pequena, o que é inadequado para treinar um modelo de AM supervisionado. Para o nosso conjunto de dados de treinamento, utilizamos o *Synthetic Minority Oversampling Technique* (SMOTE) para o *oversampling* (23).

Para determinar qual modelo era mais eficaz usando nossos dados, realizamos uma validação cruzada (*cross-validation*) de 10 vezes das três técnicas de AM supervisionado. O conjunto de dados é dividido em dez partes, nove sendo usadas como conjunto de dados de treinamento e a outra como um conjunto de dados de validação. O propósito desta técnica é evitar o *overfitting* e estimar o quão bem o modelo se sairá com novos dados. *Overfitting* é uma situação indesejada no AM, pois significa que o modelo aprendeu demais sobre os dados de treinamento e é incapaz de generalizar para novos dados. A Figura 4.2 apresenta o processo de validação cruzada.

Figura 4.2: Processo de Validação Cruzada. Adaptado de (2)

Uma coisa que consideramos antes da utilização da validação cruzada é como esse processo deveria dividir nosso *dataset*. Primeiro, tivemos que assumir que é um problema de classificação binária. Neste cenário, a divisão aleatória de dados poderia resultar em partições com diferentes distribuições estatísticas

para a classe alvo. Aplicamos, então, uma divisão estratificada no processo de validação cruzada para superar esse problema. Isso garante que cada divisão tenha a mesma distribuição estatística em relação à variável alvo. Assim, obtivemos uma avaliação mais precisa dos modelos (68).

4.3 ABORDAGENS DE AM SUPERVISIONADO E NÃO SUPERVISIONADO

Conforme relatado na Seção 4.2, para abordagem de AM supervisionado, optamos pelo uso dos algoritmos Regressão Logística, Floresta Aleatória e *XGBoost*. Para a aplicação de DA aos dados, selecionamos, inicialmente, três algoritmos mais frequentemente utilizados e com melhores desempenhos na bibliografia pesquisada: *iForest*, *Autoencoder* e LSTM. Porém, antes da aplicação dos algoritmos, foram necessárias mais transformações ao *dataset*, de forma a se capturar as anomalias criadas pelo ataque. Assim, foram criadas 15 novas variáveis a partir das pré-existentes, o que significa dizer que foram extraídas mais informações dos dados presentes. Estas novas variáveis mostraram-se muito úteis, possibilitando-nos a identificar os ataques nos dias e horários exatos em que ocorreram, conforme se observa nas Figuras 4.3 e 4.4.

Este fato é corroborado pela Figura 4.3, em que se nota um crescimento repentino da média de sessões por usuário. Porém, somente seria possível ter certeza desta afirmação após avaliação do algoritmo não supervisionado, pois, afinal, poderia ser apenas um crescimento sazonal da utilização.

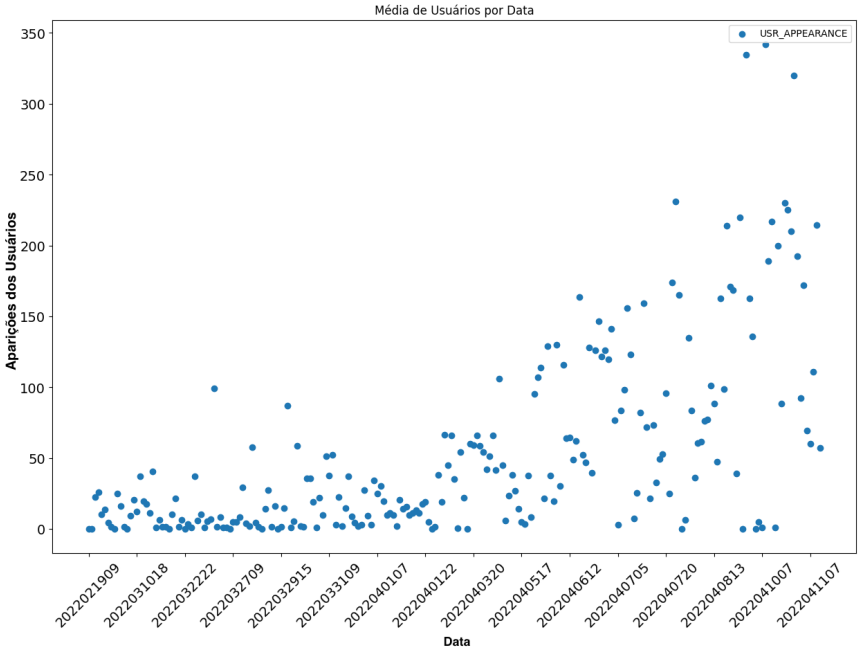


Figura 4.3: Média de sessões de usuários.

Na Figura 4.4 é possível verificar os pontos que foram considerados anomalias pelo modelo *iForest* treinado, corroborando a percepção inicial de que algo diferente do esperado havia acontecido.

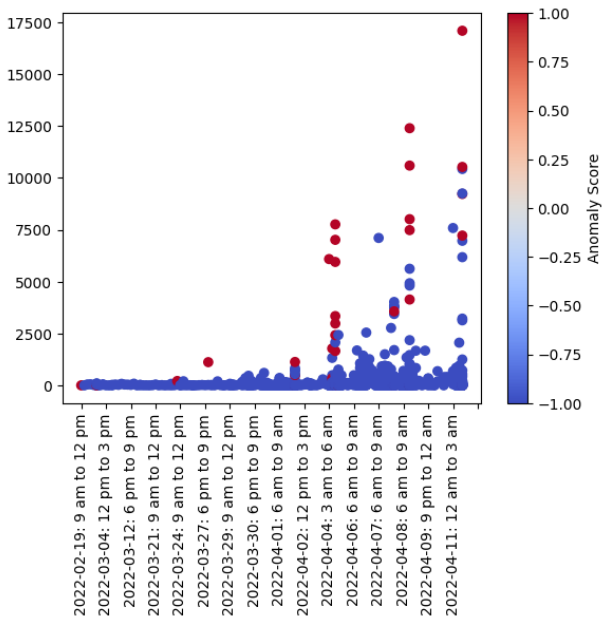
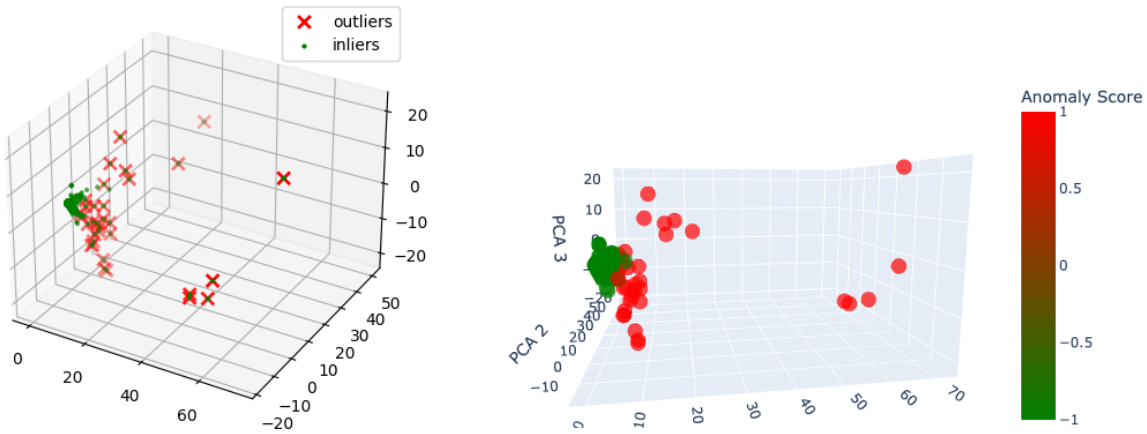


Figura 4.4: Anomalias detectadas na quantidade de sessões de usuários. Pontos em vermelho denotam anomalias identificadas.

Como forma de facilitar a visualização das anomalias do nosso *dataset*, realizamos uma redução de dimensionalidade utilizando o algoritmo PCA e colocamos os pontos em gráficos em 3 dimensões. Assim, as Figuras 4.5a e 4.5b foram geradas e condensadas na Figura 4.5. Nota-se uma concentração dos pontos normais no plano do componente PCA 1. À medida que os pontos avançam em direção aos demais planos, eles apresentam-se mais dispersos e são classificados, em sua maioria, como anomalias.



(a) *Visualização 3D dos pontos anômalos.*

(b) Visualização 3D dos pontos anômalos sob outra perspectiva.

Figura 4.5: Visualização de pontos anômalos após redução de dimensionalidade (PCA).

Antes de aplicarmos os algoritmos de AM aos dados, utilizamos uma técnica chamada de *feature scaling*, que consiste em trazer as variáveis do *dataset* para a mesma escala, pois esta é uma forma com que os algoritmos de AM trabalham melhor. A escala *standard scaling* foi a escolhida – ela escala todos os valores das variáveis para valores entre -1 e 1, o que representa uma distribuição normal com média 0 e desvio padrão 1 (21). Porém, é muito importante observar que esta técnica deve ser aplicada a cada *dataset*

isoladamente, de forma a evitar vazamento de dados ou *data leakage*. O vazamento de dados ocorre quando o modelo treinado tem conhecimento dos dados presentes no *dataset* de teste (61). Caso seja aplicada a escala com um *dataset* inteiro e, posteriormente, dividi-lo em treinamento-validação-teste, a avaliação fica comprometida, pois os *datasets* de validação e treinamento carregam consigo informações do *dataset* de treino.

Executamos, então, a validação cruzada a cada um dos algoritmos escolhidos. Usamos, então, as métricas Acurácia, Precisão, Sensibilidade e F1-*score* para definir qual o melhor algoritmo.

Hiperparâmetros são parâmetros de configuração dos algoritmos de AM (21). São responsáveis por definir caraterísticas do algoritmo que podem trazer melhora ou piora ao resultado - necessitam, portanto, de cuidado ao realizar alterações em seus valores. Para o iForest, ajustamos apenas dois hiperparâmetros: a quantidade de estimadores (quantidade de árvores de decisão que ele iria criar para treinar o modelo) e o limiar de DA.

Para o algoritmo *Autoencoder*, foram criados um *encoder* e um *decoder*, ambos com duas camadas ocultas. A dimensão dos dados de entrada equivale ao número de variáveis do *dataset* e as camadas ocultas criadas com dimensões na metade do tamanho da camada anterior. Como não é possível para uma RN treinar todos os dados de uma vez, o hiperparâmetro "tamanho de lote" (*batch size*) deve ser definido. Ele determina qual a quantidade de dados utilizados para treinar o modelo por vez. Para que o modelo seja treinado em todos os dados, ele é treinado de lote em lote. Ao final deste processo, os pesos e vieses são atualizados pelo mecanismo de retropropagação. Este processo completo é chamado de época. Configuramos nosso *Autoencoder* para 30 épocas de treinamento – quanto maior esse número, maior a probabilidade de o modelo diminuir os erros residuais de reconstrução, mas muito maior é o consumo de memória e o tempo de processamento (3, 21, 56). O pseudocódigo para o algoritmo *Autoencoder* utilizado no nosso trabalho está presente no Algoritmo 1.

Algorithm 1 Pseudocódigo do *Autoencoder*

1: $dimensoes\_entrada \leftarrow$ número de colunas do $dataset$
2: $dimensoes\_codificacao \leftarrow$ arredonda$(dimensoes_entrada/2)$
3: $dimensoes\_camada\_oculta \leftarrow$ arredonda$(dimensoes_codificacao/2)$
4: Inicialize $camada\_entrada$ com $Entrada$ com formato $(dimensoes\_entrada,)$
5: $encoder \leftarrow$ camada Dense com $dimensoes\_codificacao$ unidades, ativação $sigmoid$ e regularizador $l1$
6: $encoder \leftarrow$ camada Dense com $dimensoes\_camada\_oculta$ unidades e ativação $sigmoid$
7: $decoder \leftarrow$ camada Dense com $dimensoes\_camada\_oculta$ unidades e ativação $relu$
8: $decoder \leftarrow$ camada Dense com $dimensoes\_codificacao$ unidades e ativação $sigmoid$
9: $decoder \leftarrow$ camada Dense com $dimensoes\_entrada$ unidades e ativação $sigmoid$
10: $autoencoder \leftarrow$ Modelo com entradas $camada\_entrada$ e saídas $decoder$

Conforme mencionado, nosso *Autoencoder*, em seu *encoder*, utiliza-se de duas camadas ocultas no *encoder* para comprimir os dados. Inicia-se com uma camada de entrada com o número de dimensões igual à quantidade de *features* do *dataset*, comprime para uma camada com metade das dimensões da camada de entrada. A camada seguinte também possui metade das dimensões da camada anterior e representa o estado de compressão máxima deste modelo. A partir de então, entra-se no *decoder*, em que os dados são reconstruídos. Em sua primeira camada, tem-se a quantidade de dimensões da última camada do *encoder*. Na camada seguinte, o número de dimensões dobra, caracterizando-se uma descompressão ou

reconstrução. O mesmo ocorre na última camada. Ao final do processo, temos os dados reconstruídos e seu resultado é comparado com os dados de entrada. O erro dessa, então, é utilizado como base para determinação das anomalias quando aplicado o modelo aos dados de teste.

Os mesmos hiperparâmetros do nosso *Autoencoder* foram utilizados para a LSTM. Porém, a configuração da LSTM é um pouco diferente, visto que não possui explicitamente a arquitetura *encoder/decoder*. Nela, foram configuradas três camadas ocultas intercaladas com camadas de *dropout*, configurada com uma taxa padrão de 20%. Esta camada de *dropout* pertence a uma técnica chamada regularização e tem o objetivo de prevenir o *overfitting* do modelo (21, 56). Ao final, uma camada de saída reconstrói os dados da sequência e tem-se, então, a possibilidade de serem comparados os valores de saída com a entrada e calcular seu erro – essa comparação se dá pela métrica *mean squared error* – MSE. Desta forma, nossa LSTM funciona de modo análogo a um *Autoencoder*, no sentido de comparar os valores reconstruídos com os inicialmente entrados. O pseudocódigo 2 representa a LSTM criada para o nosso projeto.

Algorithm 2 Pseudocódigo da LSTM

1: Inicializa $scaler$ como MinMaxScaler()
2: $X\_train \leftarrow$ fit e transform X_train usando $scaler$
3: $X\_test \leftarrow$ fit e transform X_test usando $scaler$
4: $X\_train\_reshaped \leftarrow$ Transforma_dataset_em_sequencia_3D $(X\_train, tamanho\_sequencia)$
5: $X\_test\_reshaped \leftarrow$ Transforma_dataset_em_sequencia_3D $(X\_test, tamanho\_sequencia)$
6: Inicialize $lstm$ como um modelo Sequential()
7: Adicione camada LSTM à $lstm$ com 50 unidades, ativação $sigmoid$, $return\_sequences$ True e formato de entrada $(tamanho\_sequencia, quantidade\_dimensoes\_dataset)$
8: Adicione camada de Dropout à $lstm$ com taxa 0.2
9: Adicione camada LSTM à $lstm$ com 50 unidades, ativação $sigmoid$, $return\_sequences$ True
10: Adicione camada de Dropout à $lstm$ com taxa 0.2
11: Adicione camada LSTM à $lstm$ com 50 unidades, ativação $relu$
12: Adicione camada de Dropout à $lstm$ com taxa 0.2
13: Adicione camada Dense à $lstm$ com $quantidade\_dimensoes\_dataset$ unidades

Nossa LSTM inicia-se por normalizar os dados, de forma a mantê-los dentro de um intervalo entre 0 e 1 (por isso o uso do *MinMaxScaler()*, que devolve valores sempre dentro deste intervalo). Este processo é importante, pois traz todas as variáveis do *dataset* para a mesma escala. Isso faz com que o modelo aprenda melhor e mais rápido, além de auxiliar na prevenção do problema do gradiente evanescente (56). Posteriormente, é realizado um processo de transformação do *dataset* em uma sequência tridimensional. Esta transformação é necessária, pois a LSTM é uma RNR projetada para lidar com sequência de dados (*seq2seq*) e capturar dependências temporais desses dados. Outra informação importante a ser notada no nosso pseudocódigo diz respeito às camadas ocultas: trabalhamos com 50 neurônios por camada. Esses neurônios representam componentes que realizam cálculos e mantêm um estado interno ao longo do tempo, contribuindo para a capacidade da rede de capturar dependências temporais.

Conforme supramencionado, optamos por nossa LSTM ter três camadas ocultas com 50 neurônios cada e uma de saída com o número de neurônios igual ao número de *features* do *dataset* (aquilo que se quer prever). Poderíamos ter optado por mais camadas ocultas e/ou mais neurônios, o que poderia potencialmente melhorar os resultados do nosso modelo. Esta melhora se dá pois modelos mais profundos conseguem capturar relações mais complexas entre os dados, permitindo representações mais ricas dos dados. Porém, este não é um fato garantido e encontra alguns problemas, como possível *overfitting* do modelo e custo computacional bem mais alto. Optamos, então, por manter as escolhas descritas no pseudocódigo.

Por fim, apontamos o uso das funções de ativação nos algoritmos de AP. Elas têm a função de introduzir não-linearidade aos modelos. Isso permite ao modelo capturar relações complexas e não-lineares entre os dados de entrada e saída (56). Isso significa dizer que permite que o modelo lide com padrões ou comportamentos nos dados que não são simplesmente lineares ou diretamente proporcionais. As funções utilizadas foram ReLU (*Rectified Linear Unit*), cujo uso traz aceleração da convergência do treinamento e na mitigação do problema do gradiente evanescente; e a *Sigmoid*, que, por retornar valores entre 0 e 1, é útil para classificação binária, o que é o nosso caso. De forma a facilitar o entendimento dos pseudocódigos apresentados, nós disponibilizamos a implementação de ambos nos Apêndices I e II. Disponibilizamos, também, o código-fonte completo e os *datasets* em repositório do GitHub [1].

4.4 CONSIDERAÇÕES FINAIS

Neste capítulo, apresentamos um histórico do ataque que foi o responsável por fornecer nossos *datasets*. Passamos pelo processo de identificação do SIDDoS sofrido e pela identificação do melhor local para realizar a detecção do ataque. Discutimos as características dos *datasets* e as transformações que foram necessárias serem aplicadas a eles para obtermos os melhores resultados possíveis. Por fim, apresentamos ambas as abordagens utilizadas na detecção do SIDDoS, supervisionada e não supervisionada e mais detalhes sobre os algoritmos utilizados.

No próximo capítulo, apresentaremos os resultados obtidos no nosso trabalho, suas análises e discussões a respeito.

[1] Código-fonte da dissertação e *datasets* anonimizados: https://github.com/daniloamchagas/codigodissertacao

5 ANÁLISE DOS DADOS E RESULTADOS

Neste capítulo, analisamos os resultados obtidos pela nossa solução, tanto em sua vertente de AM supervisionado como para DA. Ao final, discutimos os resultados obtidos e o que eles representam para a segurança cibernética de uma infraestrutura de bancos de dados.

5.1 CONFIGURAÇÃO DOS EXPERIMENTOS

A maneira mais comum de testar a eficácia de um modelo é dividir o conjunto de dados em dois ou três subconjuntos, incluindo treinamento-teste ou treinamento-validação-teste (61). No entanto, como mencionado anteriormente, coletamos o *dataset* de teste não no momento inicial, mas quando o ataque ainda estava acontecendo, em momento posterior. Optamos por testar os modelos em novos dados em vez de dividir o *dataset* de treinamento por algumas razões, conforme descrito abaixo.

Primeiro, tivemos que considerar que nosso *dataset* inicial precisava ser maior para ser possível dividir sem impactar negativamente o treinamento. Outro aspecto considerado é que, ao coletar um novo *dataset* de teste, evitamos qualquer possibilidade de vazamento de dados. Esta situação, conforme descrito na seção 4.3, é indesejável, pois o modelo treinado conhece os dados do *dataset* de teste, trazendo resultados irreais. Neste caso, ao aplicar o modelo a dados de produção, reais, as métricas obtidas provavelmente não terão um desempenho comparável ao desempenho do *dataset* de teste.

Em segunda instância, também queríamos determinar quão prático seria nosso trabalho em uma situação real. No entanto, para usar essa abordagem de um *dataset* de testes não oriundo da divisão do *dataset* original, é imperativo determinar se os novos dados têm a mesma distribuição estatística que o *dataset* de treinamento. Para este fim, usamos a Validação Adversarial (62).

A Validação Adversarial é uma técnica aplicada para investigar se as características do *dataset* de teste são as mesmas do *dataset* de treinamento. Identificar quaisquer mudanças na distribuição de dados que possam fazer o modelo treinado ter um desempenho inferior com novos dados é crucial. Para aplicá-la, é necessário excluir, do *dataset*, a sua variável alvo e criar uma nova variável alvo representando se os dados vêm de *dataset* de treinamento ou de teste. Em seguida, um modelo de classificação faz previsões e a métrica ROC-AUC é usada para avaliação dos resultados desta técnica. Um valor resultante próximo de 50% indica que ambos os dados são indistinguíveis entre si e, portanto, têm a mesma distribuição (62). Em nosso caso, alcançamos um ROC-AUC de 51,31% em um modelo *XGBoost*, evidenciando que podemos usar nossos novos dados como um *dataset* de teste.

Relativamente à configuração dos nossos experimentos, utilizamos os componentes e suas respectivas versões descritos na Tabela 5.1. Todos os experimentos foram executados em um *notebook* MacBook Air com processador Apple M2, 16 GB de RAM, com uma CPU de 8 núcleos, GPU com 16 núcleos, e outros 16 núcleos exclusivos para Inteligência Artificial, chamados comercialmente de *Neural Engine*.

Tabela 5.1: Lista de componentes do experimento.

Componente	**Versão**
Python	3.10.6
Matplotlib	3.6.3
Plotly	5.12.0
Pandas	1.5.2
NumPy	1.22.3
Jupyter Core	5.1.3
Jupyter Client	7.4.9
Scikit-Learn	1.2.0
Keras	2.13.1
TensorFlow	2.13.0
XGBoost	1.7.5

Tabela 5.2: Métricas obtidas para o conjunto de dados de treinamento-validação

Avaliação de Desempenho - *Dataset* de Treinamento - % - Média em Validação Cruzada			
Métrica	Regressão Logística	Floresta Aleatória	*XGBoost*
Acurácia	95,87	**99,97**	99,89
Precisão	91,84	**99,92**	99,75
Sensibilidade	**100**	**100**	**100**
F1-*score*	95,6	**99,96**	99,88
ROC-AUC	97,99	**100**	**100**

5.2 RESULTADOS OBTIDOS

De forma a termos uma percepção exata de como se saem as abordagens escolhidas (AM supervisionado e DA), optamos por estabelecer as mesmas métricas para avaliação dos resultados para ambas. Escolhemos as mais utilizadas na literatura, que são a Acurácia, Precisão, Sensibilidade, F1-*score* e curva ROC-AUC. Todas elas estão baseadas nos valores das classes VP, FP, VN e FN definidos na Matriz de Confusão presente na Tabela 2.1 e permitem que analisemos quão bem os modelos identificam essas classes. É possível perceber pelas métricas selecionadas, inclusive, que algumas escolhas para otimização de FN e VP podem ser feitas por meio da observação dessas métricas. Caso se deseje um modelo que diminua a quantidade de FN, por exemplo, as métricas obtidas pelos modelos treinados indicarão este fato a quem as analisar. Assim, elas se enquadram perfeitamente nos objetivos deste trabalho.

5.2.1 Impacto do AM Supervisionado nos Resultados

Para a abordagem de AM supervisionada, comparamos o desempenho de três classificadores: Regressão Logística, Floresta Aleatória e *XGBoost*. Estes classificadores foram selecionados por possuírem baixo custo computacional e apresentarem resultados satisfatórios na literatura. Os resultados da aplicação dos algoritmos para o *dataset* de treinamento-validação são apresentados na Tabela 5.2.

Após executar os modelos treinados no *dataset* de teste, obtivemos os resultados apresentados na Tabela 5.3.

Tabela 5.3: Resultados de desempenho para o conjunto de dados de teste

Avaliação de Desempenho - *Dataset* de Teste - %			
Métrica	Regressão Logística	Floresta Aleatória	*XGBoost*
Acurácia	79,04	87,17	**95,72**
Precisão	77,38	**100**	**100**
Sensibilidade	68,42	77,38	**89,47**
F1-*score*	72,63	81,25	**94,44**
ROC-AUC	88,78	81,4	**97,5**

Embora o melhor modelo em relação aos resultados de treinamento-validação tenha sido o Floresta Aleatória, em novos dados, o *XGBoost* mostrou-se mais eficaz, pois apresenta valores mais altos para todas as métricas escolhidas (Tabela 5.3). Isto torna possível determinar que o *XGBoost*, face aos *datasets*, modelou os dados de melhor forma que os demais algoritmos, visto que os classificou de uma forma que otimizou valores de FP e FN, principalmente. A otimização destas classes é fundamental para a identificação precisa do que é realmente ataque e do que não é, possibilitando uma ferramenta mais precisa.

A Figura 5.1 mostra a curva ROC-AUC para os três classificadores com o conjunto de dados de teste. A linha tracejada em um gráfico ROC-AUC representa a curva de um classificador puramente aleatório. Em uma análise visual, quanto mais próxima uma curva estiver do canto superior esquerdo do gráfico, melhor será o classificador. Isso significa que este classificador tem uma alta taxa de verdadeiros positivos, um baixo número de falsos negativos, e uma baixa taxa de falsos positivos.

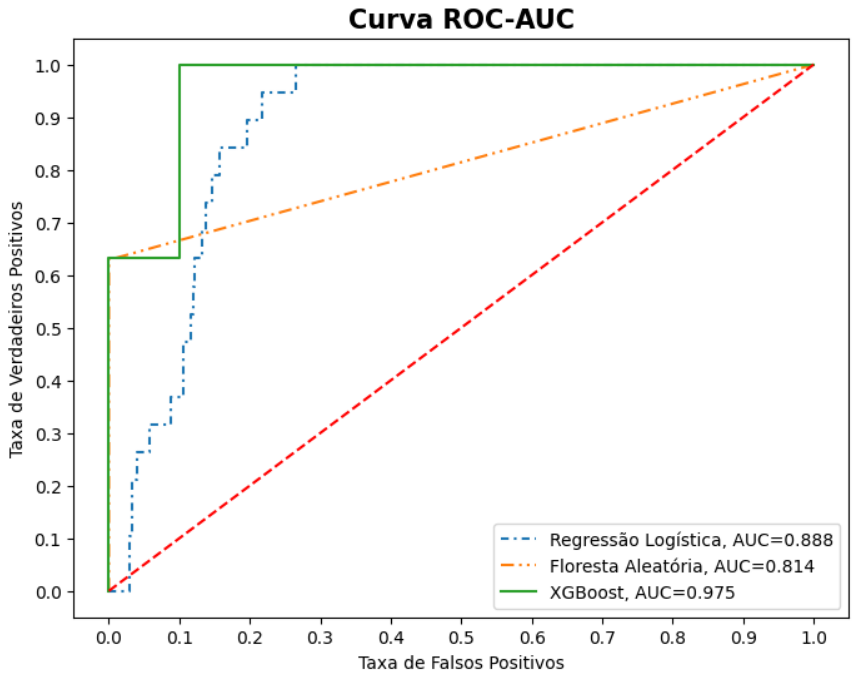


Figura 5.1: ROC-AUC para o conjunto de dados de teste.

Os valores de ROC-AUC para os classificadores de Regressão Logística e Floresta Aleatória foram menores, o que indica que o *XGBoost* é o melhor dos classificadores analisados para o problema. Vale ressaltar que ambos os classificadores, Floresta Aleatória e *XGBoost*, não tiveram FP no conjunto de dados de teste. No entanto, o modelo Floresta Aleatória teve uma taxa maior de FN, com base em seus resultados

de Sensibilidade e F1-*score* – a mesma análise se aplica ao classificador de Regressão Logística. Esta está longe de ser uma situação desejável, pois o classificador pode perder algumas informações valiosas no caso de um ataque a um SGBD. Ou seja, ataques que realmente estão sendo conduzidos deixarão de ser classificados como tal pelo classificador de Regressão Logística, em número maior que o *XGBoost*.

5.2.2 Impacto do AM Não Supervisionado (Detecção de Anomalias) nos Resultados

O primeiro algoritmo escolhido para executar uma DA no nosso *dataset* foi o *iForest*. Como nosso intuito inicial era validar que é possível realizar a DA para nossos *datasets*, escolhemos este que é um algoritmo bastante utilizado na literatura em problemas de DA, pois possui baixo custo computacional e bons resultados. Ele utiliza-se de Árvores de Decisão para tomada de decisão sobre os valores que lhe são apresentados.

Inicialmente, treinamos o modelo no *dataset* de treino balanceado, em que temos 53% de classes negativas (0, não ataque ou normal) e 47% de classes positivas (1, ataque ou anomalia). Aplicado o modelo treinado no *dataset* de treinamento, obtivemos as métricas relacionadas abaixo:

a) Acurácia: 68,07%

b) Sensibilidade: 83,33%

c) Precisão: 57,37%

d) F1-*score*: 67,95%

e) ROC-AUC: 70,48%

Esses valores foram influenciados por uma grande taxa de FN, conforme evidencia a Matriz de Confusão apresentada na Figura 5.2.

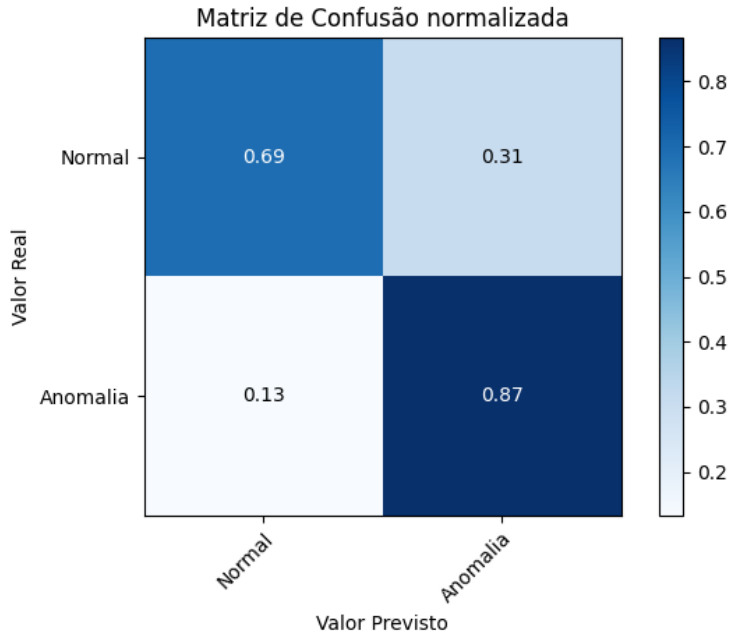


Figura 5.2: Matriz de Confusão normalizada para *dataset* de teste, treinado com *dataset* de treinamento balanceado

Cabe ressaltar que o *dataset* de treinamento original possui cerca de 3% de registros rotulados como ataques – portanto, anomalias –, o que condiz com um cenário realista de início de ataque. Assim, optamos por experimentar treinando-se o modelo com o *dastaset* original e aplicando-se ao *dataset* de testes, no qual obtivemos as seguintes métricas:

a) Acurácia: 73,27%

b) Sensibilidade: 82,81%

c) Precisão: 63,02%

d) F1-*score*: 74,78%

e) ROC-AUC: 71,57%

A matriz de confusão desse novo experimento encontra-se na Figura 5.3.

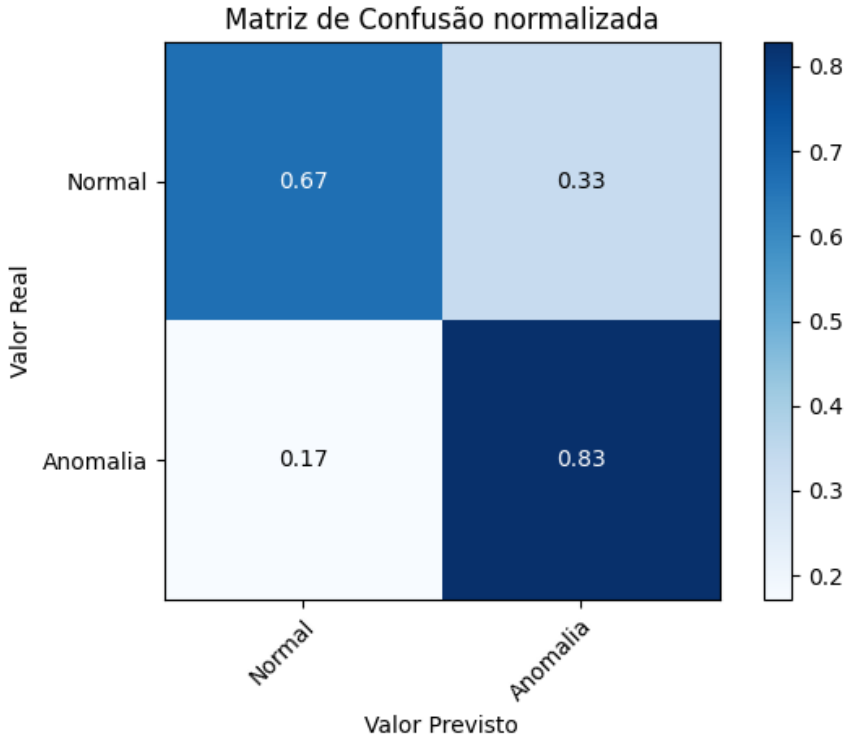


Figura 5.3: Matriz de Confusão normalizada – *dataset* de teste

Nota-se uma melhora importante nas métricas observadas (à exceção da Sensibilidade por uma pequena margem), o que nos indica que é a melhor forma de tratar o assunto sob a ótica da DA. Este fato explica-se por o *dataset* balanceado apresentar uma distribuição estatística completamente diferente daquilo que é observado no quotidiano. Assim, os *outliers* do modelo são diferentes do que se esperaria numa operação normal. Este fato pode ser observado pela Figura 5.4, que representa a plotagem da Figura 4.3, porém com os dados balanceados.

Validada nossa hipótese de que é possível determinar ataques a um SGBD por meio de DA sobre os *logs* de sessões dos usuários, partimos para o uso de abordagens de AP – a literatura tem focado em um maior uso de tais algoritmos. Entendemos que isso se deve ao aumento do poder de processamento disponível nos

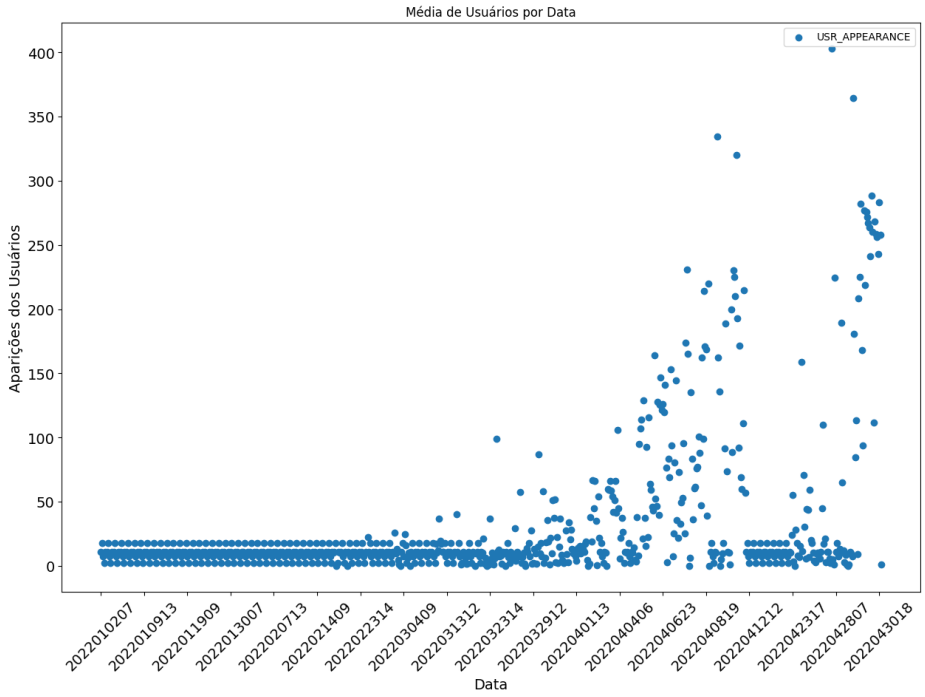


Figura 5.4: Média de sessões de usuários do *dataset* de treinamento balanceado.

últimos anos associado à diminuição de seus custos, o que facilita o uso desses algoritmos, que consomem mais recursos que os algoritmos tradicionais. Como forma de comparação, o algoritmo de AP mais lento empregado neste trabalho, o LSTM, foi cerca de trinta vezes mais lento para treinar do que o *iForest*.

Quanto à escolha dos algoritmos, optamos por escolher algoritmos que representam duas categorias diferentes: *Autoencoder*, uma RN que comprime e reconstrói os valores, de forma a se detectar erros na reconstrução, o que indicaria uma anomalia em valores de reconstrução altos; LSTM, um algoritmo para previsão de valores em sequência (*seq2seq*), em que prevê o próximo elemento e o erro desta previsão é comparado com o valor esperado, indicando uma anomalia em casos de erro mais alto.

A submissão dos dados ao *Autoencoder* trouxe resultados melhores em umas áreas em relação ao *iForest*, mas piores em outras. De acordo com a Matriz de Confusão gerada pelo resultado deste algoritmo, presente na Figura 5.7, nota-se que houve um número menor de FP, o que é muito bom, mas houve uma quantidade de FN um pouco mais alta que no *iForest*. Quanto mais FNs um modelo de AM para segurança cibernética gera, significa que mais ataques ele deixou de identificar e, portanto, menor a visibilidade de atividades maliciosas do ambiente para a equipe responsável.

As métricas obtidas pela execução do *Autoencoder* foram as seguintes:

a) Acurácia: 75,02%

b) Sensibilidade: 80,79%

c) Precisão: 65,65%

d) F1-*score*: 72,43%

e) ROC-AUC: 75,93%

A Figura 5.5 apresenta o gráfico de perda do modelo *Autoencoder*, tanto para o *dataset* de treinamento quanto para o *dataset* de teste. Este gráfico demonstra o quanto o erro do modelo variou por época de treinamento. Essa variação se deve aos ajustes de pesos e vieses promovidos pelo algoritmo de retropropagação.

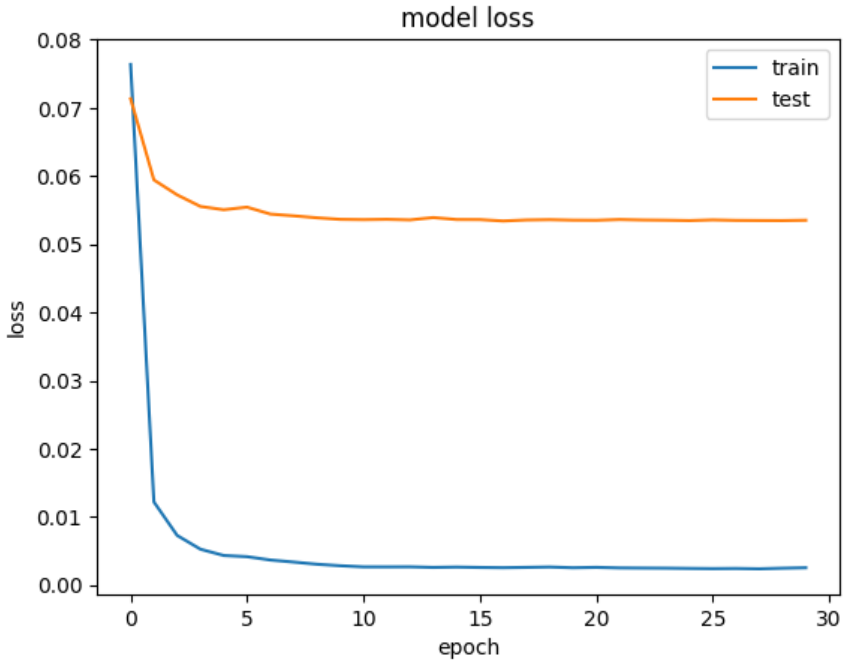


Figura 5.5: Evolução do erro de reconstrução do modelo *Autoencoder*.

A aplicação dos dados ao *Autoencoder* trouxe alguns aprendizados interessantes em relação ao limiar para determinação de uma anomalia. Nota-se que sua variação altera as taxa de FP e de FN dos resultados. Porém, essa mudança é inversamente proporcional: ao aumentar a taxa de FP, diminui-se a de FN e o contrário é verdadeiro. Este fato é notado pela Figura 5.6, em que estão apresentados valores de Precisão e Sensibilidade para diferentes patamares – estas métricas foram escolhidas, pois identificam indiretamente os valores de FP e FN. Esta particularidade tem nome: *tradeoff precision-recall* (24, 46).

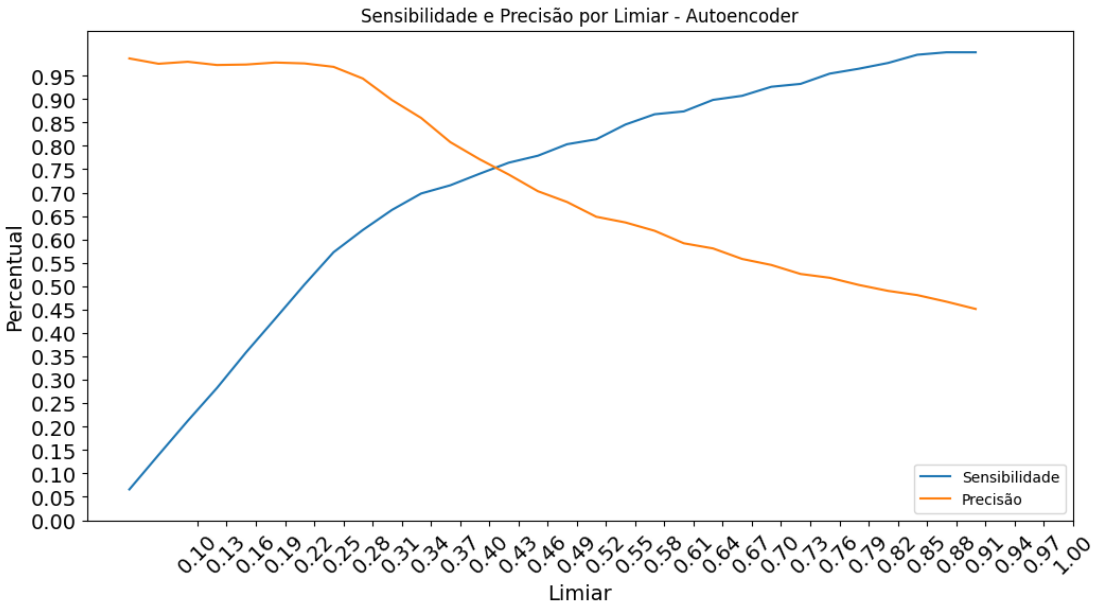


Figura 5.6: Relação entre Limiar e Sensibilidade e Precisão - *Autoencoder*.

Cabe, então, ao detentor de uma solução desta natureza ponderar qual o limiar mais útil para sua operação. Uma baixa taxa de FN, por exemplo, significa que a solução deixa passar incólumes poucos ataques. Porém, como efeito colateral, o sistema emitirá mais alertas de ataques do que o normal, o que pode acabar trazendo descrença na ferramenta por parte da equipe.

O limiar arbitrariamente escolhido por nós para apresentar as métricas foi o da intercessão entre ambos os gráficos, que é por volta de 0,4. Este valor representa o percentil final da distribuição, ou seja, há ainda 40% dos registros acima desse limiar. Não coincidentemente, ele é próximo ao percentual de registros anômalos para o *dataset* de teste. Este valor e o fenômeno observado condizem com duas das formas de definir um limiar citadas no Capítulo 2, na subseção 2.3.4, a saber: pelo percentual de registros anômalos do *dataset* e pela taxa de falsos positivos. Então, para o limiar escolhido, as métricas obtidas foram as seguintes:

a) Acurácia: 80,47%

b) Sensibilidade: 75,09%

c) Precisão: 76,43%

d) F1-*score*: 75,75%

e) ROC-AUC: 79,62%

A Matriz de Confusão resultante da escolha de limiar e do modelo foi a presente na Figura 5.7.

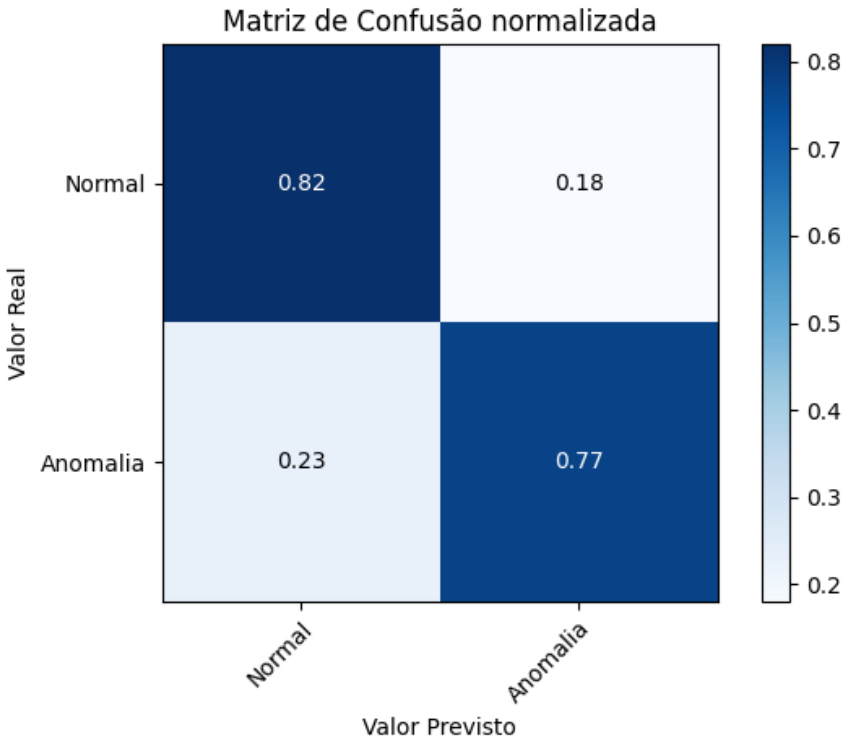


Figura 5.7: Matriz de Confusão normalizada – *Autoencoder* – *dataset* de teste.

Para a LSTM, obtivemos resultados similares aos do *Autoencoder*. As perdas do modelo acabaram por convergir melhor do que no *Autoencoder*, conforme se observa na Figura 5.8, possivelmente pela aplicação do processo de regularização do modelo. A Matriz de Confusão presente na Figura 5.10 denota que houve uma piora em relação às taxas de falso negativo para o mesmo limiar do outro modelo de AP. Este limiar também foi escolhido de acordo com o equilíbrio demonstrado pelo gráfico da Figura 5.9.

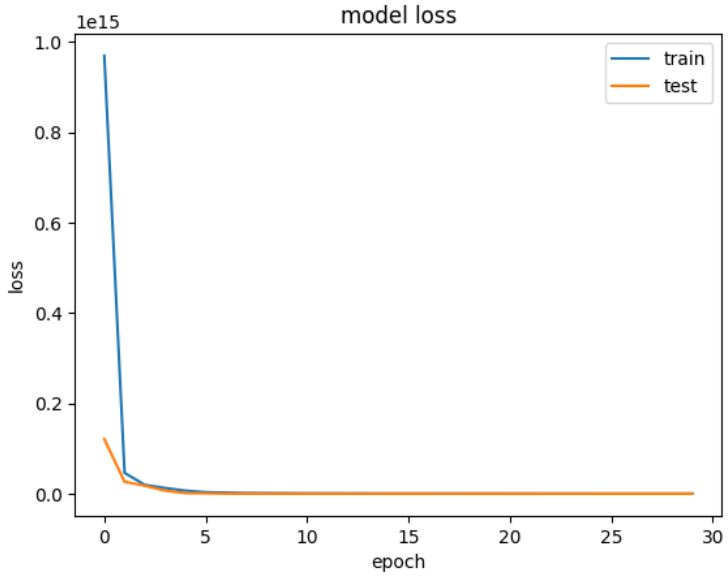


Figura 5.8: Evolução do erro de reconstrução do modelo LSTM.

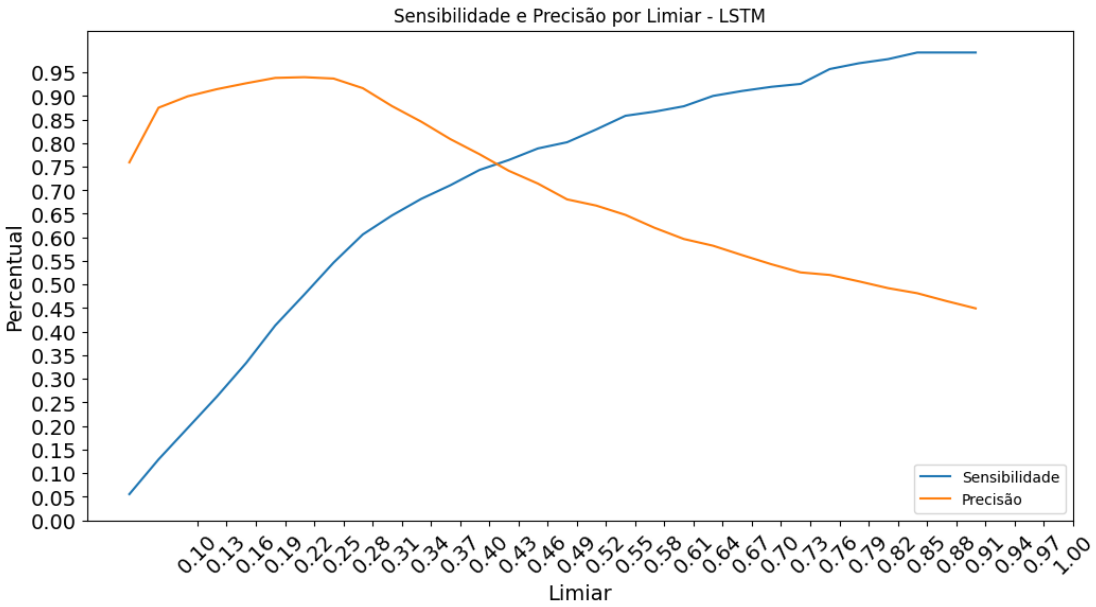


Figura 5.9: Relação entre Limiar e Sensibilidade e Precisão - LSTM.

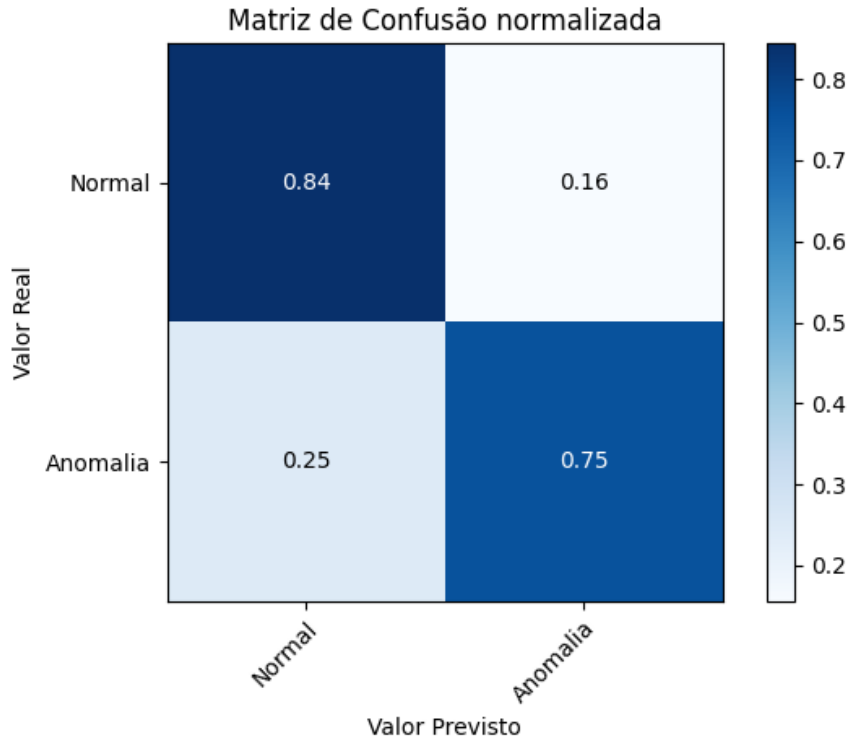


Figura 5.10: Matriz de Confusão normalizada – LSTM – *dataset* de teste.

As métricas obtidas pelo modelo LSTM estão descritas abaixo:

a) Acurácia: 80,72%

b) Sensibilidade: 75,35%

c) Precisão: 76,76%

d) F1-*score*: 76,05%

e) ROC-AUC: 79,87%

A Figura 5.11 apresenta as melhores métricas obtidas de todos os modelos, supervisionado e não supervisionado.

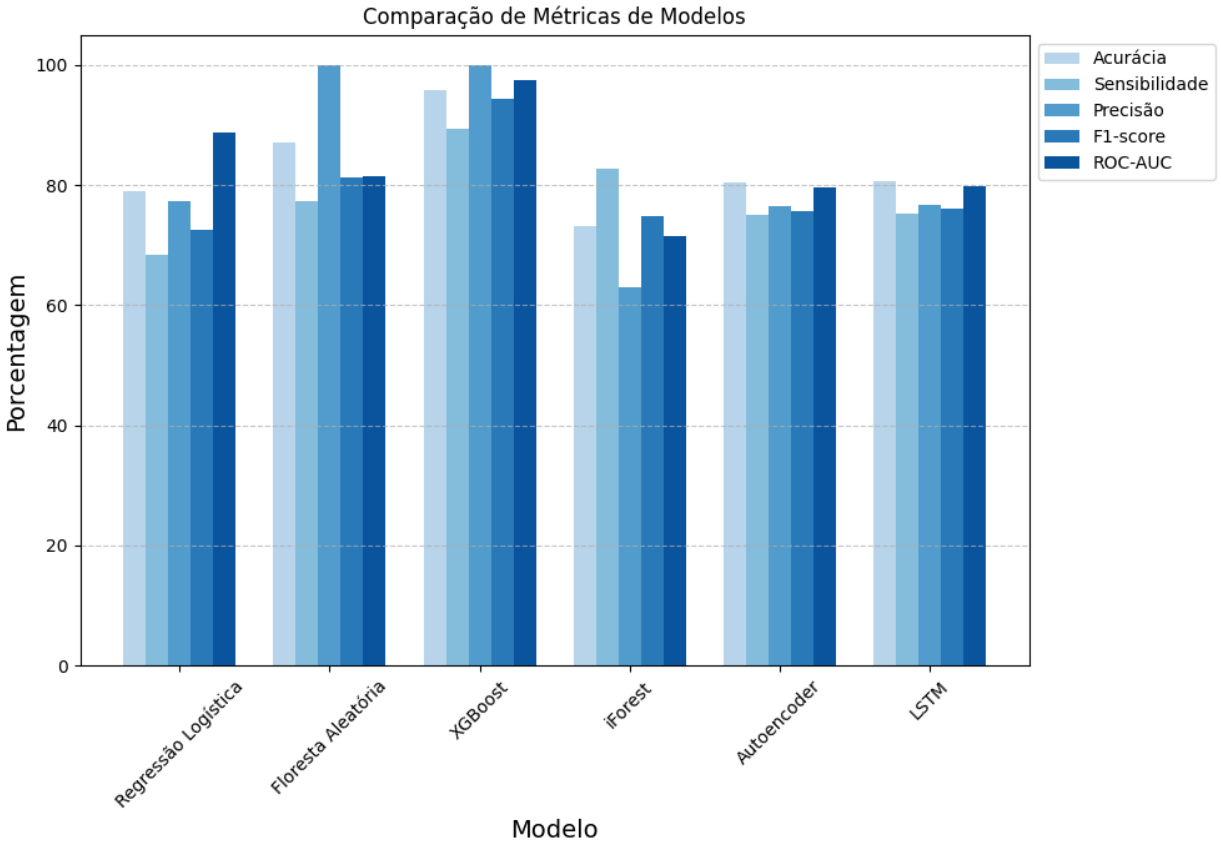


Figura 5.11: Melhores métricas para os modelos testados - *dataset* de teste.

5.3 DISCUSSÃO

A abordagem supervisionada permite-nos uma detecção de ataques SIDDoS com grande eficácia, conforme demonstrado pelas métricas obtidas na aplicação do modelo ao *dataset* de teste. Este fato é encorajador pois três motivos: os *datasets*, tanto de treinamento quanto de testes, são provenientes do mundo real, de um ataque efetivamente ocorrido. O segundo diz respeito a termos conseguido provar que é possível criar uma ferramenta adicional de segurança viável ao ambiente, de fácil implementação. O terceiro diz respeito à eficiência dessa detecção, com baixas taxas de FP e FN.

Há que se considerar, no entanto, que a abordagem supervisionada exige que os registros sejam previamente rotulados como normais ou ataques (ou anômalos). Este é um trabalho manual, executado por equipes especializadas, o que nem sempre uma organização dispõe. Este fato diminui um pouco a aplicabilidade da solução face a outros tipos de ataques. Nestes casos, a rotulação do *dataset* deveria ser multiclasse, em que deveriam ser identificados manualmente os padrões de cada tipo de ataque e, então, aplicar o respectivo rótulo. Para resolver esta dificuldade, investigamos a alternativa não supervisionada de DA.

A DA apresentou métricas com valores menores, porém ainda auspiciosos ao se considerar que os dados são provenientes de produção. Era de se esperar que isto ocorresse, visto que os algoritmos empregados não tinham conhecimento prévio do ataque. Mesmo assim, os resultados obtidos podem ser definidos como extremamente positivos, com leve vantagem para o modelo LSTM em todas as métricas. Isto possibilita uma proteção maior à infraestrutura, pois tem o potencial de identificar quaisquer atividades que fogem ao padrão e, portanto, maliciosas. Ou seja, não somente ataque SIDDoS poderá ser detectado pela solução, mas atividades diferentes daquelas normalmente empregadas na operação e provimento de serviço diários. Porém, há que se existir um estudo no ambiente em que for implantada ferramenta similar a respeito do limiar, pois ficou demonstrado que alteram as taxas de ataques verdadeiros não detectados ou detecta-se tráfego normal como ataques. Desta forma, este equilíbrio tem que ser escolhido da forma que melhor atenda aos anseios dos administradores.

Um uso muito interessante que pode ser dado à nossa ferramenta diz respeito a executar, inicialmente, a abordagem não supervisionada para novos dados e submeter as anomalias identificadas pelo modelo a uma análise de especialistas. Desta forma, seriam identificados com precisão o que é VP, FP ou FN e, então, aplicados os devidos rótulos ao *dataset*. A análise da equipe especializada se daria, então, para rotular corretamente os registros identificados como ataques, retirando manualmente os FPs. A partir deste novo *dataset* rotulado, treina-se o modelo supervisionado com os dados atualizados, criando-se, então, um modelo mais robusto e eficaz. Desta forma, pode-se aplicar este modelo supervisionado a novos dados que venham a surgir, detectando-se com mais precisão instâncias normais ou maliciosas. Este é um exemplo de um processo iterativo que é permitido e habilitado pela solução por nós desenvolvida, o que credencia esta solução como sendo uma possível camada de segurança a mais na infraestrutura.

Pode-se, como forma de potencializar este uso, utilizar um limiar diferente do escolhido neste trabalho e que privilegie a diminuição dos FNs - ou seja, aumentando a Sensibilidade. Porém, como efeito colateral, temos a diminuição da Precisão e, portanto, o aumento do número de FPs. Como exemplo, escolheríamos o limiar 0,85, que é o que promove um alto valor de Sensibilidade com a menor penalização à Precisão

possível, conforme visto na Figura 5.12.

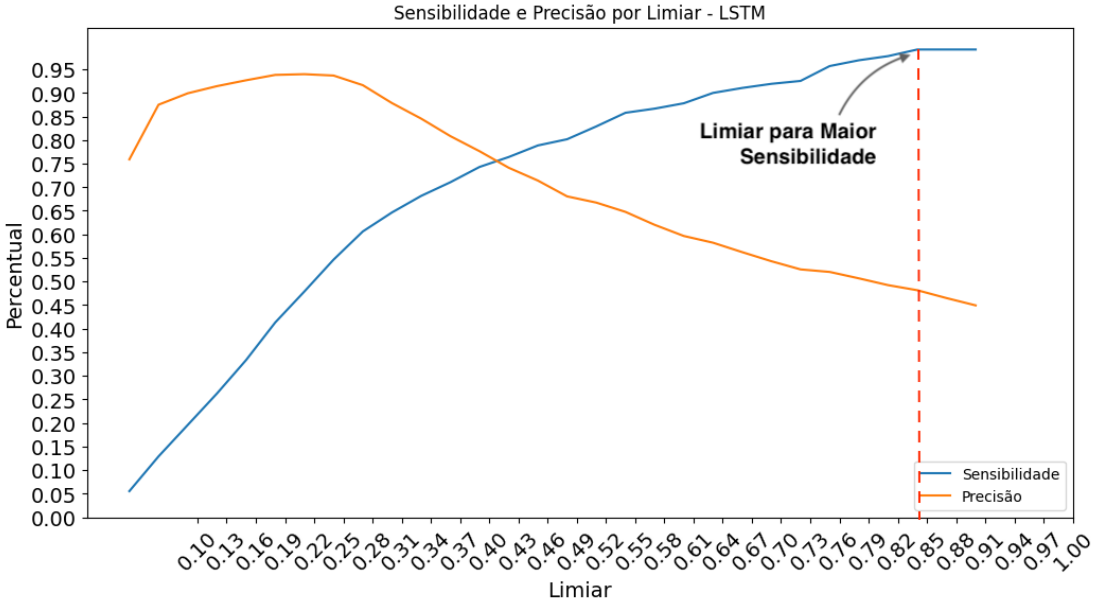


Figura 5.12: Escolha de limiar diferente a partir da curva Sensibilidade x Precisão

Como resultado desta estratégia, temos a diminuição drástica dos registros classificados como FN (ataques reais que não foram classificados como ataques). Porém, temos um aumento de registros incorretamente definidos como ataques, conforme se pode perceber pela Matriz de Confusão presente na Figura 5.13 e pelas métricas:

- Sensibilidade: 99,21%
- Precisão: 47,58%

Pode-se, também, escolher um limiar que privilegie a diminuição dos FPs, o que aumentaria, consequentemente, o número de FNs. Esta abordagem é embasada pelo fato de o nosso trabalho atuar numa camada mais profunda da infraestrutura. Desta forma, muito provavelmente há outros mecanismos de defesa anteriores a ela, que, cada um a sua maneira, aplica controles que poderiam identificar tráfegos que indiquem inequivocamente um ataque. Portanto, houve uma avaliação prévia que já detectou ataques menos furtivos, mais propagados pela literatura, deixando a última camada com uma detecção mais pontual. Escolhendo esta abordagem de diminuição de FPs, nossa ferramenta entregaria uma detecção mais precisa de atividades realmente maliciosas no SGBD. Como exemplo, temos as seguintes métricas para esta estratégia, representadas pela figura 5.14:

- Sensibilidade: 54,3%
- Precisão: 96,57%

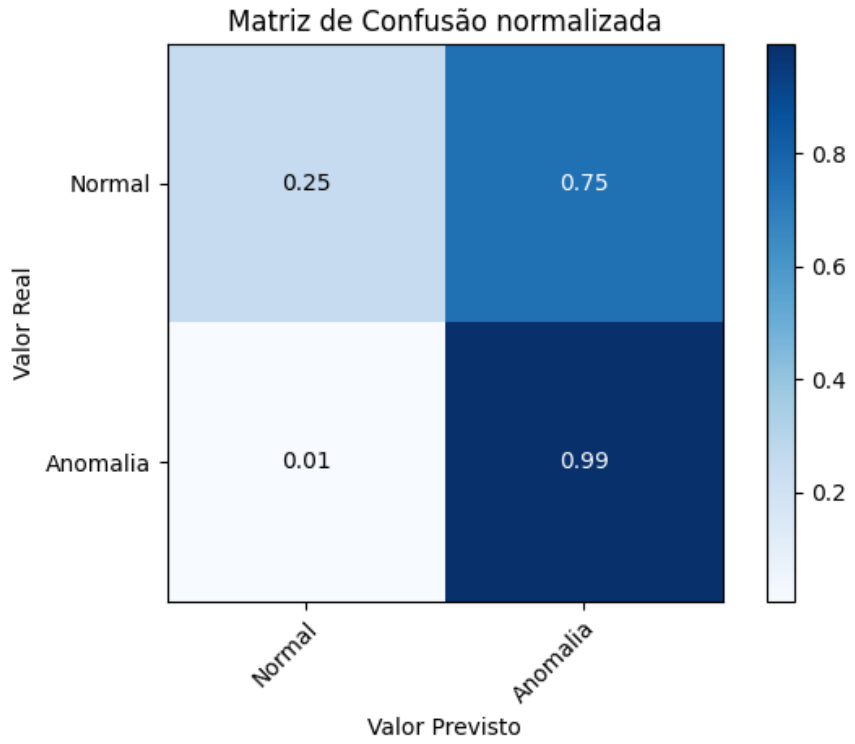


Figura 5.13: Matriz de Confusão para limiar escolhido.

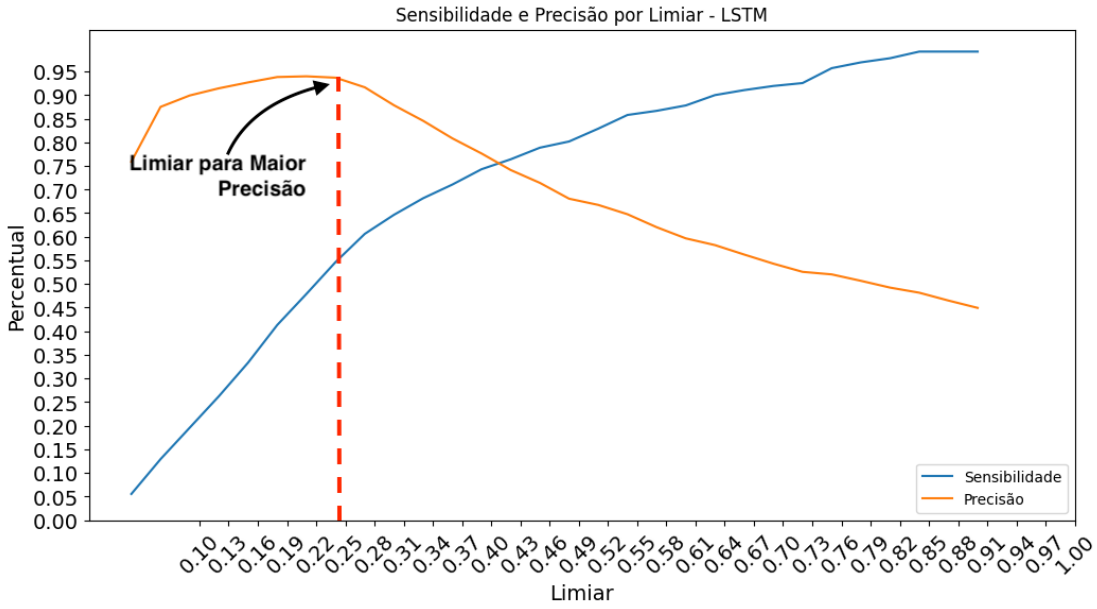


Figura 5.14: Escolha de limiar diferente a partir da curva Sensibilidade x Precisão

Como ponto positivo, esta abordagem traria uma maior confiança na ferramenta por parte da equipe de Segurança Cibernética. Ou seja, o ataque detectado é praticamente garantido que seja realmente um ataque e os mecanismos de mitigação definidos em seus processos internos poderiam ser colocados em ação. Confia-se, portanto, nos mecanismos para detecção de atividades maliciosas em outros pontos da infraestrutura e acrescenta-se uma ferramenta assertiva para detecção mais próxima aos dados armazenados

pelas aplicações.

Esta é uma escolha que a equipe de Segurança Cibernética pode tomar e que nossa ferramenta permite, o que demonstra sua efetividade e flexibilidade. É uma decisão crítica para o processo de Resposta a Incidentes de Segurança da Informação, pois afeta diretamente a capacidade de detecção e reação da equipe de Segurança Cibernética. Um limiar muito alto pode reduzir os FPs, mas também pode deixar passar ataques que estão abaixo do nível de alerta. Um limiar muito baixo pode aumentar os FNs, mas também pode gerar um excesso de alertas que sobrecarregam a equipe e dificultam a identificação das ameaças reais.

Ao utilizar nossa ferramenta, a equipe de Segurança Cibernética pode ajustar o limiar de acordo com o seu contexto, seus recursos e sua estratégia de defesa. A ferramenta permite visualizar o impacto da mudança do limiar nas métricas de desempenho da rede neural, como a Precisão, a Sensibilidade, a Especificidade e o valor preditivo positivo. Assim, a equipe pode escolher o limiar que melhor equilibra a taxa de detecção com a taxa de falsos alarmes, otimizando o processo de Resposta a Incidentes e aumentando o nível de segurança cibernética observado.

6 CONCLUSÃO

Nosso trabalho demonstrou a viabilidade de se adicionar uma camada de segurança a partir de dados internos ao SGBD. Este é um local de fonte de dados pouco usual na literatura, mas que encontra respaldo no fato de possuir informações de qualidade acerca de seus parâmetros de desempenho. Este fato pode ser usado a favor para detecção de atividades maliciosas no SGBD, conforme demonstrado aqui.

Tomando-se por base um ataque do mundo real e que contou com a classificação dos registros do *dataset* por equipe especializada, foi possível estabelecer que uma abordagem de AM supervisionada apresenta excelente desempenho. Porém, como essa classificação é algo trabalhoso e difícil de se suceder em tempo de operação, optamos por investigar, também, uma abordagem de AM não supervisionada por meio da DA. Obtivemos resultados promissores, em que o ataque perpetrado e registrado no *dataset* foi detectado com ótimos resultados. Isso abre espaço para que outras atividades maliciosas (não somente SIDDoS) sejam detectadas.

Assim, fica demonstrado que é possível utilizar-se de outra ferramenta para melhorar outros pilares de segurança da informação para um SGBD. Esta afirmação é corroborada pelo fato de ter sido utilizado um *dataset* de teste do mundo real. Isso mostra que não só é possível detectar ataques DDoS em um SGBD usando seus registros de dados, mas também é possível alcançar alta confiança nas previsões.

O trabalho desenvolvido apresentou desafios e consequentes oportunidades de crescimento e conhecimento acerca de segurança cibernética aplicada a SGBDs. A proteção destes tem sido menos explorada que a proteção de outros elementos de uma infraestrutura, como, por exemplo, servidores *web* ou de aplicação. Desta forma, nosso trabalho traz um olhar sobre este componente crítico de uma infraestrutura sob a ótica da segurança cibernética, apresentando ferramentas para detecção de ataques efetivas. Isto contribui para o aumento da segurança do ambiente computacional de uma instituição, pois é adicionada mais uma camada de segurança, corroborando um princípio da segurança cibernética de abordar a segurança em camadas.

6.1 LIMITAÇÕES

O trabalho apresenta, como limitação, apenas dois *datasets* com a mesma origem. Seria interessante, para pesquisas futuras, a coleta de novos dados, preferencialmente a partir de ataques reais. Caso não seja possível, a obtenção desses dados de forma sintética apresenta-se como alternativa. A detecção de apenas um ataque (o SIDDoS) neste momento - esta, mais uma limitação do trabalho - está diretamente ligada ao *dataset* coletado. Ou seja, a geração e/ou coleta de novo *dataset*, mais abrangente, possibilitará a aplicação do trabalho para outras ameaças a SGBDs.

Outra limitação diz respeito à estratégia escolhida de apenas detectar o ataque, ao invés de o prevenir. Compreende-se que a detecção é o primeiro passo para a prevenção, mas esta apresenta novos e possivelmente maiores desafios.

6.2 TRABALHOS FUTUROS

Como trabalhos futuros, entendemos ser possível explorar outras naturezas de ataques a SGBDs. Isto agrega maior valor à nossa solução, pois aumenta o escopo de proteção deste componente da infraestrutura. Algumas das possibilidades de trabalhos futuros são os seguintes:

- Coleta e/ou geração de novos *datasets* similares aos utilizados no nosso trabalho, preferencialmente englobando outros tipos de ataques a SGBDs: possibilitará a ampliação dos ataques detectados pela solução;
- Uso de aprendizado com reforço (*reinforcement learning*) para aprimoramento dos modelos de detecção de ataques aqui apresentados;
- Detecção de exfiltração de dados de um ambiente de SGBD: a partir de consultas executadas, dados retornados e o destino desses dados, detectar se os dados estão sendo enviados com o intuito de exfiltração;
- Detecção de ataques SQLIA genéricos usando análise multivariada (diferentes fontes de dados, de diferentes componentes, incluindo os *logs* internos do SGBD);
- Detecção de SQLIA avançados, como SQL *smuggling*: detecção de ataques SQLIA menos explorados na literatura, usando como *dataset* dados gerados dentro do SGBD;
- Uso de algoritmos estado-da-arte, como *Transformers* e RetNets (*Retentive Networks*), para DA;
- Prevenção de ataques, ao invés de sua simples detecção: proposição de uma arquitetura que permita identificar e prevenir ataques a SGBDs de forma automática.

REFERÊNCIAS BIBLIOGRÁFICAS

1 MAHJABIN, T.; XIAO, Y.; SUN, G.; JIANG, W. A survey of distributed denial-of-service attack, prevention, and mitigation techniques. *International Journal of Distributed Sensor Networks*, SAGE Publications Sage UK: London, England, v. 13, n. 12, p. 1550147717741463, 2017.

2 CHAGAS, D. A.; FILHO, G. P. R.; MENEGUETTE, R. I.; BONACIN, R.; GONÇALVES, V. P. Machine learning for detection of distributed denial-of-service attacks from queries executed in dbms. In: SBC. *Anais do XXVIII Workshop de Gerência e Operação de Redes e Serviços*. [S.l.], 2023. p. 57–70.

3 MORETTIN, P. A.; SINGER, J. M. *Estatıstica e Ciência de Dados*. [S.l.]: LTC, 2022. ISBN 978-85-216-3816-2.

4 POUYANFAR, S.; SADIQ, S.; YAN, Y.; TIAN, H.; TAO, Y.; REYES, M. P.; SHYU, M.-L.; CHEN, S.-C.; IYENGAR, S. S. A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys*, Association for Computing Machinery, New York, NY, USA, v. 51, n. 5, sep 2019. ISSN 0360-0300. Disponível em: <https://doi-org.ez54.periodicos.capes.gov.br/10.1145/3234150>.

5 BERMAN, D. S.; BUCZAK, A. L.; CHAVIS, J. S.; CORBETT, C. L. A survey of deep learning methods for cyber security. *Information*, MDPI, v. 10, n. 4, p. 122, 2019.

6 ZHANG, H.; NIAN, k.; COLEMAN, T.; LI, Y. Spectral ranking and unsupervised feature selection for point, collective, and contextual anomaly detection. *International Journal of Data Science and Analytics*, v. 9, 02 2020.

7 WILIEM, A.; MADASU, V.; BOLES, W.; YARLAGADDA, P. A context space model for detecting anomalous behaviour in video surveillance. In: . [S.l.: s.n.], 2012. p. 18–24. ISBN 978-1-4673-0798-7.

8 DWIVEDI, Y. K.; HUGHES, D. L.; COOMBS, C.; CONSTANTIOU, I.; DUAN, Y.; EDWARDS, J. S.; GUPTA, B.; LAL, B.; MISRA, S.; PRASHANT, P. et al. Impact of covid-19 pandemic on information management research and practice: Transforming education, work and life. *International journal of information management*, Elsevier, v. 55, p. 102211, 2020.

9 PRANGGONO, B.; ARABO, A. Covid-19 pandemic cybersecurity issues. *Internet Technology Letters*, Wiley Online Library, v. 4, n. 2, p. e247, 2021.

10 SILVA, R. F.; BARBOSA, R.; BERNARDINO, J. Intrusion detection systems for mitigating sql injection attacks: review and state-of-practice. *International Journal of Information Security and Privacy (IJISP)*, IGI Global, v. 14, n. 2, p. 20–40, 2020.

11 KUMAR, D.-N.; SRIRAMOJU, S.; SARMA, S. A study on machine learning techniques towards the detection of distributed denial of service attacks. In: . [S.l.: s.n.], 2018.

12 BROOKS, R. R.; YU, L.; OAKLEY, J.; TUSING, N. et al. Distributed denial of service (ddos): A history. *IEEE Annals of the History of Computing*, IEEE, 2021.

13 MEDEIROS, I.; BEATRIZ, M.; NEVES, N.; CORREIA, M. Septic: detecting injection attacks and vulnerabilities inside the dbms. *IEEE Transactions on Reliability*, IEEE, v. 68, n. 3, p. 1168–1188, 2019.

14 IRUNGU, J.; GRAHAM, S.; GIRMA, A.; KACEM, T. Artificial intelligence techniques for sql injection attack detection. In: *Proceedings of the 2023 8th International Conference on Intelligent Information Technology*. [S.l.: s.n.], 2023. p. 38–45.

15 ELMASRI, R.; NAVATHE, S. *Fundamentals of Database Systems*. 7. ed. USA: Pearson, 2016. ISBN Your-ISBN-Number-Here.

16 ALWAN, Z. S.; YOUNIS, M. F. Detection and prevention of sql injection attack: A survey. *International Journal of Computer Science and Mobile Computing*, v. 6, n. 8, p. 5–17, 2017.

17 VARSHNEY, K.; UJJWAL, R. Lssqlidp: Literature survey on sql injection detection and prevention techniques. *Journal of Statistics and Management Systems*, Taylor & Francis, v. 22, n. 2, p. 257–269, 2019.

18 ALIERO, M. S.; QURESHI, K. N.; PASHA, M. F.; GHANI, I.; YAURI, R. A. Systematic review analysis on sqlia detection and prevention approaches. *Wireless Personal Communications*, Springer, v. 112, n. 4, p. 2297–2333, 2020.

19 GURINA, A.; ELISEEV, V. Anomaly-based method for detecting multiple classes of network attacks. *Information*, MDPI, v. 10, n. 3, p. 84, 2019.

20 FILHO, F. S. d. L.; SILVEIRA, F. A.; JUNIOR, A. de M. B.; VARGAS-SOLAR, G.; SILVEIRA, L. F. Smart detection: an online approach for dos/ddos attack detection using machine learning. *Security and Communication Networks*, Hindawi, v. 2019, 2019.

21 GÉRON, A. Hands-on machine learning with scikit-learn and tensorflow: Concepts. *Tools, and Techniques to build intelligent systems*, O'Reilly Media, 2017.

22 GORMEZ, Y.; AYDIN, Z.; KARADEMIR, R.; GUNGOR, V. C. A deep learning approach with bayesian optimization and ensemble classifiers for detecting denial of service attacks. *International Journal of Communication Systems*, Wiley Online Library, v. 33, n. 11, p. e4401, 2020.

23 KAUR, H.; PANNU, H. S.; MALHI, A. K. A systematic review on imbalanced data challenges in machine learning: Applications and solutions. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 52, n. 4, p. 1–36, 2019.

24 LANDAUER, M.; ONDER, S.; SKOPIK, F.; WURZENBERGER, M. Deep learning for anomaly detection in log data: A survey. *Machine Learning with Applications*, v. 12, p. 100470, 2023. ISSN 2666-8270. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2666827023000233>.

25 JAVAHERI, D.; GORGIN, S.; LEE, J.-A.; MASDARI, M. Fuzzy logic-based ddos attacks and network traffic anomaly detection methods: Classification, overview, and future perspectives. *Information Sciences*, v. 626, p. 315–338, 2023. ISSN 0020-0255. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0020025523000683>.

26 MOLINA, A.; GONçALVES, V.; JR., R. S.; GIUNTINI, F.; PESSIN, G.; MENEGUETTE, R.; FILHO, G. R. Weapon: Uma arquitetura para detecção de anomalias de comportamento do usuário. In: *Anais do XI Brazilian Workshop on Social Network Analysis and Mining*. Porto Alegre, RS, Brasil: SBC, 2022. p. 121–132. ISSN 2595-6094. Disponível em: <https://sol.sbc.org.br/index.php/brasnam/article/view/20522>.

27 PANG, G.; SHEN, C.; CAO, L.; HENGEL, A. V. D. Deep learning for anomaly detection: A review. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 54, n. 2, mar 2022. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3439950>.

28 POKHREL, R. *Anomaly based–intrusion detection system using user profile generated from system logs*. Tese (Doutorado) — Pulchowk Campus, 2019.

29 AKGUN, D.; HIZAL, S.; CAVUSOGLU, U. A new ddos attacks intrusion detection model based on deep learning for cybersecurity. *Computers & Security*, Elsevier, v. 118, p. 102748, 2022.

30 SOFI, I.; MAHAJAN, A.; MANSOTRA, V. Machine learning techniques used for the detection and analysis of modern types of ddos attacks. *Int. Res. J. Eng. Technol*, 2017.

31 MITTAL, M.; KUMAR, K.; BEHAL, S. Deep learning approaches for detecting ddos attacks: a systematic review. *Soft Computing*, Springer, p. 1–37, 2022.

32 KUROSE, K. R. *Computer networking: A top-down approach by james*. [S.l.: s.n.], 2017. 601 p.

33 TOGATOROP, P.; SITORUS, H. A. T.; SIRAIT, R. M.; MANURUNG, T. Database audit system design and implementation. *Jurnal Mantik*, v. 5, n. 4, p. 2535–2541, 2022.

34 GOODRICH, M.; TAMASSIA, R. *Introdução à Segurança de Computadores*. Bookman, 2013. ISBN 9788540701939. Disponível em: <https://books.google.com.br/books?id=F5Guvy6IRLoC>.

35 THOMAS, J. A case study analysis of the equifax data breach 1 a case study analysis of the equifax data breach. 12 2019.

36 CHAHAL, J. K.; BHANDARI, A.; BEHAL, S. Distributed denial of service attacks: A threat or challenge. *New Review of Information Networking*, Taylor & Francis, v. 24, n. 1, p. 31–103, 2019.

37 HAIDER, S.; AKHUNZADA, A.; MUSTAFA, I.; PATEL, T. B.; FERNANDEZ, A.; CHOO, K.-K. R.; IQBAL, J. A deep cnn ensemble framework for efficient ddos attack detection in software defined networks. *Ieee Access*, IEEE, v. 8, p. 53972–53983, 2020.

38 TRIPATHI, N.; HUBBALLI, N. Application layer denial-of-service attacks and defense mechanisms: a survey. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 54, n. 4, p. 1–33, 2021.

39 de Neira, A. B.; KANTARCI, B.; NOGUEIRA, M. Distributed denial of service attack prediction: Challenges, open issues and opportunities. *Computer Networks*, v. 222, p. 109553, 2023. ISSN 1389-1286. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128622005874>.

40 CHADD, A. Ddos attacks: past, present and future. *Network Security*, MA Business London, v. 2018, n. 7, p. 13–15, 2018.

41 SAHOO, K. S.; PANDA, S. K.; SAHOO, S.; SAHOO, B.; DASH, R. Toward secure software-defined networks against distributed denial of service attack. *The Journal of Supercomputing*, Springer, v. 75, p. 4829–4874, 2019.

42 VEDULA, V.; LAMA, P.; BOPPANA, R. V.; TREJO, L. A. On the detection of low-rate denial of service attacks at transport and application layers. *Electronics*, MDPI, v. 10, n. 17, p. 2105, 2021.

43 ALLEN, J. H. *The CERT guide to system and network security practices*. [S.l.: s.n.], 2001.

44 RANGARAJU, N. K.; SRIRAMOJU, S. B.; SARMA, S. A study on machine learning techniques towards the detection of distributed denial of service attacks. *International Journal of Pure and Applied Mathematics*, v. 120, n. 6, p. 7407–7423, 2018.

45 TANDON, R. A survey of distributed denial of service attacks and defenses. *arXiv preprint arXiv:2008.01345*, 2020.

46 HASHEM, I.; ISLAM, M.; HAQUE, S. M.; JABED, Z. I.; SAKIB, N. A proposed technique for simultaneously detecting ddos and sql injection attacks. *International Journal of Computer Applications*, v. 975, p. 8887, 2021.

47 GÜMÜŞBAŞ, D.; YILDIRIM, T.; GENOVESE, A.; SCOTTI, F. A comprehensive survey of databases and deep learning methods for cybersecurity and intrusion detection systems. *IEEE Systems Journal*, IEEE, v. 15, n. 2, p. 1717–1731, 2020.

48 SOUZA, A.; NOBRE, R.; GONçALVES, V.; FILHO, G. R. Uma solução em névoa via objetos inteligentes para lidar com a heterogeneidade dos dados em um ambiente residencial. In: *Anais Estendidos do XXXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Porto Alegre, RS, Brasil: SBC, 2021. p. 257–264. ISSN 2177-9384. Disponível em: <https://sol.sbc.org.br/index.php/sbrc_estendido/article/view/17179>.

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

50 GORISHNIY, Y.; RUBACHEV, I.; KHRULKOV, V.; BABENKO, A. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, v. 34, p. 18932–18943, 2021.

51 NAEEM, S.; ALI, A.; ANAM, S.; AHMED, M. M. An unsupervised machine learning algorithms: Comprehensive review. *Int. J. Comput. Digit. Syst*, 2023.

52 THUDUMU, S.; BRANCH, P.; JIN, J.; SINGH, J. J. A comprehensive survey of anomaly detection techniques for high dimensional big data. *Journal of Big Data*, Springer, v. 7, n. 1, p. 1–30, 2020.

53 GOLDSTEIN, M.; UCHIDA, S. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS one*, Public Library of Science San Francisco, CA USA, v. 11, n. 4, p. e0152173, 2016.

54 HUANG, S.; LIU, Y.; FUNG, C.; HE, R.; ZHAO, Y.; YANG, H.; LUAN, Z. Hitanomaly: Hierarchical transformers for anomaly detection in system log. *IEEE transactions on network and service management*, IEEE, v. 17, n. 4, p. 2064–2076, 2020.

55 GULLI, A.; KAPOOR, A.; PAL, S. *Deep learning with TensorFlow 2 and Keras: regression, ConvNets, GANs, RNNs, NLP, and more with TensorFlow 2 and the Keras API*. [S.l.]: Packt Publishing Ltd, 2019.

56 GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

57 HOJJATI, H.; HO, T. K. K.; ARMANFARD, N. *Self-Supervised Anomaly Detection: A Survey and Outlook*. 2023.

58 SHARMA, B.; POKHAREL, P.; JOSHI, B. User behavior analytics for anomaly detection using lstm autoencoder-insider threat detection. In: *Proceedings of the 11th international conference on advances in information technology*. [S.l.: s.n.], 2020. p. 1–9.

59 CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, ACM New York, NY, USA, v. 41, n. 3, p. 1–58, 2009.

60 WANG, G. et al. A transformer autoencoder with hidden layer extension for multivariate time series anomaly detection. 2023.

61 KERNBACH, J. M.; STAARTJES, V. E. Foundations of machine learning-based clinical prediction modeling: Part ii—generalization and overfitting. *Machine Learning in Clinical Neuroscience*, Springer, p. 15–21, 2022.

62 BURKOV, A. *Machine Learning Engineering*. [S.l.]: Andriy Burkov, 2020. ISBN 9781777005467.

63 FONSECA, J.; VIEIRA, M.; MADEIRA, H. Online detection of malicious data access using dbms auditing. In: *Proceedings of the 2008 ACM symposium on Applied computing*. [S.l.: s.n.], 2008. p. 1013–1020.

64 LI, S.; YIN, Q.; LI, G.; LI, Q.; LIU, Z.; ZHU, J. Unsupervised contextual anomaly detection for database systems. In: *Proceedings of the 2022 International Conference on Management of Data*. [S.l.: s.n.], 2022. p. 788–802.

65 BRUNSWICK, U. U. of N. *NSL-KDD dataset*. 2023. <https://www.unb.ca/cic/datasets/nsl.html>. [Online; acessado em 25-10-2023].

66 ORACLE. *Database Reference - V$SESSION*. 2023. <https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/V-SESSION.html#GUID-28E2DC75-E157-4C0A-94AB-117C205789B9>. [Online; accessed in 09-02-2023].

67 YU, L.; ZHOU, R.; CHEN, R.; LAI, K. K. Missing data preprocessing in credit classification: One-hot encoding or imputation? *Emerging Markets Finance and Trade*, Taylor & Francis, v. 58, n. 2, p. 472–482, 2022.

68 DOAN, Q. H.; MAI, S.-H.; DO, Q. T.; THAI, D.-K. A cluster-based data splitting method for small sample and class imbalance problems in impact damage classification[formula presented]. *Applied Soft Computing*, v. 120, 2022. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85125567437&doi=10.1016\%2fj.asoc.2022.108628&partnerID=40&md5=3459fbaaeac966f4de7f8a74a8484972>.

APÊNDICES

I. CÓDIGO-FONTE DOS MODELOS DE DA

```
import pandas as pd
import numpy as np
import warnings
from sklearn.ensemble import IsolationForest
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, recall_score, precision_score
from keras.models import Model, load_model, Sequential
from keras.layers import Input, Dense, Dropout, LSTM, TimeDistributed, RepeatVector
from keras.callbacks import ModelCheckpoint, TensorBoard
from tensorflow.keras.optimizers import Adam
from keras import regularizers
import matplotlib.pyplot as plt
import seaborn as sns
from pylab import rcParams

# Importa bibliotecas proprias
import sys
sys.path.append('../Bibliotecas')
from wrangling import *
from graphs import *
from models import *

def transform_dataframe_into_sequence(df: pd.DataFrame, sequence_length: int=10):
    sequences = []
    for i in range(0, len(df) - sequence_length + 1):
        sequence = df.iloc[i:i + sequence_length, :]
        sequences.append(sequence)

    return sequences

def reshape_sequence_into_3d_array(df: pd.DataFrame, sequence_length: int=10):
    # Transforma o dataframe em uma sequencia
    sequences = transform_dataframe_into_sequence(df, sequence_length)
    # Faz o reshape das sequencias para terem 3 dimensoes, necessarias para
    # uma LSTM
    reshaped_sequence = np.array(sequences)

    return reshaped_sequence.reshape((reshaped_sequence.shape[0],
            reshaped_sequence.shape[1], reshaped_sequence.shape[2]))

# Modelo Autoencoder
def autoencoder(X: pd.DataFrame, hidden_dim: int=4, n_epochs: int=30,
                batch_size: int=128, learning_rate: float=0.1,
                encoder_dim_activation: str='tanh',
```

```
                encoder_hidden_dim_activation: str='relu',
                decoder_hidden_dim_activation: str='relu',
                decoder_dim_activation: str='sigmoid',
                decoder_activation: str='sigmoid'):
    input_dim = X.shape[1]
    encoding_dim = int(input_dim / 2)
    hidden_dim = int(encoding_dim / 2)

    input_layer = Input(shape=(input_dim, ))

    encoder = Dense(encoding_dim, activation=encoder_dim_activation,
                activity_regularizer=regularizers.l1(10e-5))(input_layer) # Camada
                # de compressao com regularizacao
    encoder = Dense(hidden_dim, activation=encoder_hidden_dim_activation)(encoder)
    decoder = Dense(hidden_dim, activation=decoder_hidden_dim_activation)(encoder)
    decoder = Dense(encoding_dim, activation=decoder_dim_activation)(encoder)
    decoder = Dense(input_dim, activation=decoder_activation)(decoder)

    autoencoder = Model(inputs=input_layer, outputs=decoder)

    return autoencoder

# Modelo LSTM
def lstm(X: pd.DataFrame, X_test: pd.DataFrame, encoding_dim: int=8,
            hidden_dim: int=4, n_epochs: int=30,
            batch_size: int=128, learning_rate: float=0.1,
            encoder_dim_activation: str='sigmoid',
            encoder_hidden_dim_activation: str='relu',
            decoder_hidden_dim_activation: str='relu',
            decoder_dim_activation: str='sigmoid',
            decoder_activation: str='sigmoid'):
    features = X.columns.to_list()

    X_train_copy = X[features].copy()
    X_test_copy = X_test[features].copy()
    scaler = MinMaxScaler()
    X_train_copy[features] = scaler.fit_transform(X_train_copy[features])
    X_test_copy[features] = scaler.fit_transform(X_test_copy[features])

    # Prepara a entrada para a LSTM
    # Reshape das entradas para a LSTM
    X_train_reshaped = reshape_sequence_into_3d_array(X_train, 10)
    X_test_reshaped = reshape_sequence_into_3d_array(X_test, 10)


    # Define p modelo LSTM
    lstm = Sequential()
    lstm.add(LSTM(units=50, activation=encoder_hidden_dim_activation,
             return_sequences=True, input_shape=(X_train_reshaped.shape[1],
             X_train_reshaped.shape[2])))
    lstm.add(Dropout(0.2)) # Camada de regularizacao
    lstm.add(LSTM(units=50, activation=encoder_hidden_dim_activation,
```

```
98              return_sequences=True))
99      lstm.add(Dropout(0.2)) # Camada de regularizacao
100     lstm.add(LSTM(units=50, activation=encoder_hidden_dim_activation))
101     lstm.add(Dropout(0.2)) # Camada de regularizacao
102     lstm.add(Dense(units=X_train_reshaped.shape[2]))  # A camada de saida tem o
103     # mesmo numero de neuronios que a camada de entrada
104
105     return lstm
106
107 def neural_network(X: pd.DataFrame, y, X_test:pd.DataFrame, y_test, nn_type:
108                    str='autoencoder', encoding_dim: int=8, hidden_dim: int=4,
109                    n_epochs: int=30, batch_size: int=128,
110                    learning_rate: float=0.1):
111     X_train = X.copy()
112     y_train = y.copy()
113     X_test_copy = X_test.copy()
114     y_test_copy = y_test.copy()
115
116     if nn_type == 'autoencoder':
117         nn = autoencoder(X_train, encoding_dim, hidden_dim, n_epochs, batch_size,
118                          learning_rate)
119     else:
120         nn = lstm(X=X_train, X_test=X_test, encoding_dim=encoding_dim,
121                   hidden_dim=hidden_dim, n_epochs=n_epochs, batch_size=batch_size,
122                   learning_rate=learning_rate)
123
124     nn.compile(optimizer='adam',
125                    loss='mean_squared_error',
126                    metrics=['accuracy'])
127
128     if nn_type == 'lstm':
129         X_train = reshape_sequence_into_3d_array(X_train, 10)
130         X_test_copy = reshape_sequence_into_3d_array(X_test_copy, 10)
131         history = nn.fit(X_train, y_train[:-9],
132                          epochs=n_epochs,
133                          batch_size=batch_size,
134                          shuffle=True,
135                          validation_data=(X_test_copy, y_test_copy),
136                          verbose=1,
137                          callbacks=[checkpointer, tensorboard]).history
138     else:
139         history = nn.fit(X_train, y_train,
140                          epochs=n_epochs,
141                          batch_size=batch_size,
142                          shuffle=True,
143                          validation_data=(X_test, y_test),
144                          verbose=1,
145                          callbacks=[checkpointer, tensorboard]).history
146
147
148     prediction = nn.predict(X_test_copy)
149
```

```
150         return history, prediction
```

II. CÓDIGO-FONTE AM NÃO SUPERVISIONADO (DETECÇÃO DE ANOMALIAS)

```
import pandas as pd
import numpy as np
import warnings
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, recall_score, precision_score
from keras.models import Model, load_model, Sequential
from keras.layers import Input, Dense, Dropout, LSTM, TimeDistributed, RepeatVector
from keras.callbacks import ModelCheckpoint, TensorBoard
from tensorflow.keras.optimizers import Adam
from keras import regularizers
import matplotlib.pyplot as plt
import seaborn as sns
from pylab import rcParams

# Importa bibliotecas proprias
import sys
sys.path.append('../Bibliotecas')
from wrangling import *
from graphs import *
from models import *

##############################################################
# Inicializacoes
##############################################################
warnings.filterwarnings('ignore')

# configurando pandas para mostrar todas as linhas e colunas
pd.set_option('display.max_rows', 150) # None para mostrar todas as linhas
pd.set_option('display.max_columns', None)

# configurando pandas para nao mostrar notacao cientifica para numeros
pd.set_option('display.float_format', lambda x: '%.2f' % x)

caminho = "/Users/daniloamchagas/mestrado/"

dataset_com_anomalias = pd.read_parquet(caminho +
        'transformed_dataset_new.parquet.gzip') # Dataset com anomalias
dataset = pd.read_parquet(caminho + 'transformed_dataset.parquet.gzip')
dataset_teste = pd.read_parquet(caminho +
        'transformed_test_dataset.parquet.gzip') # Dataset de testes


```

```
colunas_a_retirar = ['Year', 'Month', 'Day', 'Hour', 'Minute', 'Second'] # Colunas
# retiradas para nao influenciar na DA
dataset = dataset.drop(colunas_a_retirar, axis=1)
dataset_com_anomalias = dataset_com_anomalias.drop(colunas_a_retirar, axis=1)
dataset_teste = dataset_teste.drop(colunas_a_retirar, axis=1)

colunas_excesso = [coluna for coluna in dataset.columns if coluna not in
        dataset_teste.columns]
dataset_teste[colunas_excesso] = 0

# Cria dataset para previsao, sem variavel alvo
numeric_columns = return_numeric_columns(dataset)
X_train = dataset[numeric_columns].drop('isAttack', axis=1)
y_train = dataset['isAttack'] # variavel alvo
X_test = dataset_teste.drop('isAttack', axis=1)
X_test = X_test[X_train.columns]
y_test = dataset_teste['isAttack'] # variavel alvo

# Treino e previsao com Floresta de Isolamento (iForest)
pred_X_teste, pred_scores_X_teste = return_scores_iForest_model(X_train, X_teste,
        contamination=0.05, n_estimators=100, verbose=0)
pred_X_teste = pred_X_teste * -1 # Ajuste do score retornado, que e 1 para normal
# e -1 pra anomalia
X_test_iForest = X_test.copy()
X_test_iForest['anomaly'] = pred_X_test
outliers_teste = X_test_iForest.loc[X_test_iForest['anomaly']==1]
outlier_index_teste = list(outliers_teste.index)
X_test_iForest = X_test_iForest.drop('anomaly', axis=1)

# Transforma os valores -1 (normal) em 0
adjusted_pred_X_test_iForest = [1 if x == 1 else 0 for x in pred_X_test]

# Imprime as metricas para o iForest
print_classification_metrics(y_test, adjusted_pred_X_test_iForest)

# Gera grafico Matriz de Confusao normalizada
plot_confusion_matrix(y_test, adjusted_pred_X_test_iForest, classes=['Normal',
        'Anomalia'], normalize=True, language='pt-br')

# Agrega dataset de treino para gerar graficos de anomalias por periodo do dia
X_agg_sum_dataset = dataset.groupby(index_columns).sum().drop(columns_to_drop,
        axis=1)
y_agg_sum_dataset = X_agg_sum_dataset['isAttack']
y_adjusted_agg_sum_dataset = [1 if x > 0 else 0 for x in y_agg_sum_dataset]
X_agg_sum_dataset.drop('isAttack', axis=1, inplace=True)

pred_sum, pred_sum_scores = return_scores_iForest_model(X_agg_sum_dataset,
        contamination=contamination, verbose=verbose, n_estimators=n_estimators)
pred_sum = pred_sum * -1
X_agg_sum_dataset['anomaly'] = pred_sum

# Gera Figura 4.4
```

```
generate_scatter_plot_anomaly_detection(X_agg_sum_dataset, 'Date_Time_Period',
        'USR_APPEARANCE', pred_sum, nbins=20, isXaxisIndex=True)

# Reduz a dimensao do dataset para 3 dimensoes para plotar um grafico 3D
X_PCA = X_agg_sum_dataset.copy()
X_PCA = X_PCA.reset_index()
outliers = X_PCA.loc[X_PCA['anomaly']==1]
outlier_index = list(outliers.index)

# Gera grafico 3D das anomalias previstas pelo iForest
generate_3D_anomaly_graph_PCA(X_PCA, outlier_index)

# Gera grafico 3D no Plotly, interativo, das anomalias previstas pelo iForest
generate_3D_anomaly_graph_PCA_plotly(X_PCA, outlier_index, X_PCA['anomaly'])

# Treinamento e previsao do modelo Autoencoder
history_autoencoder, prediction_autoencoder = neural_network(X_train, y_train,
        X_test, y_test, 'autoencoder', encoding_dim=8, hidden_dim=4, n_epochs=30,
        batch_size=128, learning_rate=0.1)

# Treinamento e previsao do modelo LSTM
history_lstm, prediction_lstm = neural_network(X=X_train, y=y_train, X_test=X_test,
        y_test=y_test, nn_type='lstm', encoding_dim=8, hidden_dim=4, n_epochs=30,
        batch_size=128, learning_rate=0.1)

# Gera graficos das perdas de treinamento e teste
plot_model_loss(history_autoencoder)
plot_model_loss(history_lstm)

# Gera as pontuacoes de anomalias de ambos os modelos
X_test_copy = X_test.copy()
X_test_copy['MSE'] = 0
scaled_data = MinMaxScaler().fit_transform(X_test)
mse_autoencoder = np.mean(np.power(scaled_data - prediction_autoencoder, 2), axis=1)
mse_lstm = np.mean(np.power(scaled_data - prediction_lstm, 2), axis=1)
X_test_copy['MSE_autoencoder'] = mse_autoencoder
X_test_copy['MSE_lstm'] = mse_lstm

# Limiar para dataset de teste
mse_threshold_autoencoder = np.quantile(mse_autoencoder, 0.6)
mse_threshold_lstm = np.quantile(mse_lstm, 0.6)

dataset_after_neural_network = pd.concat([X_test_copy, y_test], axis=1)

# Determina, para ambos os modelos, se e anomalia ou nao de acordo com o limiar
# escolhido
dataset_after_neural_network['anomaly_autoencoder'] = [1 if mse_autoencoder >
        mse_threshold_autoencoder else 0 for mse_autoencoder in
        dataset_after_neural_network['MSE_autoencoder']]
dataset_after_neural_network['anomaly_lstm'] = [1 if mse_lstm > mse_threshold_lstm
        else 0 for mse_lstm in dataset_after_neural_network['MSE_lstm']]
```

```
148 # Metricas para Autoencoder
149 print_classification_metrics(dataset_after_neural_network['isAttack'],
150         dataset_after_neural_network['anomaly_autoencoder'])
151
152 # Metricas para LSTM
153 print_classification_metrics(dataset_after_neural_network['isAttack'],
154         dataset_after_neural_network['anomaly_lstm'])
155
156 # Gera figura 5.6
157 plot_recall_precision(dataset_after_neural_network, 'autoencoder')
158
159 # Gera figura 5.9
160 plot_recall_precision(dataset_after_neural_network, 'lstm')
```