UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ
CAMPUS DE FOZ DO IGUAÇU
PROGRAMA DE PÓS-GRADUAÇÃO EM
ENGENHARIA ELÉTRICA E COMPUTAÇÃO

DISSERTAÇÃO DE MESTRADO

# ABORDAGENS PARA O PROBLEMA DO DESBALANCEAMENTO EM DETECÇÃO DE INTRUSÃO - UM ESTUDO DE CASO APLICANDO CIC-IDS2018

CRISTIANO LUIZ STRESSER DA SILVA

FOZ DO IGUAÇU
2024

Cristiano Luiz Stresser da Silva

# Abordagens para o Problema do Desbalanceamento em Detecção de Intrusão - Um Estudo de Caso Aplicando CIC-IDS2018

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Computação da Universidade Estadual do Paraná como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica e Computação. Área de concentração: Sistemas Elétricos e Computação.

Orientador: Renato Bobsin Machado

Foz do Iguaçu
2024

Ficha de identificação da obra elaborada através do Formulário de Geração Automática do Sistema de Bibliotecas da Unioeste.

Silva, Cristiano Luiz Stresser da
   Abordagens para o problema do desbalanceamento em detecção de intrusão - Um estudo de caso aplicando CIC-IDS2018 / Cristiano Luiz Stresser da Silva; orientador Renato Bobsin Machado. -- Foz do Iguaçu, 2024.
   94 p.

   Dissertação (Mestrado Acadêmico  Campus de Foz do Iguaçu) -- Universidade Estadual do Oeste do Paraná, Centro de Engenharias e Ciências Exatas, Programa de Pós-Graduação em Engenharia Elétrica e Computação, 2024.

   1. Detecção de Intrusão. 2. Aprendizado de Máquina. I. Machado, Renato Bobsin, orient. II. Título.

# Abordagens para o Problema do Desbalanceamento em Detecção de Intrusão - Um Estudo de Caso Aplicando CIC-IDS2018

Cristiano Luiz Stresser da Silva

Esta Dissertação de Mestrado foi apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Computação e aprovada pela Banca Examinadora assim constituída:

Prof. Dr. **Renato Bobsin Machado** - (Orientador)
Universidade Estadual do Oeste do Paraná - UNIOESTE

Prof. Dr. **Edgar Manuel Carreño Franco**
Universidade Estadual do Oeste do Paraná - UNIOESTE

Prof. Dr. **Thiago França Naves**
Universidade Federal de Goiás - UFG

Data da defesa: 28 de Março de 2024.

# Resumo

O crescimento exponencial das tecnologias digitais e da Internet tem sido acompanhado por um aumento alarmante nos crimes virtuais. Este cenário tem motivado a intensificação de investimentos direcionados à segurança cibernética. Além disso, estudos acerca do tema também seguem em constante evolução. Dentro deste contexto, o presente trabalho consiste em um método de detecção de intrusão que aborda os problemas inerentes ao desbalanceamento presente no conjunto de dados CIC-IDS2018, por meio de técnicas de pré-processamento e treinamento do modelo. O método aborda o uso combinado de técnicas de *undersampling*, *oversampling* e pesos para treinamento sensível a custo. Com a abordagem adotada para endereçar o desbalanceamento, foi possível proporcionar uma melhoria na média aritmética das AUC do modelo de 92,0% para 98,2%. Além disso, a classe minoritária WebAttack demonstrou um aumento de AUC de 56,2% para 99,6%. Por fim, a acurácia média obtida foi de 95,4%, aproximando-se dos resultados de trabalhos relacionados. Os experimentos conduzidos demonstram que a abordagem proposta pode melhorar a capacidade de detecção e identificação de ameaças, especialmente em classes minoritárias, sem comprometer significativamente o desempenho geral.

**Palavras-chave**: Detecção de Intrusão, CIC-IDS2018, LightGBM, Desbalanceamento.

# Abstract

The exponential growth of digital technologies and the Internet has been accompanied by an alarming increase in cybercrimes. This scenario has motivated the intensification of investments in cybersecurity. Furthermore, studies on the topic are also constantly evolving. Within this context, this work consists of an intrusion detection method that addresses the problems associated with the imbalance present in the CIC-IDS2018 dataset, through pre-processing and model training techniques. The method addresses the combined use of *undersampling* and *oversampling* techniques along with weights for cost-sensitive training. With the approach used to address the imbalance, it was possible to provide an improvement in the macro average of the models' AUC from 92.0% to 98.2%. Additionally, the WebAttack minority class demonstrated an AUC increase from 56.2% to 99.6%. Finally, the mean accuracy obtained was 95.4%, approaching the results of related works. The experiments conducted show that the proposed approach can improve performance on intrusion detection and identification, especially in minority classes, without significantly compromising the overall performance.

**Keywords**: Intrusion Detection, CIC-IDS2018, LightGBM, Imbalance.

# Sumário

**Lista de Figuras** **9**

**Lista de Tabelas** **10**

**1 Introdução** **14**

1.1 Objetivo . . . . . 17

1.2 Proposta . . . . . 17

1.3 Estrutura do Trabalho . . . . . 18

**2 Fundamentação Teórica** **20**

2.1 Detecção de Intrusão . . . . . 20

2.2 Conjuntos de dados . . . . . 22

2.3 Tipos de Ataques . . . . . 27

2.3.1 Força-bruta . . . . . 27

2.3.2 Heartbleed . . . . . 27

2.3.3 Botnet . . . . . 28

2.3.4 Negação de Serviço . . . . . 28

2.3.5 Negação de Serviço Distribuída . . . . . 29

2.3.6 Ataques da Web . . . . . 29

2.3.7 Infiltração . . . . . 29

2.4 Inteligência Artificial . . . . . 30

2.4.1 Aprendizado de Máquina . . . . . 31

2.4.2 Mineração de Dados . . . . . 32

2.4.3 Árvores de Decisão . . . . . 33

2.4.4 *Ensemble Learning* . . . . . 35

2.4.5 *Random Forest* . . . . . 36

2.4.6 AdaBoost . . . . . 36

2.4.7 *Gradient Boosting* . . . . . 37

2.4.8 LightGBM . . . . . 38

2.5 Preparação dos dados . . . . . 39

2.5.1 Limpeza de dados . . . 39
2.5.2 Transformação de dados . . . 39
2.6 Avaliação de modelos . . . 41
2.6.1 *Holdout* . . . 41
2.6.2 Cross-validation . . . 42
2.6.3 Métricas . . . 42

**3 Trabalhos Relacionados** **47**
3.1 Histórico . . . 47
3.2 Estado da Arte . . . 49

**4 Materiais e Métodos** **53**
4.1 Local de Experimentação . . . 53
4.2 Materiais . . . 53
4.3 Método . . . 54
4.4 Pré-processamento . . . 54
4.4.1 Coleta dos Dados . . . 55
4.4.2 Limpeza dos Dados . . . 56
4.4.3 Transformação dos Dados . . . 58
4.5 Processamento . . . 62
4.6 Pós-processamento . . . 65

**5 Resultados e Discussão** **67**
5.1 Pré-processamento . . . 67
5.1.1 Coleta dos dados . . . 67
5.1.2 Limpeza dos dados . . . 67
5.1.3 Transformação dos dados . . . 68
5.1.4 Construção do modelo base . . . 70
5.1.5 Experimentos para definição das razões SMOTE . . . 71
5.2 Processamento . . . 74
5.2.1 Experimentos para definição dos pesos das classes . . . 75
5.2.2 Experimentos para ajuste de hiper-parâmetros . . . 77
5.3 Comparação . . . 80

**6 Conclusão** **83**

**Referências Bibliográficas** **85**

**A Apêndices** **92**

A.1 Erro com utilização de GOSS e CUDA . . . . . 92

# Lista de Figuras


Figura 1.1: Custos de crimes cibernéticos por ano. Fonte: Adaptado de Morgan & Braue (2022) . . . . . 15

Figura 2.1: Organização Generalizada de um IDS. Fonte: Adaptado de Wu & Banzhaf (2010) . . . . . 21

Figura 2.2: Classificação de um IDS. Fonte: Adaptado de Wu & Banzhaf (2010) . . . 22

Figura 2.3: Topologia de rede utilizada na construção do *dataset* CIC-IDS2018. Fonte: University of New Brunswick (2018) . . . . . 24

Figura 2.4: Árvore de decisão classificando clientes de alto e baixo risco. Fonte: Adaptado de Alpaydin (2021) . . . . . 34

Figura 2.5: 5-fold cross-validation. Fonte: Autor . . . . . 43

Figura 2.6: Gráfico de características operacionais do receptor (ROC). Fonte: Adaptado de Witten, Frank, Hall & Pal (2016) . . . . . 45

Figura 4.1: Método experimental da proposta. Fonte: Autor . . . . . 55

Figura 4.2: Otimização das razões SMOTE. Fonte: Autor . . . . . 61

Figura 4.3: Criação do modelo base. Fonte: Autor . . . . . 62

Figura 4.4: Processamento. Fonte: Autor . . . . . 63

Figura 5.1: Seleção de atributos. Fonte: Autor . . . . . 69

Figura 5.2: Evolução da otimização de razões SMOTE. Fonte: Autor . . . . . 72

Figura 5.3: AUC média em função das razões SMOTE. Fonte: Autor . . . . . 73

Figura 5.4: Evolução da otimização dos pesos. Fonte: Autor . . . . . 76

Figura 5.5: AUC média em função dos pesos. Fonte: Autor . . . . . 76

Figura 5.6: Evolução da otimização de hiper-parâmetros. Fonte: Autor . . . . . 79

Figura 5.7: AUC média em função dos hiper-parâmetros. Fonte: Autor . . . . . 79

# Lista de Tabelas


Tabela 2.1: Atributos do *dataset* CIC-IDS2018. . . . . 24
Tabela 2.2: Matriz de confusão. . . . . 43
Tabela 3.1: Trabalhos relacionados. . . . . 52
Tabela 4.1: Arquivos que compõem o *dataset* CIC-IDS2018. . . . . 56
Tabela 4.2: Tipagem definida para a manipulação do *dataset* CIC-IDS2018 com Pandas. 57
Tabela 4.3: Atributos com valor constante no *dataset* CIC-IDS2018. . . . . 57
Tabela 4.4: Exemplos por classe do *dataset* CIC-IDS2018. . . . . 58
Tabela 4.5: Agrupamento das classes do *dataset* CIC-IDS2018. . . . . 59
Tabela 4.6: Atributos selecionados. . . . . 60
Tabela 4.7: Representatividade das classes após *undersampling*. . . . . 60
Tabela 4.8: Definição do espaço de busca para otimização das razões SMOTE. . . . . 62
Tabela 4.9: Definição dos parâmetros estáticos do LightGBM. . . . . 63
Tabela 4.10: Definição do espaço de busca para otimização dos pesos. . . . . 64
Tabela 4.11: Definição do espaço de busca para ajuste dos hiper-parâmetros. . . . . 65
Tabela 5.1: AUCs do modelo base. . . . . 70
Tabela 5.2: AUCs com a aplicação de SMOTE. . . . . 72
Tabela 5.3: Testes estatísticos entre o modelo base e com SMOTE. . . . . 73
Tabela 5.4: AUC com o ajuste dos pesos das classes. . . . . 75
Tabela 5.5: Testes estatísticos entre o modelo com SMOTE e modelo com SMOTE e pesos. . . . . 77
Tabela 5.6: AUCs com o ajuste de hiper-parâmetros. . . . . 78
Tabela 5.7: Testes estatísticos entre AUCs médias do modelo com SMOTE e pesos e modelo com ajuste de hiper-parâmetros. . . . . 80
Tabela 5.8: Comparação dos resultados de trabalhos relacionados. . . . . 81

# Lista de Símbolos

| | |
|---|---|
| $\bar{x}$ | Média |
| $\sigma$ | Desvio Padrão |
| $H_0$ | Hipótese nula |
| $H_1$ | Hipótese alternativa |

# Lista de Siglas e Abreviaturas

| | |
|---|---|
| AM | Aprendizado de Máquina |
| AUC | *Area Under Receiver Operating Characteristics* |
| CPU | *Central Processing Unit* |
| DARPA | Defense Advanced Research Projects Agency |
| DDoS | *Distributed Denial of Service* |
| DNN | *Deep Feed-Forward Neural Network* |
| DoS | *Denial of Service* |
| DT | *Decision Tree* |
| DVWA | *Damn Vulnerable Web App* |
| EC2 | *Elastic Compute Cloud* |
| EFB | Exclusive Feature Bundling |
| FN | Falsos negativos |
| FP | Falsos positivos |
| FTP | *File Transfer Protocol* |
| GBDT | *Gradient Boosting Decision Tree* |
| GBT | *Gradient Boosting Tree* |
| GOSS | *Gradient-based One-side Sampling* |
| GPU | *Graphics Processing Unit* |
| HOIC | High Orbit Ion Cannon |
| IA | Inteligência Artificial |
| ICMP | *Internet Control Message Protocol* |
| IDS | *Intrusion Detection Systems* |
| IP | *Internet Protocol* |
| KDD | *Knowledge Discovery on Databases* |
| KNN | *K-Nearest Neighbors* |
| LOIC | Low Orbit Ion Cannon |
| LSTM | *Long Short Term Memory* |
| MCC | *Matthews Correlation Coefficient* |
| MD | Mineração de Dados |
| MIT | Massachussets Institute of Technology |

| | |
|---|---|
| ML | *Machine Learning* |
| MLP | *Multilayer Perceptron* |
| PCA | *Principal Component Analysis* |
| PGEEC | Programa de Pós-Graduação em Engenharia Elétrica e Computação, UNIOESTE, Campus de Foz do Iguaçu |
| R2L | *Root to Local* |
| RF | *Random Forest* |
| ROC | *Receiver Operating Characteristics* |
| S3 | *Simple Storage Service* |
| SMOTE | *Synthetic Minority Over-sampling Technique* |
| SQL | *Structured Query Language* |
| SSH | *Secure Shell* |
| SVM | *Support Vector Machine* |
| TCP | *Transmission Control Protocol* |
| TFN | Taxa de Falsos Negativos |
| TFP | Taxa de Falsos Positivos |
| TLS | *Transport Layer Security* |
| TPE | *Tree-structured Parzen Estimator* |
| U2R | *User to Root* |
| UDP | *User Datagram Protocol* |
| UNIOESTE | Universidade Estadual do Oeste do Paraná |
| URL | *Uniform Resource Locator* |
| UvO | Um-versus-outros |
| VP | Verdadeiros positivos |
| VN | Verdadeiros negativos |
| XSS | *Cross-Site Scripting* |

# Capítulo 1

# Introdução

O avanço das aplicações tecnológicas, a digitalização e o uso da Internet estão em constante crescimento, acompanhados pelo expansão da computação em nuvem, disponibilidade de serviços online e dispositivos conectados à rede. O aumento do uso desses serviços foi alavancado pela pandemia, com 60% da população mundial utilizando a Internet em 2020, (THE WORLD BANK, 2022). Este crescimento corresponde a 11% em relação ao ano anterior.

Paralelamente a esta constatação, os crimes cibernéticos também seguem em aceleração (Saxena, Sinha & Shukla, 2017), com a estimativa de provocar 9,5 trilhões de dólares em danos mundialmente durante 2024. A previsão é de que esses custos cresçam 15% ao ano até 2025, atingindo $10,5 trilhões, contra $3 trilhões em 2015 (Morgan & Osborne, 2023), conforme apresentados na Figura 1.1. Entre as principais atividades praticadas por usuários mal-intencionados, estão ataques de negação de serviço (DOS), força bruta, infiltração e manipulação, tendo como objetivo comprometer o funcionamento de serviços, obter ou alterar informações confidenciais, entre outros (Sohi, Seifert & Ganji, 2021; Crosbie & Spafford, 1995; Heady, Luger, Maccabe & Servilla, 1990). A essas práticas, é empregado o termo guarda-chuva intrusão.

Os alarmantes prejuízos causados por estas atividades causam preocupação às empresas, levando à procura por recursos para seu controle. Em uma pesquisa conduzida por Bissell, Fox, LaSalle & Dal Cin (2021), 82% das empresas participantes relataram que houve aumento no orçamento destinado a segurança cibernética, em relação ao ano anterior. Além disso, a quantidade global de postos de trabalho em aberto para segurança cibernética cresceu 350% entre 2013 e 2021, de 1 a 3,5 milhões, mantendo essa mesma quantidade de vagas em aberto para 2023 (Morgan & Osborne, 2023).

A fim de reduzir os danos, existem diversos recursos voltados à segurança computacional, visando mitigar riscos, desenvolvidos tanto pela indústria, quanto pela comunidade científica. Administradores de segurança empregam tradicionalmente mecanismos de proteção por senha, técnicas de criptografia, controle de acesso e *firewalls* para proteção das redes (Karatas, Demir & Sahingoz, 2020). Apesar de contribuírem, este conjunto de técnicas não são suficientes para evitar comportamentos mal-intencionados. Em conjunto com os dispositivos anteriormente mencionados, os sistemas de detecção de intrusão, ou IDS (*Intrusion Detection*

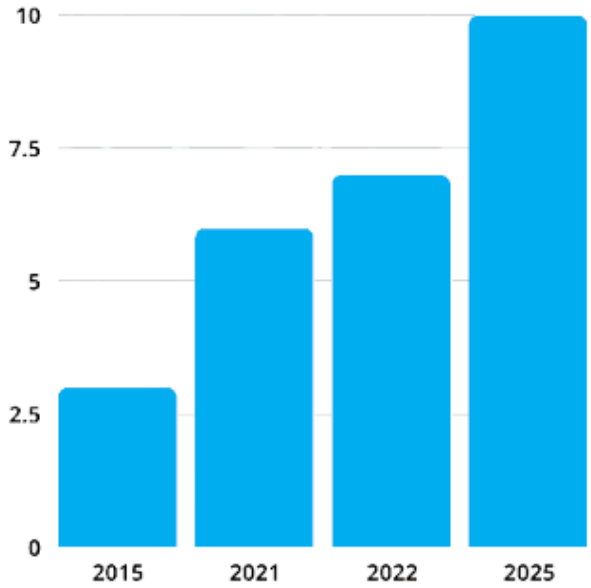


**Figura 1.1:** Custos de crimes cibernéticos por ano.
Fonte: Adaptado de Morgan & Braue (2022)

*Systems*), são utilizados em larga escala em infraestrutura de segurança de redes, tendo como objetivo identificar ataques maliciosos por meio do monitoramento do tráfego da rede. Logo após as primeiras iniciativas relacionadas à detecção de intrusão, como a proposta de Denning & Neumann (1985), baseada em sistemas especialistas, iniciaram-se as contribuições da academia para o aprimoramento dos IDS, que seguem até a atualidade, com a utilização de aprendizado de máquina (Spirakis, 1994; Pfahringer, 2000; Machado, 2005; Govindarajan & Chandrasekaran, 2011; Souza, 2018; D'Hooge, Wauters, Volckaert & De Turck, 2020; Karatas et al., 2020; Leevy, Hancock, Zuech & Khoshgoftaar, 2021).

Para possibilitar o desenvolvimento e a avaliação dos sistemas de detecção de intrusão, é fundamental que sejam estabelecidos conjuntos de dados padronizados (Sommer & Paxson, 2010). Com esse propósito, em 1998 o MIT (Massachussets Institute of Technology) criou o primeiro *dataset* com utilização ampla, o DARPA98, contendo capturas de pacotes de rede com tráfego típico e anômalo, caracterizando intrusões. Fomentado pelo crescimento no desenvolvimento de sistemas de detecção de intrusão e atenção à segurança cibernética, ao longo dos anos foram disponibilizados diversos novos *datasets*, atualizando e endereçando deficiências encontradas, como DARPA99, KDD Cup 99, e NSL-KDD (Lippmann, Haines, Fried, Korba & Das, 2000; Tavallaee, Bagheri, Lu & Ghorbani, 2009), além de propostas que acompanham uma abordagem sistemática para geração de conjuntos de dados, como ISCX 2012 e CIC-IDS (Shiravi, Shiravi, Tavallaee & Ghorbani, 2012; Sharafaldin, Lashkari & Ghorbani, 2018).

Os *datasets* mais recentes, que foram criados com a premissa de melhoria na qualidade dos dados, ainda possuem pontos de atenção, exigindo cuidados no pré-processamento e modelagem em aprendizado de máquina (Thakkar & Lohiya, 2020), assim como habitualmente ocorre nessas tarefas. Um IDS exemplar pode ser descrito como o que é capaz de detectar

cada tipo de ataque precisamente (Panigrahi & Borah, 2018). Deste modo, é importante que os classificadores tenham desempenho satisfatório para cada uma das classes, e não apenas uma acurácia global alta, em que a maior parte das instâncias pertençam a uma classe majoritária. Há um elevado número de estudos utilizando os *datasets* CIC-IDS, entretanto muitos deles não dão a devida importância a aspectos como desbalanceamento e utilização de métricas que permitam avaliar sua performance com os ataques de classes minoritárias, muitas vezes apresentando classificadores com acurácias quase perfeitas, porém sem explorar a possibilidade de ocorrência de viés e *overfitting* (Leevy & Khoshgoftaar, 2020; Kulkarni, Chong & Batarseh, 2020). Tendo em vista este cenário, é importante destacar que há uma série de abordagens com o objetivo de minimizar os efeitos do desbalanceamento, aplicadas tanto no pré-processamento, com tratamento de dados, quanto no treinamento dos algoritmos de aprendizado (Pears, Finlay & Connor, 2014; Amin, Anwar, Adnan, Nawaz, Howard, Qadir, Hawalah & Hussain, 2016; Guo, Zhou & Wu, 2020; Kaur & Gosain, 2018).

Fitni & Ramli (2020) comparam os resultados de vários algoritmos conhecidos de aprendizado de máquina aplicados ao *dataset* CIC-IDS-2018, com classificação multi-classe, sem a utilização de técnicas de balanceamento. Os resultados gerais dos modelos demonstram ótimas acurácias, porém as classes minoritárias possuem altas taxas de classificação incorreta.

No trabalho de Karatas et al. (2020), é apresentada uma comparação de IDS utilizando 6 (seis) algoritmos de aprendizado de máquina, com o *dataset* CIC-IDS-2018. Como o desbalanceamento é presente neste, os autores aplicaram a técnica de amostragem *Synthetic Minority Over-sampling Technique* (SMOTE) (Chawla, Bowyer, Hall & Kegelmeyer, 2002). Dentre as implementações, a que utilizou gradient boosting, com o algoritmo LightGBM (Ke, Meng, Finley, Wang, Chen, Ma, Ye & Liu, 2017), apresentou a melhor acurácia média após o balanceamento. Entretanto, o mesmo também demonstrou maior sensibilidade ao desbalanceamento, quando comparado ao Adaboost e árvores de decisão (Witten et al., 2016), apesar de ser menos sensível que algoritmos como KNN.

O desbalanceamento está relacionado à distribuição das classes em um conjunto de dados. Enquanto tecnicamente qualquer *dataset* que possua diferença entre a quantidade de instâncias de cada classe pode ser considerado desbalanceado, o entendimento na comunidade é de que taxas de desbalanceamento entre 100:1 e 10000:1 representam desbalanceamento significativo ou extremo (He & Garcia, 2009). O *dataset* utilizado neste trabalho possui algumas classes que se enquadram nessa condição, as quais serão apresentadas em detalhe posteriormente. O efeito da utilização de *datasets* com essas características, sem o devido tratamento, implica na construção de modelos que tendem a classificar com eficácia somente a classe majoritária, porém com baixa aptidão nas demais classes.

Tendo em vista este cenário, é importante destacar que há uma série de abordagens com o objetivo de minimizar os efeitos do desbalanceamento, aplicadas tanto ao tratamento de dados, quanto aos algoritmos de aprendizado. As técnicas relacionadas aos dados são aplicadas no pré-processamento, por meio de amostragem, buscando a criação de novos conjuntos de dados

balanceados (Pears et al., 2014; Amin et al., 2016; Guo et al., 2020; Kaur & Gosain, 2018). Já as abordagens relacionadas aos algoritmos são empregadas na fase de treinamento do modelo, como o aprendizado sensível a custo, onde os classificadores atribuem penalizações maiores a instâncias de classes minoritárias classificadas incorretamente (Elkan, 2001), ou métodos *ensemble*, combinados com algoritmos de *boosting*. Essas técnicas não são tão flexíveis quanto as relacionadas a dados, pois dependem dos algoritmos, restringindo sua aplicação a classificadores específicos (Tahir, Asghar, Manzoor & Noor, 2019).

Japkowicz & Stephen (2002) realizaram um estudo sistemático sobre o problema do aprendizado desbalanceado, apresentando alguns entendimentos acerca do desbalanceamento, relacionando a complexidade conceitual, tamanho de conjuntos de dados e níveis de desbalanceamento. São apresentadas estratégias para endereçar esse problema, entre as citadas anteriormente, com experimentação utilizando árvores de decisão. É avaliado se os problemas encontrados com desbalanceamento são exclusivos para esse algoritmo. Os resultados apresentam alguns pontos interessantes, como o de que em domínios separáveis linearmente aparentam não ser sensíveis ao desbalanceamento. Foram realizados experimentos com *Multilayer Perceptron* (MLP), que demonstrou menor sensibilidade a desbalanceamento do que as árvores de decisão, e o algoritmo *Support Vector Machine* (SVM), que apresentou resultados similares, independente do desbalanceamento. Por fim, os autores concluem que quanto maior o nível de desbalanceamento, maior a complexidade do conceito e menor o tamanho do conjunto de dados, maior o efeito provocado pelo desbalanceamento.

## 1.1 Objetivo

Considerando os problemas apresentados acerca da segurança computacional, bem como o desenvolvimento contínuo de soluções para detecção de intrusão, esse trabalho tem como objetivo propor uma abordagem para o problema do desbalanceamento em detecção de intrusão que melhore a performance da classificação, levando em consideração as classes minoritárias.

Estabelece-se como hipótese a ser investigada neste trabalho, que a aplicação de técnicas de balanceamento e treinamento sensível a custo para detecção de intrusão com LightGBM e CIC-IDS2018, proporciona melhorias na performance.

## 1.2 Proposta

Dentre os trabalhos analisados no levantamento bibliográfico, foram identificadas diversas técnicas empregadas nos métodos propostos, incluindo amostragem para redução no desbalanceamento dos conjuntos de dados e aplicação de função de perda sensível a custo na fase de

treinamento, as quais provocaram melhorias na performance dos modelos.

Foi observado que nas técnicas de balanceamento, quando aplicado *oversampling* com SMOTE, em que o número de instâncias de classes minoritárias é aumentado, é percebida uma melhoria na acurácia. Entretanto, o crescimento do conjunto de dados também faz com que o tempo de treinamento e utilização de recursos aumentem. Assim, neste trabalho é proposta a utilização do balanceamento combinando técnicas de *undersampling*, reduzindo o número de exemplos da classe majoritária, em conjunto com o *oversampling* de classes minoritárias, buscando obter um conjunto de dados mais balanceado, sem aumento da quantidade de dados a ser processada.

Outra técnica empregada em alguns trabalhos é a função de perda sensível a custo, que penaliza mais severamente erros de classificação em classes minoritárias. Deste modo, mesmo com um desbalanceamento mais expressivo, os modelos não apresentam variação de performance em classes minoritárias.

Com a utilização dessa técnica em conjunto com as de balanceamento, o método proposto nesse trabalho possibilita uma intervenção ainda menor no conjunto de dados com a replicação de exemplos, mantendo uma menor utilização de recursos e tempo para o treinamento do modelo.

A primeira etapa do método proposto é a aplicação de *undersampling* na classe majoritária do *dataset* CIC-IDS-2018 (Sharafaldin et al., 2018), com a razão de 22.25% encontrada por Hua (2020), em que não foi observada redução significativa na performance, apesar da expressiva redução no tamanho do *dataset*.

Na sequência, as classes minoritárias do conjunto de dados são submetidas a *oversampling* com o método SMOTE, com experimentação em diversas razões. Para cada combinação de razões, é efetuado o treinamento do modelo, são realizadas as predições e coletadas as métricas para avaliação.

Como a próxima fase do método, o *dataset* com as melhores razões SMOTE é utilizado para experimentação no treinamento do modelo, com função de perda sensível a custo, variando os pesos e obtendo as métricas para posterior avaliação.

Por fim, a melhor configuração encontrada para criação do modelo é utilizada para experimentos com ajuste de hiper-parâmetros.

## 1.3 Estrutura do Trabalho

No Capítulo 2 são apresentados conceitos acerca de segurança computacional, detecção de intrusão, aprendizado de máquina e avaliação de modelos. Na sequência, o Capítulo 3 fornece uma revisão da literatura disponível, visando estabelecer uma base sólida para iden-

tificação de lacunas para pesquisa e compreensão do estado da arte. O Capítulo 4 demonstra os materiais e o método experimental da abordagem proposta neste trabalho. Em seguida, a demonstração dos resultados obtidos e discussão acerca dos mesmos é realizada no Capítulo 5. Por fim, o Capítulo 6 apresenta as conclusões que puderam ser estabelecidas com os resultados e contribuições do presente trabalho.

# Capítulo 2

# Fundamentação Teórica

Neste capítulo serão apresentados conceitos e fundamentação de temas relevantes ao conteúdo do presente trabalho, como segurança computacional, detecção de intrusão e aprendizado de máquina.

Iniciamos com a seção 2.1 sobre detecção de intrusão, que explora os conceitos fundamentais dessa área crucial da segurança cibernética, delineando as técnicas e abordagens utilizadas para identificar atividades maliciosas em redes. Em seguida, na seção 2.2, discutimos a importância dos conjuntos de dados na pesquisa em detecção de intrusão, destacando conjuntos de dados padrão e sua relevância para o desenvolvimento e avaliação de modelos. A seção 2.3 explora os tipos de ataques compreendidos no *dataset* utilizado neste trabalho. Em seguida, na seção 2.4, nos concentramos em inteligência artificial, examinando suas definições e diversos métodos e técnicas para sua aplicação. A seção 2.5 trata da preparação de dados, destacando a importância da limpeza, transformação, seleção de atributos e amostragem para garantir a qualidade e relevância dos dados utilizados nos modelos de detecção. Por fim, na seção 2.6, discutimos a avaliação de modelos, abordando métricas de desempenho, métodos de validação e outras considerações cruciais para a avaliação eficaz dos modelos de detecção de intrusão desenvolvidos. Essas seções fornecem uma base sólida para o trabalho empírico e análise crítica que serão apresentados ao longo desta dissertação.

## 2.1 Detecção de Intrusão

Em conjunto com diversos dispositivos empregados na mitigação de riscos associados à segurança computacional, os sistemas de detecção de intrusão, ou IDS (*Intrusion Detection Systems*), são utilizados em larga escala em infraestrutura de segurança de redes, tendo como objetivo identificar atividades maliciosas por meio do monitoramento de atividades direcionadas a um sistema, como tráfego da rede, *logs* de aplicações e de sistemas operacionais (Pietro & Mancini, 2008).

Os IDS possuem arquitetura similar em diversos aspectos, contendo monitoramento, coleta de dados, detecção de intrusão e ações relacionadas às intrusões identificadas. Wu &

Banzhaf (2010) retratam uma generalização para a organização de tais sistemas, representado na Figura 2.1, com as linhas contínuas indicando fluxos de dados e controle, e as pontilhadas respostas a atividades intrusivas.

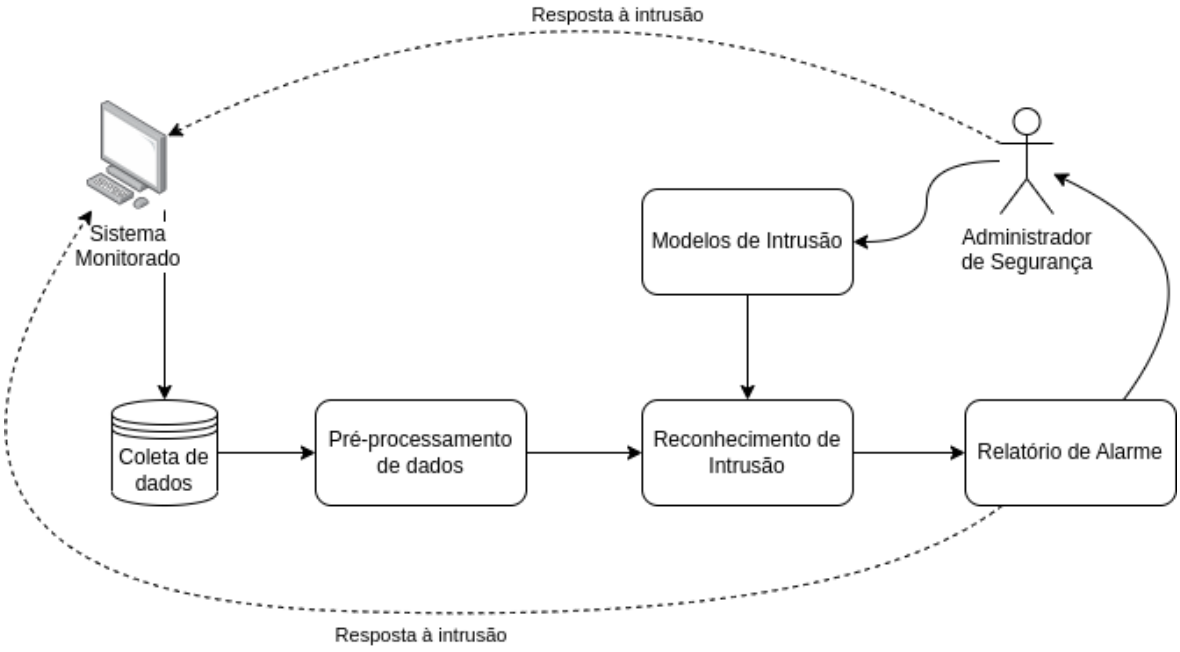


**Figura 2.1:** Organização Generalizada de um IDS.
Fonte: Adaptado de Wu & Banzhaf (2010)

Esses sistemas podem ser classificados em duas categorias, de acordo com o método de detecção: de uso indevido, que identifica as intrusões com base em descrições definidas de ataques conhecidos; e de anomalias, que elabora modelos para distinção de comportamento normal e ataques. O primeiro tem maior efetividade para intrusões já conhecidas, porém como há constante evolução e mutação dos ataques, pode tornar-se ineficiente. Em contrapartida, os baseados em anomalia tem maior eficácia para detecção de ataques novos, porém sua grande dificuldade é encontrar os limites para distinção entre comportamento normal e anormal, devido à deficiência de exemplos anormais para treinamento (Wu & Banzhaf, 2010).

Classificações adicionais foram propostas pelos autores, apresentadas na Figura 2.2, como as relacionadas ao tipo de resposta à intrusão, que pode ser passiva ou ativa. Em relação à fonte dos dados, existem duas origens possíveis: *Hosts*, ou as máquinas que hospedam as aplicações e serviços, que podem fornecer dados de *logs* do sistema ou de aplicações para análise do IDS; e redes, em que são obtidos metadados associados a capturas de tráfego de rede. Quanto ao local de detecção, é possível que seja centralizado, bem como distribuído.

As primeiras iniciativas relacionadas à detecção de intrusão, como a proposta de Denning & Neumann (1985), utilizavam abordagens baseadas em combinações de sistemas especialistas e estatística. Os modelos de detecção também dependiam de conhecimento de especialistas em segurança. Na sequência, a partir da metade dos anos 1990, houve uma evolução na fase de coleta dos dados, passando a ser automatizada. Além disso, Inteligência Artificial (IA) e apren-

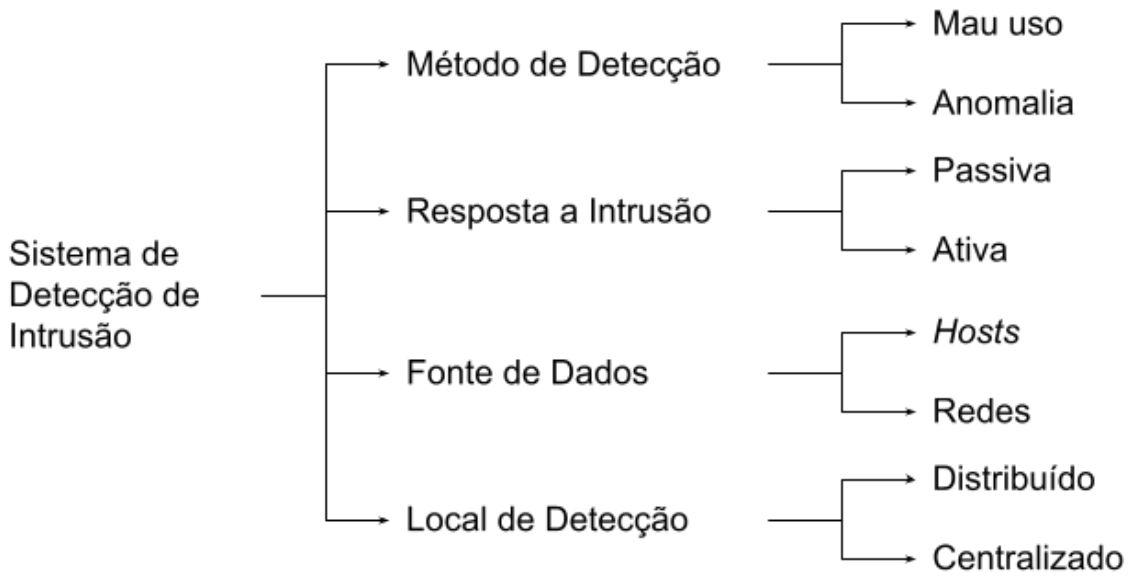


**Figura 2.2:** Classificação de um IDS.
Fonte: Adaptado de Wu & Banzhaf (2010)

dizado de máquina passaram a ser empregados para a construção de modelos a partir de dados de treinamento. Desde o princípio, houveram contribuições da academia para o aprimoramento dos IDS, que seguem até a atualidade (Spirakis, 1994; Pfahringer, 2000; Machado, 2005; Govindarajan & Chandrasekaran, 2011; Souza, 2018; D'Hooge et al., 2020; Karatas et al., 2020; Leevy et al., 2021).

## 2.2 Conjuntos de dados

Um requisito fundamental para o desenvolvimento de IDS, é a disponibilidade de *datasets*, ou conjuntos de dados. Ao final dos anos 1990, um problema que se colocava era a falta de padronização para avaliação de IDS. Diversas comparações não apresentavam taxas de alarme falso, o que quando significativas, podem ocasionar um custo elevado na manutenção desses sistemas, devido à necessidade de especialistas para análise. Para possibilitar a obtenção dessas taxas, se faz necessária a disponibilidade de instâncias que caracterizem comportamento de uso normal, ou tráfego de *background*. Além disso, as avaliações de performance não utilizavam os mesmos conjuntos de dados, contendo ataques, topologias de rede, sistemas operacionais e serviços distintos. Ao longo dessa seção, serão apresentados alguns entre os principais conjuntos de dados para detecção de intrusão.

Para o estabelecimento de uma coletânea de conjuntos de dados padronizados, permitindo a avaliação e comparação de IDSs, em 1998, o MIT, patrocinado pela DARPA e o Laboratório de Pesquisa da Força Aérea, criou o *dataset* DARPA98. O mesmo contém capturas de tráfego de rede *tcpdump* em máquinas UNIX. No ano subsequente, com a evolução dos ataques, um

novo *dataset* foi disponibilizado, com novos ataques e a inclusão de máquinas com sistemas Microsoft NT, o qual foi denominado DARPA99 (Lippmann et al., 2000). Em seguida, o *dataset* KDD Cup 99 foi criado, a partir de uma transformação, com base nas capturas de tráfego do DARPA99, contendo 41 atributos considerados adequados à classificação por algoritmos de aprendizado de máquina. Este é o conjunto de dados mais utilizado para a avaliação de IDS, apesar de ser extensivamente criticado e considerado inadequado por diversos especialistas (Siddique, Akhtar, Aslam Khan & Kim, 2019; McHugh, 2000; Mahoney & Chan, 2003).

Com o intuito de solucionar alguns problemas do KDD99, Tavallaee et al. (2009) propuseram uma derivação do *dataset*, chamada NSL-KDD. As melhorias incluíram a eliminação da redundância de exemplos, que provocava viés nos modelos treinados. Outra deficiência endereçada foi o tamanho do *dataset*. Segundo os autores, devido à grande quantidade de exemplos, muitos modelos criados com base no KDD99 utilizavam subconjuntos aleatórios para treinamento, dificultando a comparação entre diferentes publicações. Assim, o NSL-KDD conta com um número razoável de exemplos para treino e teste, permitindo a utilização do *dataset* completo.

Apesar das diversas melhorias proporcionadas pelo NSL-KDD, que provocaram um aumento de sua utilização em substituição ao KDD99, ainda existem diversos problemas não endereçados, como sua acurácia, habilidade de representar condições reais, além da obsolência e impossibilidade de reprodução, extensão e modificação. Segundo Sommer & Paxson (2010), o desafio mais significativo encontrado para avaliação de IDS é a falta de *datasets* públicos apropriados. Com o propósito de tratar essas limitações, foi concebida uma abordagem sistemática para a geração de *datasets*, facilitando a análise, testes e avaliação de IDS, com foco em detecção por anomalia (Shiravi et al., 2012). Esta iniciativa disponibilizou o *dataset* ISCX2012, e possibilitou versões com melhorias e atualizações seguindo abordagens similares, o CIC-IDS-2017 e CSE-CIC-IDS2018, utilizado nesse trabalho (Sharafaldin et al., 2018).

A infraestrutura criada para o ambiente de testes do CSE-CIC-IDS2018, que será referenciado intercambiavelmente como CIC-IDS2018 neste trabalho, é composta por duas redes separadas, a atacante e a vítima. Ao contrário das abordagens anteriores, esta compreende diversos dispositivos comuns e equipamentos necessários, como roteadores, *firewalls*, *switches*, além dos sistemas operacionais utilizados mais amplamente, Windows, Linux e Macintosh. A figura 2.3 apresenta parte da topologia utilizada para construção da base CIC-IDS2018. À esquerda, em amarelo, é demonstrada a rede vítima, aproximando-se de uma rede corporativa típica, com diversos departamentos e uma rede de servidores internos. Em vermelho, é apresentada a rede dos atacantes. Também dando atenção a deficiências já identificadas há décadas, esse trabalho priorizou a geração de tráfego de *background* realístico.

O CIC-IDS2018 compreende 80 colunas, que consistem na classe, porta de destino, protocolo, endereço IP e marca de tempo do fluxo, além de outros 75 atributos estatísticos, apresentados na tabela 2.1, obtidos com a ferramenta CICFlowMeter V3 [1], que os computa a partir

[1] https://www.unb.ca/cic/research/applications.html#CICFlowMeter

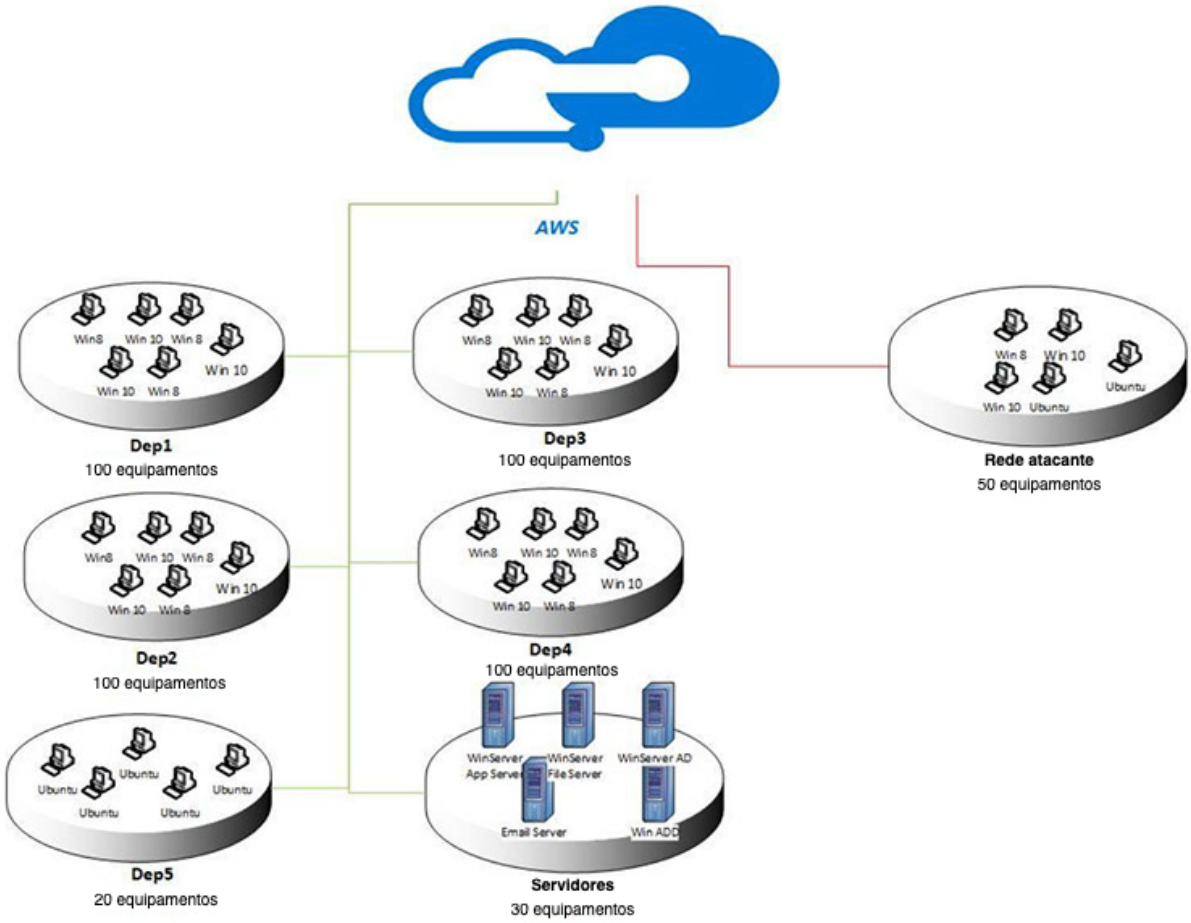


**Figura 2.3:** Topologia de rede utilizada na construção do *dataset* CIC-IDS2018.
Fonte: University of New Brunswick (2018)

da geração de fluxos bidirecionais, em que o primeiro pacote determina o encaminhamento (origem para destino, ou *forward*), além da direção reversa (destino para origem, ou *backward*). Esses atributos são calculados em ambas direções. Normalmente fluxos do protocolo TCP são terminados por um pacote FIN, enquanto fluxos UDP finalizam de acordo com um tempo limite do fluxo. Esse tempo limite, pode ser especificado independentemente do protocolo.

**Tabela 2.1:** Atributos do *dataset* CIC-IDS2018.

| *Atributo* | *Descrição* |
|---|---|
| Dst Port | Porta de Destino |
| Protocol | Protocolo |
| Timestamp | Marca de tempo do início do fluxo |
| Flow Duration | Duração do fluxo |
| Tot Fwd Pkts | Total de pacotes na direção *forward* |
| Tot Bwd Pkts | Total de pacotes na direção *backward* |
| TotLen Fwd Pkts | Tamanho total dos pacotes na direção *forward* |
| TotLen Bwd Pkts | Tamanho total dos pacotes na direção *backward* |
| Fwd Pkt Len Max | Maior pacote na direção *forward* |

| | |
|---|---|
| Fwd Pkt Len Min | Menor pacote na direção *forward* |
| Fwd Pkt Len Mean | Tamanho médio do pacote na direção *forward* |
| Fwd Pkt Len Std | Desvio padrão do tamanho dos pacotes na direção *forward* |
| Bwd Pkt Len Max | Maior pacote na direção *backward* |
| Bwd Pkt Len Min | Menor pacote na direção *backward* |
| Bwd Pkt Len Mean | Média do tamanho dos pacotes na direção *backward* |
| Bwd Pkt Len Std | Desvio padrão do tamanho dos pacotes na direção *backward* |
| Flow Byts/s | Taxa de bytes transferidos por segundo no fluxo |
| Flow Pkts/s | Taxa de pacotes transferidos por segundo no fluxo |
| Flow IAT Mean | Tempo médio entre dois fluxos |
| Flow IAT Std | Desvio padrão do tempo entre dois fluxos |
| Flow IAT Max | Tempo máximo entre dois fluxos |
| Flow IAT Min | Tempo mínimo entre dois fluxos |
| Fwd IAT Tot | Tempo total entre dois pacotes enviados na direção *forward* |
| Fwd IAT Mean | Tempo médio entre dois pacotes enviados na direção *forward* |
| Fwd IAT Std | Desvio padrão do tempo entre dois pacotes enviados na direção *forward* |
| Fwd IAT Max | Tempo máximo entre dois pacotes enviados na direção *forward* |
| Fwd IAT Min | Tempo mínimo entre dois pacotes enviados na direção *forward* |
| Bwd IAT Tot | Tempo total entre dois pacotes enviados na direção *backward* |
| Bwd IAT Mean | Tempo médio entre dois pacotes enviados na direção *backward* |
| Bwd IAT Std | Desvio padrão do tempo entre dois pacotes enviados na direção *backward* |
| Bwd IAT Max | Tempo máximo entre dois pacotes enviados na direção *backward* |
| Bwd IAT Min | Tempo mínimo entre dois pacotes enviados na direção *backward* |
| Fwd PSH Flags | Contagem de flags PSH em pacotes na direção *forward* (0 para UDP) |
| Bwd PSH Flags | Contagem de flags PSH em pacotes na direção *backward* (0 para UDP) |
| Fwd URG Flags | Contagem de flags URG em pacotes na direção *forward* (0 para UDP) |
| Bwd URG Flags | Contagem de flags URG em pacotes na direção *backward* (0 para UDP) |
| Fwd Header Len | Bytes totais usados para cabeçalhos na direção *forward* |
| Bwd Header Len | Bytes totais usados para cabeçalhos na direção *backward* |
| Fwd Pkts/s | Quantidade de pacotes na direção *forward* por segundo |
| Bwd Pkts/s | Quantidade de pacotes na direção *backward* por segundo |
| Pkt Len Min | Comprimento mínimo de um fluxo |
| Pkt Len Max | Comprimento máximo de um fluxo |
| Pkt Len Mean | Comprimento médio de um fluxo |
| Pkt Len Std | Desvio padrão do comprimento de um fluxo |
| Pkt Len Var | Tempo mínimo entre chegadas de pacotes |
| FIN Flag Cnt | Quantidade de pacotes com a flag FIN |
| SYN Flag Cnt | Quantidade de pacotes com a flag SYN |
| RST Flag Cnt | Quantidade de pacotes com a flag RST |
| PSH Flag Cnt | Quantidade de pacotes com a flag PUSH |

| | |
|---|---|
| ACK Flag Cnt | Quantidade de pacotes com a flag ACK |
| URG Flag Cnt | Quantidade de pacotes com a flag URG |
| CWE Flag Count | Quantidade de pacotes com a flag CWE |
| ECE Flag Cnt | Quantidade de pacotes com a flag ECE |
| Down/Up Ratio | Razão entre download e upload |
| Pkt Size Avg | Tamanho médio do pacote |
| Fwd Seg Size Avg | Tamanho médio observado na direção *forward* |
| Bwd Seg Size Avg | Tamanho médio observado na direção *backward* |
| Fwd Byts/b Avg | Média de bytes de transmissão em massa na direção *forward* |
| Fwd Pkts/b Avg | Média de pacotes de transmissão em massa na direção *forward* |
| Fwd Blk Rate Avg | Média da taxa de transmissão em massa na direção *forward* |
| Bwd Byts/b Avg | Média de bytes de transmissão em massa na direção *backward* |
| Bwd Pkts/b Avg | Média de pacotes de transmissão em massa na direção *backward* |
| Bwd Blk Rate Avg | Média da taxa de transmissão em massa na direção *backward* |
| Subflow Fwd Pkts | Média de pacotes em um sub-fluxo na direção *forward* |
| Subflow Fwd Byts | Média de bytes em um sub-fluxo na direção *forward* |
| Subflow Bwd Pkts | Média de pacotes em um sub-fluxo na direção *backward* |
| Subflow Bwd Byts | Média de bytes em um sub-fluxo na direção *backward* |
| Init Fwd Win Byts | Bytes enviados na janela inicial na direção *forward* |
| Init Bwd Win Byts | Bytes enviados na janela inicial na direção *backward* |
| Fwd Act Data Pkts | Pacotes com pelo menos 1 byte de carga de dados TCP na direção *forward* |
| Fwd Seg Size Min | Tamanho mínimo do segmento observado na direção *forward* |
| Active Mean | Tempo médio que um fluxo esteve ativo antes de ficar inativo |
| Active Std | Desvio padrão do tempo que um fluxo esteve ativo antes de ficar inativo |
| Active Max | Tempo máximo que um fluxo esteve ativo antes de ficar inativo |
| Active Min | Tempo mínimo que um fluxo esteve ativo antes de ficar inativo |
| Idle Mean | Tempo médio que um fluxo esteve inativo antes de ficar ativo |
| Idle Std | Desvio padrão do tempo que um fluxo esteve inativo antes de ficar ativo |
| Idle Max | Tempo máximo que um fluxo esteve inativo antes de ficar ativo |
| Idle Min | Tempo mínimo que um fluxo esteve inativo antes de ficar ativo |
| Label | Rótulo que classifica a instância |

Fonte: University of New Brunswick (2018)

O *dataset* CIC-IDS2018, compreende uma série de ataques comumente utilizados por usuários mal-intencionados. Na seção a seguir, os mesmos são apresentados, juntamente a detalhes sobre o funcionamento.

## 2.3 Tipos de Ataques

### 2.3.1 Força-bruta

Ataques de força-bruta são o tipo de ataque mais popular, podendo ser aplicados para quebra de senhas e descoberta de conteúdo oculto em aplicações da Internet. Seu princípio de funcionamento é simples, e se dá por tentativas sucessivas até alcançar o objetivo. Existe uma série de ferramentas para condução desse tipo de ataque, como Hydra[2], Medusa[3], Ncrack[4], módulos do Metasploit[5], entre outros.

Para a construção do *dataset* CIC-IDS2018, utilizado neste trabalho, os ataques desta categoria foram conduzidos com a ferramenta Patator[6], por ser a mais compreensiva, além de mais confiável e flexível que as demais. Foram utilizados os módulos FTP e SSH, juntamente a um dicionário de 90 milhões de palavras.

### 2.3.2 Heartbleed

Uma família de ataques com grande popularidade, são os baseados em vulnerabilidades de *software* (Igure & Williams, 2008). Eventualmente as mesmas são reconhecidas e atualizações para correção são disponibilizadas. Entretanto, algumas afetam *softwares* sendo executados em milhões de servidores ou vítimas, levando um longo período de tempo para aplicação em todas os computadores. Assim, durante esse período, estes permanecem vulneráveis aos ataques.

Um dos casos que ganhou destaque nos últimos anos, é o ataque Heartbleed (Kyatam, Alhayajneh & Hayajneh, 2017). O mesmo surge de uma falha na biblioteca de criptografia OpenSSL, utilizada amplamente em implementações do protocolo de segurança TLS. Seu funcionamento é baseado no envio de uma solicitação de *heartbeat*, que tem como objetivo verificar se o computador remoto está *online*. Entretanto, no ataque é enviada uma mensagem maliciosa, que pode permitir que a vítima aceite-a, e transmita dados sigilosos, como dados da memória do servidor. Isso possibilita a obtenção da chave privada da vítima, a qual é utilizada para criptografar os dados que trafegam a partir dessa máquina, assim, permitindo a leitura de dados sigilosos.

No CIC-IDS2018, foi empregada a ferramenta Heartleech[7] para a busca por sistemas vulneráveis a esse *bug*. A mesma também possibilita a execução do ataque.

[2]https://github.com/vanhauser-thc/thc-hydra
[3]https://github.com/jmk-foofus/medusa
[4]https://nmap.org/ncrack/man.html
[5]https://www.metasploit.com/
[6]https://github.com/lanjelot/patator
[7]https://github.com/robertdavidgraham/heartleech

### 2.3.3 Botnet

Botnet é um conjunto de dispositivos conectados à Internet, que são utilizados pelo proprietário para realizar diversas tarefas maliciosas. Esse tipo de ataque pode ser usado para roubo de dados, envio de *spam*, além de permitir acesso ao dispositivo e sua conexão. Uma das principais finalidades é potencializar a realização de tarefas maliciosas, devido ao aumento de recursos, com os diversos dispositivos controlados pelo atacante, denominados zumbis (Feily, Shahrestani & Ramadass, 2009).

No *dataset* deste trabalho, foram utilizadas algumas ferramentas para compor as atividades associadas a ataques do tipo Botnet. A primeira, Zeus[8], trata-se de um *malware* do tipo Cavalo de Tróia, podendo ser executada em sistemas operacionais Microsoft Windows. Suas utilizações incluem atividades maliciosas e criminais, sendo frequentemente empregado no roubo de informações bancárias, por meio da captura de pressionamento de teclas, bem como instalação de Ransomware, um tipo de *malware* que facilita a execução de atividades como o sequestro de dados, impedindo o acesso até o pagamento do resgate (Beaman, Barkworth, Akande, Hakak & Khan, 2021). O mesmo é difundido principalmente por meio de downloads *drive-by*, que consistem na incorporação do *software* malicioso ao *software* em que se tem o propósito de obter. Como complemento, uma Botnet de código aberto, chamada Ares[9], foi utilizada.

### 2.3.4 Negação de Serviço

Ataques de negação de serviço, também conhecidos como DoS, ou *Denial of Service*, se caracterizam por tornar uma máquina ou recurso de rede indisponível temporariamente. Normalmente, isso é efetuado com a inundação da máquina alvo com requisições, provocando a sobrecarga do sistema e tornando o mesmo incapaz de responder a requisições legítimas (Gupta & Badve, 2017).

Slowloris[10] é uma ferramenta de ataque de negação de serviço, que permite que uma única máquina comprometa um servidor web em outra máquina, requerendo uma mínima largura de banda e sem efeitos colaterais nos demais serviços e portas não relacionados. A mesma foi utilizada no CIC-IDS2018.

[8]https://github.com/Visgean/Zeus
[9]https://github.com/sweetsoftware/Ares
[10]https://github.com/gkbrk/slowloris

### 2.3.5 Negação de Serviço Distribuída

Esse ataque, também conhecido como DDoS, é uma variação do anterior, porém com maior robustez. O mesmo consiste na utilização de múltiplos sistemas comprometidos, como por exemplo, uma Botnet. Nesses casos, o tráfego de rede é muito maior, comprometendo serviços com maior escala.

High Orbit Ion Cannon, com abreviatura HOIC[11], é uma ferramenta de código aberto para teste de carga em redes e negação de serviço, que permite o ataque a até 256 URLs simultâneas. A mesma foi desenvolvida para substituir a Low Orbit Ion Cannon, ou LOIC, que era desenvolvida pela Praetox Technologies. No cenário do *dataset* CIC-IDS2018, a ferramenta HOIC foi empregada para realizar os ataques de DDoS, com o uso de 4 computadores distintos.

### 2.3.6 Ataques da Web

Ataques da Web surgem diariamente, devido ao aumento da atenção com segurança pelas organizações. *SQL Injection* é um dos ataques dessa família, que consiste em uma vulnerabilidade em que, por falta de filtragem na entrada de dados em aplicações Web, se torna possível que o atacante execute comandos SQL no servidor da aplicação. XSS, ou *Cross-Site Scripting*, é outro tipo de ataque bastante difundido, permitindo a injeção de *scripts* em páginas web, possibilitando sua execução pelos usuários do site comprometido. Além destes, há o ataque de força bruta sobre HTTP, que funciona por meio de repetidas tentativas para descoberta da senha do administrador (Hoque, Bhuyan, Baishya, Bhattacharyya & Kalita, 2014; Igure & Williams, 2008; University of New Brunswick, 2018).

No *dataset* utilizado neste trabalho, é utilizada uma aplicação com diversas vulnerabilidades como vítima, chamada Damn Vulnerable Web App (DVWA). A mesma foi concebida para permitir a desenvolvedores de aplicações web, a realização de testes em um ambiente legal. Para a automação dos ataques, foi realizado o desenvolvimento com o Framework Selenium[12].

### 2.3.7 Infiltração

Ataques de infiltração de dentro da rede habitualmente ocorrem com a exploração de um *software* com vulnerabilidades. Quando a exploração é sucedida, é criada uma *backdoor* na máquina vítima, possibilitando ao atacante a execução de ataques dentro da rede da vítima. Alguns exemplos são *IP Sweep*, *Full Port Scan*, e enumeração de serviços com a ferramenta

[11] https://sourceforge.net/projects/highorbitioncannon/
[12] https://www.selenium.dev

Nmap (Orebaugh & Pinkard, 2011).

No respectivo cenário do CIC-IDS2018, a exploração ocorre em uma aplicação vulnerável, com o Adobe Acrobat Reader 9. O ataque é iniciado com o envio de um documento malicioso por e-mail. Com o Metasploit Framework, uma *backdoor* é executada na máquina vítima, após a exploração por meio do documento malicioso. Assim, os ataques citados anteriormente são executados.

## 2.4 Inteligência Artificial

A Inteligência Artificial (IA) possui uma série de definições distintas citadas ao longo dos anos, que devido à complexidade do tema, tornam difícil sua conceituação de forma simples e robusta. A primeira menção ao termo ocorreu em 1955, pelo professor John McCarthy, a definindo da seguinte maneira: "O objetivo da IA é desenvolver máquinas que se comportam como se fossem inteligentes"(Ertel, 2018).

Outras definições incluem a apresentada na Encyclopaedia Britannica (1991): "IA é a habilidade de computadores digitais, ou computadores controlados por robôs, de resolver problemas que normalmente são associados à maior capacidade intelectual dos humanos". Essa definição apresenta imperfeições, pois possibilitaria reconhecer que a maioria dos computadores caracterizam IA, devido a capacidade de realizar tarefas como operações matemáticas complexas, ou memorizar longos textos, com facilidade (Ertel, 2018). Esse dilema pode ser resolvido com elegância, pela definição de Rich (1983): "Inteligência artificial é o estudo de como fazer com que computadores façam coisas em que, atualmente, pessoas são melhores". A autora foi capaz de caracterizar o que pesquisadores de IA vêm fazendo há tempos e por muitos anos adiante essa definição deve permanecer atualizada.

Norvig & Russell (2020) apresentam algumas definições segregadas em quatro abordagens, que historicamente foram seguidas por diferentes autores, com distintos métodos:

- **Agir humanamente:** associada ao processo de pensamento e raciocínio, mensura o sucesso com base na fidelidade com a performance humana. Uma abordagem para avaliação dessa característica, é o teste de TURING (1950), que é aprovado caso um interrogador humano, após a colocação de algumas questões, é incapaz de distinguir se as respostas foram de uma máquina ou de uma pessoa;
- **Pensar humanamente:** associada ao comportamento, mensura o sucesso com base na fidelidade com a performance humana. Essa capacidade pode ser avaliada comparando o processo de raciocínio observado em um humano e do programa;
- **Pensar racionalmente:** associada ao processo de pensamento e raciocínio, mensura o sucesso com base na racionalidade, ou performance ideal, ou seja, que o sistema faça

a coisa certa, considerando as informações que detêm. Compara-se com a aplicação da lógica, com o emprego de processos de raciocínio irrefutáveis;

- **Agir racionalmente:** associada ao comportamento, mensura o sucesso com base na racionalidade ou performance humana. Verifica-se pela similaridade a um agente racional, em que suas ações encontram o melhor resultado, ou em caso de incerteza, o melhor resultado esperado.

### 2.4.1 Aprendizado de Máquina

*Machine Learning* (ML), ou Aprendizado de Máquina, é um campo de pesquisa multidisciplinar, que envolve áreas como computação, estatística e probabilidade, teoria da informação, filosofia, psicologia, neurobiologia, entre outros. Sobre a definição desse termo, no que tange ao aprendizado, Mitchell (1997) define como qualquer programa de computador, capaz de melhorar sua performance em alguma tarefa por meio de experiência.

Com o advento da computação e sua crescente expansão para usos diversos, como na indústria, manufatura, comércio, finanças, entre outros, há uma geração e coleta contínua de dados. O mesmo ocorre com a digitalização presente na vida cotidiana, em tarefas associadas ao consumo, bem como nas mídias sociais, em que partes de nossas vidas são gravadas e se transformam em dados. Quaisquer sejam as fontes destes dados, se os mesmos permanecerem armazenados apenas por razões operacionais, associadas às funcionalidades que os geram, sua utilidade é restrita. Assim, procuram-se maneiras de aproveitar o potencial destes dados para a criação de produtos ou serviços úteis. Para essa transformação, o aprendizado de máquina tem sido uma peça fundamental (Alpaydin, 2021).

Para resolver um problema com o uso de computadores, é necessário um algoritmo, que consiste em uma série de instruções para transformar uma entrada em uma saída. Entretanto, para algumas tarefas, não existem algoritmos específicos. Como exemplo, para uma das tarefas em que se emprega o aprendizado de máquina, na predição de comportamento de consumidores, ou distinguir e-mails legítimos e *spam*. Nessas situações, é conhecida a saída esperada, bem como os dados para entrada, porém não se sabe como transformar a entrada na saída (Alpaydin, 2014).

Segundo os autores, a falta de conhecimento para a definição de um algoritmo para essa tarefa, pode ser compensada com dados. Sem dificuldade, é possível compilar milhares de mensagens de e-mail, previamente classificadas como *spam* ou não, submetendo ao computador, ou máquina, para aprendizado das características que constituem o *spam*. Apesar de não ser capaz de identificar o processo completo, com total exatidão, os resultados obtidos são uma boa e útil aproximação.

Ainda, conforme Alpaydin (2014), o aprendizado de máquina é a programação de compu-

tadores para otimizar um critério de performance, utilizando dados de exemplo ou experiências prévias. Enquanto o modelo é composto por uma série de parâmetros, o aprendizado consiste na otimização destes. Esse modelo pode ser preditivo, quando realiza predições do futuro, descritivo, quando obtém conhecimento a partir dos dados, ou uma combinação de ambos.

### 2.4.2 Mineração de Dados

De acordo com Witten et al. (2016), mineração de dados é definida como um processo de descoberta de padrões em dados. O mesmo deve ser automático, ou semi-automático, com o último sendo mais comum. Os padrões encontrados devem levar a alguma vantagem, e os dados utilizados existem em quantidades substanciais. Outra conceituação é a extração de informações implícitas, previamente desconhecidas e potencialmente úteis, dos dados.

Ainda segundo os autores, o aprendizado de máquina, mencionado na subseção anterior, fornece a base técnica para a mineração de dados, usado para a extração de informação a partir de dados brutos existentes em bancos de dados.

As primeiras publicações acerca de mineração de dados, apareceram no início dos anos 1990, consistindo em trabalhos apresentados em oficinas de KDD (*Knowledge Discovery on Databases*), ou descoberta de conhecimento em bancos de dados, também conhecida como extração de conhecimento. Essas publicações eram direcionadas principalmente a aspectos práticos e tecnologias na aplicação de KDD, sem aprofundamento quanto aos métodos (Piatetsky-Shapiro, Matheus, Smyth & Uthurusamy, 1994). As mesmas tiveram grande importância, servindo como fonte de possíveis aplicações e inspiração para novos casos de uso.

O processo de extração de conhecimento (KDD), foi definido por Fayyad, Piatetsky-Shapiro & Smyth (1996) como o processo não trivial de identificar padrões nos dados que sejam válidos, novos, potencialmente úteis, e por fim, compreensíveis. Por vezes, o termo KDD é utilizado intercambiavelmente com o termo mineração de dados, apesar de tecnicamente esta ser apenas uma das etapas do processo, devido à sua importância. O processo foi proposto de modo interativo e iterativo, sumarizado em 9 (nove) etapas:

- **Aprendizado do domínio de aplicação:** Inclui informações prévias relevantes, além dos objetivos da aplicação;
- **Criação do conjunto de dados alvo:** Seleção de um conjunto de dados em que a descoberta será realizada;
- **Limpeza e pré-processamento dos dados:** Operações básicas, como remoção de ruído e *outliers*, quando adequado, definição de estratégias para tratamento de dados faltantes, modelagem de dados, como tipagem, esquemas e mapeamento de valores faltantes ou desconhecidos;
- **Redução e projeção dos dados:** Descoberta de atributos úteis para representação dos

dados, redução de dimensionalidade e uso de métodos de transformação, com objetivo de diminuir a quantidade de variáveis consideradas, ou encontrar representações invariantes para os dados;

- **Escolha da função de MD:** Definição do propósito do modelo que será derivado pelo algoritmo de mineração de dados. Inclui opções como sumarização, classificação, regressão e *clustering*;
- **Escolha do algoritmo de MD:** Inclui a seleção do método a ser utilizado para busca de padrões nos dados, como decisão entre modelos e parâmetros apropriados, bem como a combinação do método escolhido com o critério do processo KDD, em que o propósito pode estar mais associado à compreensão do modelo, do que sua capacidade preditiva;
- **Mineração de dados:** Consiste na busca por padrões de interesse em um formato de representação específico, ou um conjunto dessas representações, incluindo árvores, regressão, *clustering*, modelagem de sequências, dependências e análises de linha;
- **Interpretação:** Interpretação de padrões extraídos, com a possibilidade de retorno a etapas anteriores, bem como a visualização, remoção ou tradução dos mesmos em termos compreensíveis pelos usuários;
- **Uso do conhecimento extraído:** Inclui a inclusão do conhecimento extraído nos sistemas de performance, utilização para definição de ações, ou documentação e relato para as partes interessadas.

### 2.4.3 Árvores de Decisão

Um dos mais antigos métodos de aprendizado de máquina supervisionado são as árvores de decisão. Elas possuem simplicidade no treinamento, bem como na predição, com exatidão em muitos domínios de aplicação, de acordo com Alpaydin (2021), material utilizado como referência para a descrição deste assunto, juntamente a Witten et al. (2016).

Com base em diferentes atributos de entrada, a árvore de decisão encontra as instâncias de treinamento com maior similaridade. A mesma é composta por nós de decisão e folhas. Iniciando na raiz da árvore, cada nó de decisão aplica um teste à entrada, com base em regras se-então. Dependendo do resultado obtido, é tomado um dos caminhos em direção ao próximo nó de decisão, ou por fim, à folha, fornecendo o resultado final da classificação. A figura 2.4 apresenta uma árvore com dois (2) nós de decisão, com o objetivo de classificar clientes de baixo ou alto risco.

Caso o atributo testado no nó de decisão seja nominal, o número de filhos geralmente será a quantidade de valores distintos para aquele atributo. Nessas situações, o atributo não será testado novamente além desse nó de decisão, em outros níveis da árvore. Caso o atributo seja numérico, habitualmente seu valor é comparado a outro valor constante, utilizando operações

como maior que ou menor que, resultando em apenas dois (2) filhos. Também há possibilidade de definir intervalos para comparação, assim, permitindo múltiplos filhos (Witten et al., 2016).

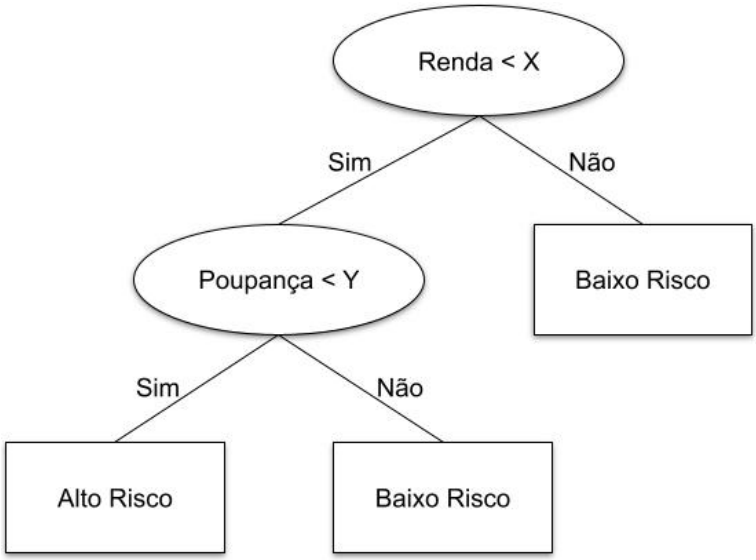


**Figura 2.4:** Árvore de decisão classificando clientes de alto e baixo risco.
Fonte: Adaptado de Alpaydin (2021)

Um dos aspectos que tornam esse tipo de algoritmo interessante, é a facilidade de compreensão. Cada um dos caminhos entre a raiz e uma folha da árvore pode ser traduzido em regras se-então. Entretanto, existem variações mais sofisticadas, com múltiplas árvores, em que a interpretação não é trivial, conforme será apresentado nas próximas subseções.

A construção de uma árvore de decisão é uma tarefa recursiva. Inicia-se com a definição de um atributo para se situar no nó raiz, e a partir daí, são criados alguns galhos, de acordo com o tipo do atributo. Isso faz com que o conjunto de exemplos seja dividido em subconjuntos, para cada valor ou faixa de valores do atributo. O processo então é repetido recursivamente, partindo das instâncias que passam do nó de decisão. Se em algum momento todas instâncias de um nó possuírem a mesma classe, essa parte da árvore para de ser construída, caracterizando uma folha.

Existem situações em que o algoritmo da árvore de decisão tende a criar uma árvore com grandes dimensões, contendo estruturas desnecessárias, como quando não existem padrões a serem encontrados. Esse fenômeno é chamado de *overfitting* e ocorre em todos os tipos de algoritmos de aprendizado. O mesmo ocorre quando o modelo treinado é muito dependente de detalhes dos exemplos que lhe foram fornecidos para o treinamento. Deste modo, o modelo é capaz de predizer com exatidão esses exemplos, porém por não encontrar padrões e ser incapaz de generalizar, não pode realizar predição em exemplos desconhecidos. A poda auxilia na disputa entre um bom ajuste aos dados de treinamento e redução de complexidade desnecessária.

Para combater o *overfitting*, é empregada a técnica de poda da árvore, que consiste em uma espécie de simplificação. A mesma funciona removendo nós de decisão que não são claramente

relevantes, podendo também ser aplicada durante a criação da árvore, evitando a criação de nós de decisão desnecessários, com base em critérios de parada predefinidos, como número mínimo de exemplos para divisão, profundidade máxima da árvore, entre outros.

### 2.4.4 *Ensemble Learning*

O aprendizado com base em conjuntos, ou *ensemble learning*, consiste na combinação de predições de diversos modelos treinados nos mesmos dados, com o objetivo de maximizar sua acurácia. É conhecido que essa técnica é capaz de produzir modelos poderosos, mas apesar da melhoria de performance, isso provoca também uma perda de interpretabilidade. Entretanto, existem maneiras de derivar descrições estruturadas inteligíveis com base no aprendizado desses métodos. Com esse tipo de aprendizado, pode não ser vantajosa a escolha do melhor dos modelos para o conjunto de dados, mas da utilização de todos eles com combinação dos resultados. A apresentação dos conceitos realizada ao longo desta subseção é baseada no material de Witten et al. (2016).

Os autores fazem uma analogia com a tomada de decisões críticas por pessoas, pontuando que usualmente essas decisões são tomadas com base em opiniões de diversos especialistas. No caso da mineração de dados, esses especialistas seriam os modelos de aprendizado. As distintas opiniões dos especialistas podem ser combinadas por meio de discussão, chegando a um consenso, ou ainda, com a escolha por meio de votação.

As técnicas com mais ampla utilização são chamadas *bagging*, *boosting* e *stacking*. Elas vêm sendo desenvolvidas ao longo das últimas duas décadas, apresentando performance surpreendente. Os modelos combinados resultantes da aplicação desses métodos compartilham a desvantagem da dificuldade de análise, pois podem incluir dezenas ou até centenas de modelos, tornando difícil a compreensão de quais fatores contribuem para a melhoria nas tomadas de decisão.

*Bagging* e *boosting* adotam a abordagem de votação para a combinação das decisões em tarefas de classificação, ou médias, caso a predição seja numérica. Entretanto, suas aplicações diferem no que tange a derivação dos modelos, em que quando utilizado *bagging*, os mesmos recebem pesos iguais, e com *boosting*, os modelos mais bem-sucedidos recebem pesos maiores. A aplicação de *bagging* visa reduzir a instabilidade em árvores de decisão. Pequenas mudanças no conjunto de treinamento podem fazer com que diferentes atributos dos nós de decisão sejam selecionados, criando também distintas ramificações na árvore. Essa técnica consiste na criação de diversos subconjuntos selecionados aleatoriamente a partir do domínio do problema, submetendo cada um destes à construção de uma árvore de decisão. Por fim, as árvores são combinadas por meio de votação para cada instância de teste. Deste modo, por meio da criação de diversas árvores, cada uma pode se especializar em diferentes partes do conjunto de dados, complementando uma à outra, aumentando a robustez do modelo. Resgatando a analogia an-

terior, para a tomada de decisões, é provável que sejam consultados diversos especialistas com experiência e habilidades em diferentes partes do domínio.

Uma das limitações na aplicação de *bagging*, é que só provoca melhoria quando os modelos têm diferenças significativas e cada um deles tem boa performance em uma parte razoável dos dados. *Boosting* tem diversas semelhanças com *bagging*, entretanto, busca explicitamente por modelos que são complementares aos outros. Além disso, o processo é iterativo, os modelos não são construídos individualmente, mas cada modelo é criado com influência da performance dos construídos anteriormente. Assim, os novos modelos são encorajados a se especializar em instâncias que foram incorretamente manipuladas pelos modelos prévios.

Por sua vez, as técnicas de *stacking* surgiram mais recentemente, e ao contrário das apresentadas anteriormente, combinam modelos de tipos distintos. Além disso, não existe uma definição aceita amplamente de como deve ser aplicada, consistindo apenas em uma ideia básica, que pode ser aplicada em diversas variações. A mesma também é menos difundida na literatura de aprendizado de máquina.

### 2.4.5 *Random Forest*

Um dos algoritmos mais difundidos que implementa o conceito de *bagging*, se chama *Random Forest*. O mesmo endereça a deficiência de acurácia encontrada em árvores de decisão simples, não deixando de aproveitar sua simplicidade (Norvig & Russell, 2020).

A construção do modelo inicia-se com a criação de um conjunto de dados *bootstrap*, do mesmo tamanho do original, contendo instâncias selecionadas aleatoriamente. Uma característica importante, é que a mesma instância pode ser selecionada mais de uma vez, além de que algumas instâncias podem não ser selecionadas para o *bootstrap*. Na sequência, uma árvore de decisão é construída com um subconjunto dos atributos, selecionados aleatoriamente.

Esse procedimento é repetido diversas vezes, introduzindo uma ampla variação nas árvores criadas, tornando o modelo mais efetivo do que árvores de decisão simples. Para a tarefa de classificação, um exemplo é fornecido ao modelo, passando pela predição de cada uma das árvores. Os resultados de todas as árvores são agregados, por meio de votação, obtendo assim o resultado final (Witten et al., 2016).

### 2.4.6 AdaBoost

AdaBoost é uma entre as diversas implementações do conceito de *boosting* com ampla utilização. Ela foi concebida especificamente para tarefas de classificação. Seu funcionamento parte do pressuposto de que o algoritmo é capaz de trabalhar com pesos para as instâncias. A

mesma faz o uso de *weak learners*, que são modelos simples, com capacidades pouco superiores ao acaso. Predominantemente, o *Adaboost* é composto por *weak learners*, ou árvores com somente um nó de decisão, chamados *stumps*. Cada árvore destas possui uma participação distinta na combinação dos resultados, ao final do processo. Os conceitos e funcionamento do algoritmo Adaboost estão apresentados de acordo com Norvig & Russell (2020) e Witten et al. (2016).

A construção do modelo inicia-se com a atribuição de pesos a todas as instâncias do conjunto de dados, que neste momento possuirão valor igual. Então, para cada atributo, é criada uma árvore *stump* e são computadas as suas performances, de acordo com o índice de Gini. A melhor delas, será a primeira árvore do modelo. Em seguida, é definida a participação da mesma na combinação do resultado final do modelo, associada à sua performance. Para garantir que os erros cometidos pela árvore sejam considerados pela próxima, os pesos das instâncias são atualizados, sendo que as incorretas têm seu peso aumentado, e as corretas, reduzido.

Na sequência, o índice de Gini ponderado é utilizado para a construção da próxima árvore, enfatizando a classificação correta dos exemplos anteriormente classificados incorretamente. O procedimento é repetido até que exista uma árvore para cada um dos atributos.

Para a realização da predição, um exemplo é fornecido ao modelo, e cada uma das árvores fornece sua classificação. Para a combinação e estabelecimento do resultado final, são somadas as participações de cada árvore, e agrupadas por resultado obtido. Por fim, a maior soma representará a classificação obtida.

### 2.4.7 *Gradient Boosting*

Árvores de decisão com *gradient boosting*, GBTs, ou GBDTs, de acordo com Witten et al. (2016), são algoritmos de aprendizado de máquina usados amplamente, devido a sua eficiência, acurácia e interpretabilidade, apresentando excelente performance em diversas tarefas de aprendizado.

Sua construção se dá de maneira aditiva, baseada em *weak learners*, para a criação de modelos com boa capacidade preditiva. Similar aos conceitos de AdaBoost, novas árvores são adicionadas de maneira sequencial, buscando a correção de erros cometidos pelas árvores prévias.

O processo é iterativo, e inicia-se com a criação de um modelo base simples, com profundidade fixa, ou um resultado comum a todas as instâncias. A cada iteração, com base em uma função de perda, são calculados os gradientes negativos em relação às predições obtidas. Esses gradientes são chamados pseudo-residuais e apontam a direção e magnitude com que o modelo atual precisa de ajustes.

Com base nesses pseudo-residuais, um *weak learner* é ajustado, com um limite de profun-

didade, a fim de garantir que a capacidade preditiva seja baixa, similar aos *stumps* empregados no Adaboost. Para evitar *overfitting*, é empregada uma taxa de aprendizado, que faz com que cada uma das árvores criadas contribua em pequenas proporções ao modelo geral. O mesmo então é atualizado, considerando a nova árvore criada com a proporção definida pela taxa de aprendizado.

O processo é iterativo, e as etapas citadas no parágrafo anterior, são repetidas até que o critério de convergência seja atingido, que pode ser um número de iterações fixo, ou uma quantidade de iterações sem melhoria, entre outros.

### 2.4.8 LightGBM

O aumento constante na quantidade de dados disponíveis, associado à expansão de *big data*, trouxe novos desafios às GBDT convencionais. Para definir os pontos de divisão dos nós de decisão, as GBDT precisam estimar o ganho de informação com todas as instâncias de todos os atributos, uma tarefa de elevado custo computacional.

O LightGBM (Ke et al., 2017) surgiu com o objetivo de endereçar algumas dessas limitações, por meio da redução no número de instâncias e atributos avaliados no treinamento, propondo duas novas técnicas. A primeira delas, se chama *Gradient-based One-side Sampling* (GOSS), que leva a uma estimativa de ganho de informação mais precisa, em relação à amostragem aleatória. Ela funciona por meio da avaliação dos gradientes, selecionando instâncias que possuem maiores gradientes, e removendo aleatoriamente instâncias com gradientes pequenos. Já a outra, é chamada *Exclusive Feature Bundling* (EFB), trazendo otimizações relevantes em aplicações em que o espaço de atributos é disperso, o que normalmente ocorre em conjuntos de dados com número elevado de atributos. A mesma funciona empacotando atributos que são praticamente exclusivos.

Diversas ferramentas de *gradient boosting* usam algoritmos que dependem de dados pré-ordenados para o aprendizado, o que é uma solução simples, porém de difícil otimização. O LightGBM emprega algoritmos baseados em histogramas, os quais criam compartimentos discretos para os valores contínuos. Deste modo, são obtidos ganhos em tempo de processamento, devido à redução na complexidade, dada pelo agrupamento dos dados obtido na discretização. Além disso, o uso de valores discretos em substituição aos contínuos, permite o uso de tipos de dados menores, reduzindo o uso de memória.

Outro recurso empregado pelo LightGBM, é o crescimento da árvore por folhas, o que permite que dada uma condição, apenas uma das folhas seja dividida. Devido à possibilidade de *overfitting* desta abordagem, é possível limitar a profundidade da árvore.

## 2.5 Preparação dos dados

Uma etapa fundamental ao processo de aprendizado de máquina é a preparação dos dados, ou pré-processamento, assim como definido por Fayyad et al. (1996). Essas tarefas também estão entre as que requerem maior esforço, consistindo em transformações aplicadas aos dados brutos para que possam ser processados por algoritmos de aprendizado, a fim de obter conhecimento ou realizar predições. Além disso, uma preparação adequada dos dados, produzindo atributos bem organizados e limpos, garantirá um resultado mais preciso pelos modelos.

### 2.5.1 Limpeza de dados

A presença de valores inválidos ou faltantes é um problema encontrado frequentemente em aprendizado de máquina. Antes de submeter os dados ao modelo para treinamento, é necessário o tratamento. Técnicas para correção de valores faltantes incluem (Alpaydin, 2014):

- **Entrada de dados**: geração de dados por meio de estimativas ou predição;
- **Interpolação**: derivação de valores com base em pontos de dados com proximidade;
- **Eliminação**: remoção dos registros ou atributos que possuem valores faltantes ou inválidos.

Ainda de acordo com o autor, pontos de dados que possuem grande diferença do restante dos dados são chamados *outliers*. Os mesmos podem consistir em erros de medição ou entrada de dados, ou ainda podem ser valores imputados corretamente, porém que caracterizam observações extremas ou incomuns, não representando a população analisada. Existem diversas abordagens para o tratamento de *outliers*, como a eliminação desses registros, ou substituição de seus valores por outros mais próximos do intervalo da distribuição.

Outro problema que pode estar presente nos dados é a duplicação, o que provoca viés aos resultados produzidos pelo modelo. Registros com essa características podem ser removidos ou mesclados para tratamento. Existem situações em que a duplicação pode ser empregada propositalmente, como na correção de problemas de balanceamento, também abordado nesse trabalho.

### 2.5.2 Transformação de dados

Além dos ajustes associados à limpeza dos dados, existe mais uma série de transformações aplicáveis aos dados, que têm como objetivo torná-los mais receptivos aos métodos de aprendizado de máquina, aumentando sua efetividade.

### Seleção de atributos

Em muitas situações existe uma quantidade elevada de atributos, inapropriada à manipulação pelo esquema de aprendizado. Usualmente, a maior parte desses atributos são irrelevantes ou redundantes. Muitos algoritmos, como os baseados em árvores de decisão, são capazes de selecionar os atributos de maior relevância, porém frequentemente sua performance pode ser melhorada com seleção prévia de atributos. Utilizando uma quantidade reduzida de atributos também provoca uma redução no espaço de armazenamento necessário, bem como recursos computacionais durante a fase de treinamento e criação do modelo (Alpaydin, 2014).

A seleção manual dos atributos, com base em um conhecimento profundo acerca dos dados e do negócio, é habitualmente a melhor alternativa. Entretanto, existem métodos automatizados com grande utilidade. Para a definição de um subconjunto de atributos, existem duas abordagens distintas. A primeira, conhecida como método de filtro, consiste em realizar uma avaliação independente baseada em características gerais dos dados. Já o método *wrapper*, é formado pela avaliação do subconjunto com o algoritmo que será empregado para aprendizado.

### Amostragem

Diversas aplicações de aprendizado de máquina envolvem grandes volumes de dados, podendo haver necessidade de redução na quantidade de exemplos. Técnicas de amostragem podem sem aplicadas a fim de sanar esses problemas, conforme apresentado nos próximos parágrafos, de acordo com Kantardzic (2019).

Para essas situações, podem-se aplicar as técnicas chamadas *undersampling*, em que novos exemplos são criados, com base em outros já existentes. Em uma das abordagens possíveis, exemplos aleatórios são selecionados no conjunto de dados e removidos. Esse processo é bastante simples e direto, entretanto isso pode fazer com que registros com características de interesse sejam removidos, alterando características da população.

Outra abordagem possível para *undersampling* é a estratificada. Nesse tipo de amostragem, a população é dividida em diversos subgrupos, de acordo com suas características. Então, é selecionada aleatoriamente a fração de exemplos necessária para todos os subgrupos, mantendo a representatividade.

Em conjuntos de dados desbalanceados, em que há um desequilíbrio entre a quantidade de exemplos das classes, essa técnica pode ser empregada para a redução do número de exemplos da classe predominante, aproximando-a da classe minoritária.

Outras técnicas de amostragem, denominadas *oversampling*, consistem na geração de novos exemplos, com aplicação em situações em que há desbalanceamento. Isso pode ser realizado com replicação de exemplos já existentes, de classes minoritárias, bem como com a

geração sintética de novos pontos de dados com a classe minoritária. Uma implementação dessa técnica será apresentada com detalhes no capítulo 4.

## 2.6 Avaliação de modelos

A avaliação é uma etapa chave para progredir em mineração de dados. Para estabelecer quais algoritmos possuem efetivamente melhores resultados, quando aplicados a um conjunto de dados, ou quais transformações aplicadas no pré-processamento provocaram melhora, é fundamental que existam maneiras sistemáticas para avaliação e comparação.

Em problemas de classificação, como o abordado no presente trabalho, a medição de performance do modelo pode ser obtida por diversas métricas, como a taxa de erro. O classificador construído realiza a predição da classe de cada exemplo, e caso esteja correto, é considerado um sucesso, ou caso contrário, erro. A taxa de erros consiste na proporção de erros em relação ao total de exemplos classificados, mensurando a performance geral do modelo.

As exposições de conceitos acerca do tema de avaliação de modelos, detalhadas ao longo desta seção, foram elaboradas de acordo com Witten et al. (2016) e Ertel (2018).

### 2.6.1 *Holdout*

Quando há disponibilidade de um suprimento de dados amplo, a avaliação do modelo tem menor complexidade. Os dados podem ser divididos em um conjunto de treinamento para criação do modelo e outro para testes e coleta dos resultados. Entretanto, a disponibilidade de dados rotulados em grande volume não é encontrada habitualmente, pois muitas vezes depende do esforço de especialistas para rotulação.

A técnica que consiste na divisão do conjunto em duas partes para treino e teste, é chamada de *holdout*. Essa técnica é frequentemente aplicada com a reserva de um terço das instâncias para testes, e os outros dois terços para treinamento. Um dos problemas inerentes a essa técnica, é a possibilidade de perda da representatividade da população. Ou seja, o subconjunto selecionado para a realização dos testes pode possuir instâncias com uma classe não presente no conjunto de treinamento, fazendo com que o modelo não seja capaz de classificar exemplos dessa classe. Similarmente às técnicas de amostragem apresentadas, o processo de estratificação pode ser aplicado ao *holdout*, algumas vezes auxiliando com a representatividade desproporcional entre os conjuntos. Além disso, a fim de encontrar um bom modelo, é necessário utilizar a maior quantidade de dados possível. Entretanto, para garantir uma boa estimativa de erro na fase de testes, também é preciso maximizar a quantidade de exemplos utilizados.

### 2.6.2 Cross-validation

Para endereçar esses problemas, a técnica mais amplamente utilizada denomina-se *cross-validation*, ou *k-fold cross-validation*. A mesma consiste na divisão aleatória do conjunto de dados em $k$ *folds*, ou partições, com aproximadamente o mesmo tamanho. Então, são realizadas $k$ iterações, submetendo $k-1$ *folds* para treinamento do modelo e a partição remanescente para teste. Deste modo, todas instâncias serão usadas exatamente uma vez para teste. Por fim, os resultados das iterações são utilizados para calcular uma média, produzindo uma estimativa de erro global (Ertel, 2018).

A figura 2.5 apresenta o processo *k-fold cross-validation* com $k = 5$. De acordo com o especificado, com esse parâmetro o conjunto de dados será dividido em 5 partições e serão repetidas 5 iterações. Na primeira, o *fold* 1 será reservado para a realização dos testes e os *folds* 2 a 5 para o treinamento do modelo. Na próxima iteração, o fold 2 será utilizado para testes, e os *folds* 1, 3, 4 e 5 para treinamento do modelo. Esse processo se repetirá sucessivamente até que todos *folds* sejam utilizados para testes uma vez.

Segundo Witten et al. (2016), a maneira padrão de predizer a taxa de erros utilizando um conjunto de dados fixo para aprendizado, é o *10-fold cross-validation estratificado*. Ou seja, o conjunto de dados é dividido em 10 partições aleatoriamente, mantendo a representatividade de cada classe. O valor de $k = 10$ demonstrou ser o mais adequado para obtenção das melhores estimativas de erro, tanto com evidências teóricas, quanto com experimentações extensivas, conduzidas em diversos conjuntos de dados distintos.

Ainda de acordo com o autor, é importante que os testes sejam conduzidos sobre dados ainda não conhecidos pelo modelo e não utilizados durante o treinamento, pois nesses casos, os resultados obtidos costumam ser otimistas e não refletem a performance do modelo com dados reais utilizados futuramente. Esse conjunto de dados ainda não conhecido pelo modelo, é denominado conjunto de testes. Ademais, deve ser garantido que tanto o conjunto de treinamento, quanto o de testes, possuam exemplos que sejam representativos acerca do problema.

### 2.6.3 Métricas

Há uma série de métricas para qualificação de modelos, permitindo avaliar distintos aspectos. As mesmas devem ser escolhidas com adequação às características do conjunto de dados e tipo de atividade realizada. Para uma avaliação satisfatória dos efeitos provocados pelos tratamentos realizados na base, algumas métricas podem ser utilizadas. O cálculo destas se dá com base em quatro (4) conceitos, das matrizes de confusão. Elas representam os quantitativos de valores previstos e os reais. Os conceitos são apresentados na tabela 2.2 e explicados em seguida:

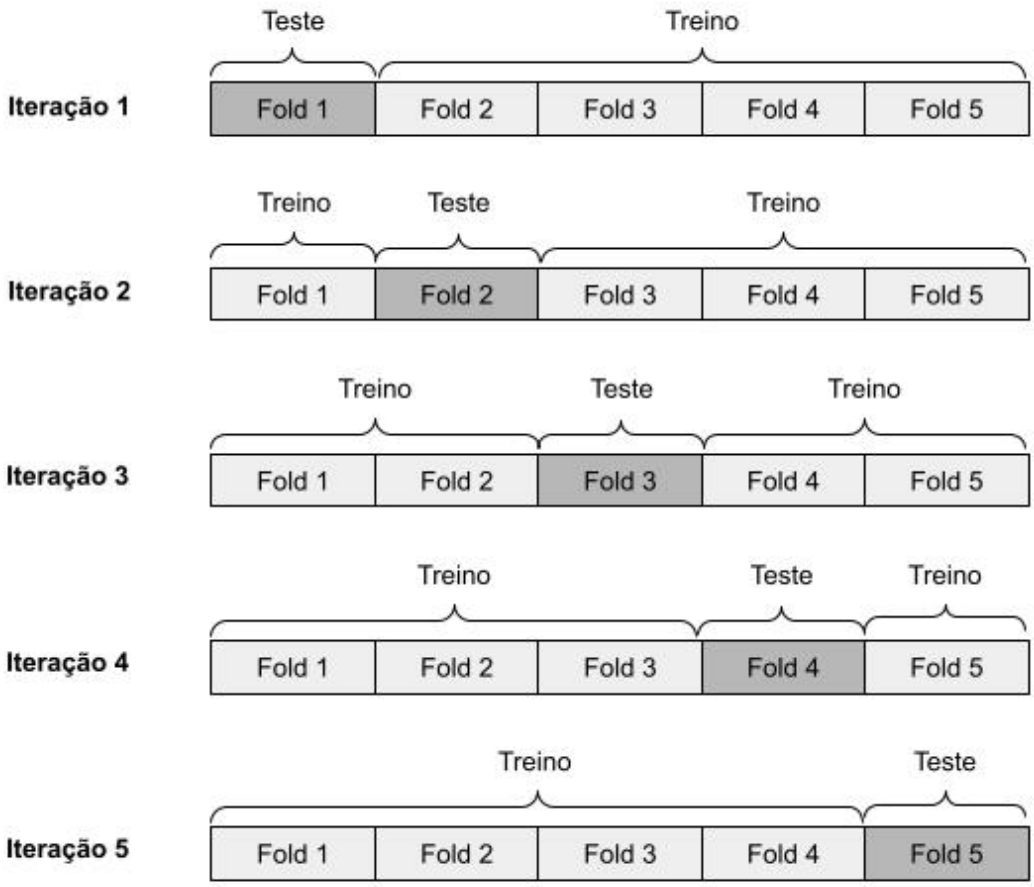


**Figura 2.5:** 5-fold cross-validation.
Fonte: Autor

- Verdadeiros positivos (VP): Quantidade de exemplos positivos classificados corretamente;
- Falsos positivos (FP): Quantidade de exemplos positivos classificados incorretamente;
- Verdadeiros negativos (VN): Quantidade de exemplos negativos classificados corretamente;
- Falsos negativos (FN): Quantidade de exemplos negativos classificados incorretamente.

**Tabela 2.2:** Matriz de confusão.

| | | **Previsto** | |
|---|---|---|---|
| | | Negativo | Positivo |
| **Real** | Negativo | **VN** | **FP** |
| | Positivo | **FN** | **VP** |

Fonte: Autor

Uma das métricas que possui utilização mais difundida é a acurácia, porém possui limitações. Ela pode ser errônea em situações onde há desbalanceamento do conjunto de dados, enfatizando o desempenho da classificação para a classe majoritária, podendo produzir resultados excessivamente otimistas. A acurácia pode ser calculada, baseando-se nos conceitos da

matriz de confusão, por meio da equação

$$Acurácia = \frac{VN + VP}{VN + FP + FN + VP}$$

Outras métricas comumente utilizadas são a precisão e sensibilidade. A precisão demonstra o quão correta é a predição dos valores positivos pelo modelo. Já a sensibilidade, ou *recall*, tem a capacidade de avaliar qual a capacidade do modelo de classificar corretamente exemplos da classe positiva (Kulkarni et al., 2020).

$$Precisão = \frac{VP}{VP + FP} \qquad Sensibilidade = \frac{VP}{VP + FN}$$

Com base na combinação destas, existe a métrica denominada F-score, F-rate, ou F1 score, que consiste em uma média, contabilizando falsos positivos e falsos negativos. A mesma é definida pela equação

$$F - score = 2 \times \frac{Sensibilidade \times Precisão}{Sensibilidade + Precisão}$$

Além dessas métricas, há a taxa de falsos positivos (TFP), também denominada taxa de alarmes falsos, que computa a quantidade de exemplos negativos incorretamente classificados sobre o total de negativos e a taxa de falsos negativos (TFN), ou taxa de perda, que consiste na quantidade de exemplos positivos incorretamente classificados em relação ao total de exemplos positivos.

$$TFP = \frac{FP}{FP + VN} \qquad TFN = \frac{FN}{FN + VP}$$

O coeficiente de correlação de Matthews (MCC), foi concebido com o propósito de fornecer uma taxa estatística mais confiável em relação à performance de todas quatro (4) categorias da matriz de confusão, para problemas de classificação binária. Essa métrica possui menor tendência a apresentar resultados excessivamente otimistas (Chicco & Jurman, 2020), e seu cálculo é dado por

$$MCC = \frac{VP \times VN - FP \times FN}{\sqrt{(VP + FP)(VP + FN)(VN + FP)(VN + FN)}}. \tag{2.1}$$

Outra linha de métodos para avaliação de performance de modelos de aprendizado de máquina, são os baseados em gráficos *receiver operating characteristics* (ROC), ou de características operacionais do receptor. Com utilização ampla em detecção de sinais, para distinção entre taxas de acerto e de alarme falso, além de sistemas de tomada de decisão em diagnóstico médico, as análises ROC também ganharam vasto uso para comparação de modelos de aprendizado de máquina, devido à percepção de que métricas como acurácia podem não ser satisfatórias (Fawcett, 2006).

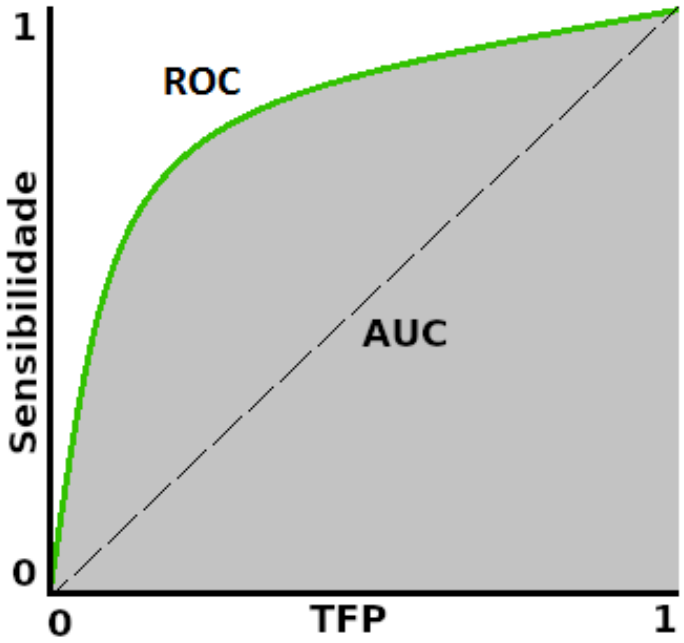


**Figura 2.6:** Gráfico de características operacionais do receptor (ROC).
Fonte: Adaptado de Witten et al. (2016)

Gráficos ROC consistem em representações de duas dimensões, em que a sensibilidade é traçada no eixo Y, e a taxa de falsos positivos no eixo X. Especialmente em modelos onde é possível obter uma pontuação, ou probabilidade de uma classe para o exemplo, essa análise possui grande potencial. Com base em uma limiar, o exemplo pode ser definido como pertencente à classe, caso a pontuação obtida seja maior ou igual à limiar, ou não pertencente caso contrário. Com variações dessa limiar sobre todo o intervalo, são obtidas as métricas de sensibilidade e taxa de falsos positivos, que podem então ser traçadas no gráfico. Com a utilização da ROC, é possível obter uma métrica denominada *area under receiver operating characteristics* (AUC), ou área sob características operacionais do receptor, que consiste na área abaixo da curva ROC, representada na figura 2.6 como a área destacada com preenchimento cinza. A AUC pode ser interpretada como a probabilidade de identificar corretamente a classe positiva, quando confrontado com um exemplo selecionado aleatoriamente entre as classes (Van Calster, Van Belle, Condous, Bourne, Timmerman & Van Huffel, 2008). Com base em critérios formais, tanto teóricos quanto empíricos, Huang & Ling (2005) demonstram que a AUC é uma métrica de desempenho melhor que a acurácia.

Apesar das vantagens apresentadas sobre a métrica AUC, a mesma foi concebida para utilização em problemas de classificação binária. Para aplicação em problemas de classificação multi-classe, existem algumas técnicas como a Um-versus-outros (UvO), em que são realizadas comparações de cada uma das classes com as demais. A média geral pode ser obtida tanto aritmeticamente, como de maneira ponderada, de acordo com a representatividade de cada classe no conjunto de dados.

Para garantir que eventuais diferenças encontradas nos resultados coletados não sejam apenas devido ao acaso, ou a erro nas estimativas de taxas de erro, é importante a formulação de hipóteses e realização de testes estatísticos. Deste modo, pode-se estabelecer que de acordo com um nível de confiança determinado, as diferenças apresentadas são significativas.

Segundo Demăr (2008), em aprendizado de máquina, a confirmação de que um determinado método é definitivamente funcional, pode ser provada por meio dos testes estatísticos. Porém, prendendo-se a isso, pode-se perder outras virtudes das ideias propostas, que podem ser verificadas por meio da compreensão e justificativa do método. O autor levanta alguns problemas com as abordagens existentes para testes estatísticos em aprendizado de máquina, porém conclui que os rituais de testagem tradicionais podem ser mantidos, porém sempre mantendo consciência sobre as limitações existentes.

# Capítulo 3

# Trabalhos Relacionados

No âmbito dos trabalhos relacionados à segurança computacional e detecção de intrusão, a literatura acadêmica demonstra um crescente interesse na aplicação de técnicas avançadas de aprendizado de máquina. A fim de fornecer uma base sólida para a contribuição desse estudo, bem como identificar lacunas para pesquisa, este capítulo explora a evolução na aplicação de aprendizado de máquina em detecção de intrusão, além de avaliar o estado da arte acerca do tema.

## 3.1 Histórico

Japkowicz & Stephen (2002) realizaram um estudo sistemático sobre o problema do aprendizado de máquina em *datasets* desbalanceados, em três (3) partes. Primeiramente, apresentam alguns entendimentos acerca do desbalanceamento, relacionando a complexidade conceitual, tamanho de conjuntos de dados e níveis de desbalanceamento. Em seguida, são apresentadas estratégias para endereçar esse problema, entre as citadas no capítulo anterior, com experimentação utilizando árvores de decisão C5.0 (Quinlan, 2001). Na terceira parte, é avaliado se os problemas encontrados com desbalanceamento são exclusivos para o algoritmo C5.0. Os resultados apresentam alguns pontos interessantes, como o de que independente do tamanho do conjunto de treinamento, domínios separáveis linearmente aparentam não ser sensíveis ao desbalanceamento. Também é demonstrado que à medida que o grau de complexidade aumenta, níveis menores de desbalanceamento provocam altas taxas de classificação incorreta.

Os experimentos com MLP, demonstraram uma menor sensibilidade a desbalanceamento do que as árvores de decisão C5.0. Além disso, a aplicação de *oversampling* a modelos MLP é mais efetiva do que *undersampling*, para *datasets* com alto desbalanceamento, porém essa diferença é menos significativa quando o desbalanceamento é menor.

O terceiro algoritmo empregado, SVM, demonstrou ser mais robusto para aplicação em *datasets* desbalanceados, apresentando resultados similares independente do desbalanceamento, com redução de eficiência quando aplicado *undersampling*. Por fim, os autores concluem que quanto maior o nível de desbalanceamento, maior a complexidade do conceito, e menor o ta-

manho do conjunto de dados, maior o efeito provocado pelo desbalanceamento do *dataset*.

No trabalho de Seo, Kim & Lo Bosco (2018), os autores realizaram o tratamento do desbalanceamento do *dataset* KDD Cup 99, com o método de *oversampling* SMOTE. Foram buscadas as razões de *oversampling* com a maior eficiência para as classes raras, por meio da geração de diversas razões aleatórias e criação de um modelo numérico para otimização, reduzindo a quantidade de experimentos necessários. Com a utilização de classificadores SVM, *Long Short Term Memory* (LSTM) e árvores de decisão (DT), foi observada uma melhoria de performance, em comparação às abordagens anteriores. A métrica coletada foi a sensibilidade ou *recall*. Entre as classes avaliadas, as melhores razões obtidas foram de 1000 vezes para a classe U2R (User to Root), 451 para R2L (Root to Local) e 1 para Probe, em que *oversampling* não provocou melhoria.

Por meio de uma busca exaustiva por trabalhos relevantes e revisados por pares, no contexto de detecção de intrusão com o uso do CICIDS2018, Leevy & Khoshgoftaar (2020) conduziram um estudo compreensivo, coletando as melhores métricas obtidas em cada um dos trabalhos. Os autores puderam observar performances incomumente altas, com alguns estudos reportando métricas de precisão de 100%, outros com sensibilidade de 100%, e ainda 3 (três) estudos apresentando AUC perfeita. Segundo os autores e a literatura disponível, tais realizações podem indicar *overfitting* dos modelos. Neste estudo, também foi relatado que a métrica que prevalece nos trabalhos abordados é a acurácia, métrica esta que não possui robustez para aplicação a conjuntos de dados com desbalanceamento, como o CICIDS2018. Alguns destes, também apresentaram as métricas precisão e sensibilidade, mas entre os estudos avaliados, apenas quatro (4) utilizaram a métrica AUC.

Outro ponto que chamou a atenção dos autores, é que apesar do desbalanceamento do *dataset*, apenas uma parcela inferior à metade dos trabalhos abordou técnicas para o tratamento deste problema.

Ademais, considerando que 60% dos cientistas de dados classificam a limpeza de dados como a tarefa que mais consome esforços no aprendizado de máquina (Ahmad & Aziz, 2019), o estudo revela que nenhum dos trabalho discute satisfatoriamente a limpeza dos dados, omitindo detalhes sobre a manipulação dos dados, o que também prejudica a reproduzibilidade dos experimentos.

Assim como observado na literatura posterior a este estudo, os autores também puderam demonstrar o negligenciamento da condução de análises estatísticas nos trabalhos pesquisados.

Com o objetivo de fornecer uma contribuição relevante, a abordagem do presente trabalho leva em consideração as diversas deficiências identificadas durante a pesquisa de trabalhos relacionados.

## 3.2 Estado da Arte

Faker & Dogdu (2019) realizaram um estudo abordando a integração de técnicas avançadas de aprendizado de máquina e *big data* a fim de melhorar a performance de IDS. Duas técnicas *ensemble* baseadas em árvores foram utilizadas, Random Forest (RF) e Gradient Boosting Tree (GBT), além de redes neurais do tipo Deep Feed-Forward (DNN). A construção dos modelos foi realizada com uso dos *datasets* CIC-IDS-2017 e UNSW-NB15. A abordagem dos autores inclui uma série de etapas, como o pré-processamento dos dados, em que a seleção de atributos foi efetuada de maneira não supervisionada com o algoritmo de *clustering* K-means, utilizando métrica de homogeneidade, a fim de identificar os atributos de maior relevância. Na sequência, os subconjuntos foram submetidos aos algoritmos para criação dos modelos com as três (3) técnicas, utilizando 5-fold cross-validation para estimar e melhorar a performance dos modelos. Entre os resultados obtidos com o *dataset* CIC-IDS-2017, destaca-se a eficácia da GBT para a classificação binária, obtendo acurácia de 99,99% com os atributos selecionados, superior aos 99,81% do *dataset* completo, também superando os resultados de RF e DNN. Para classificação multi-classe, GBT não foram utilizadas, por não possuir suporte quando utilizada no Apache Spark. Neste tipo de classificação, os resultados de DNN foram superiores à RF.

Tahir et al. (2019) utilizam uma abordagem alternativa para endereçar conjuntos de dados desbalanceados, por meio da utilização de algoritmos genéticos na definição da função de aptidão, que sucedem na resolução de uma série de problemas de classificação. Segundo os autores, as abordagens tradicionais para o desbalanceamento possuem problemas como enviesamento para a classe majoritária, elevado custo computacional e problemas de armazenamento ou alto tempo de treinamento. O estudo foi conduzido com as perspectivas de eficiência e eficácia, avaliando 14 (quatorze) algoritmos e 30 (trinta) *datasets* desbalanceados dos repositórios UCI [1] e KEEL [2], comparando seus resultados com outro algoritmos de estado da arte. O modelo proposto, EIG-GA, apresenta melhor performance nas 5 métricas avaliadas. Também destaca-se a sumarização de vantagens e desvantagens dos diversos métodos de tratamento de desbalanceamento apresentada no trabalho.

Com o objetivo de propor um sistema de detecção de intrusão especializado em classificar ataques do tipo Botnet, Kanimozhi & Jacob (2019) utilizaram o *dataset* CIC-IDS2018 junto ao algoritmo MLP, realizando também otimização de hiper-parâmetros. Foram selecionados 1048575 exemplos do conjunto de dados, com a manutenção de 80 atributos. Ademais, o processo de validação *10-fold cross-validation* foi empregado. Como métricas de avaliação, os autores coletaram a AUC e F-score. As contribuições deste trabalho são limitadas, porém os resultados apresentados sugerem que a utilização do algoritmo MLP sem otimização de hiper-parâmetros leva a *overfitting* do modelo.

No trabalho de Karatas et al. (2020), é apresentada uma comparação de IDS utilizando

[1] https://archive.ics.uci.edu/ml/index.php
[2] https://sci2s.ugr.es/keel/datasets.php

6 (seis) algoritmos de aprendizado de máquina, com o *dataset* CIC-IDS2018. Como o desbalanceamento é presente neste, os autores aplicaram a técnica de amostragem SMOTE. Dentre as implementações, a que utilizou Gradient Boosting, com o algoritmo LightGBM, apresentou a melhor acurácia média após o balanceamento. Entretanto, o mesmo também demonstrou maior sensibilidade ao desbalanceamento, quando comparado ao Adaboost e árvores de decisão, apesar de ser menos sensível que algoritmos como KNN (K-Nearest Neighbors) (Mucherino, Papajorgji & Pardalos, 2009). Os autores também puderam concluir que a aplicação de *oversampling* a este conjunto de dados, proporciona melhora na acurácia, demonstrando entre 4,01% e 30,59% de acréscimo.

Buscando aproveitar os benefícios de diversos algoritmos de aprendizado de máquina, Fitni & Ramli (2020) propuseram a utilização de *ensemble* para detecção de intrusão com o *dataset* CIC-IDS2018. A abordagem inicia-se com o pré-processamento dos dados, compreendendo a transformação e a remoção de valores inválidos, remoção de atributos com altas correlações e normalização das diversas classes disponíveis em comportamento benigno e de ataque, transformando o problema em classificação binária. A fase de treinamento foi realizada submetendo o subconjunto de treinamento, que possui 80% das instâncias, pré-processado, a sete (7) algoritmos de classificação. Para composição do modelo *ensemble* final, foram selecionados os três (3) algoritmos que apresentaram os melhores resultados, empregando a técnica de votação para a composição do resultado final da classificação. Na sequência um subconjunto com 20% das instâncias foi utilizado para teste, com a coleta de métricas como acurácia, precisão, sensibilidade e F-score. A criação do *ensemble* foi composta pelos algoritmos de Gradient Boosting, Árvore de Decisão e Regressão Logística, devido ao baixo tempo de predição associado aos bons resultados gerais obtidos.

Leevy et al. (2021) conduziram um estudo acerca do *dataset* CIC-IDS2018, investigando três (3) pontos principais, por meio de experimentação, coleta de métricas robustas em relação a conjuntos de dados desbalanceados e condução de testes estatísticos. O primeiro ponto avaliado é o impacto da seleção de atributos na performance dos classificadores. Por meio da construção de um *ensemble* com técnicas baseadas em ranking e filtragem, bem como aprendizado supervisionado, selecionando os atributos comuns em ambas, os autores demonstraram obtenção de resultados similares ou melhorados, comparando o uso de subconjunto com seleção dos 15 melhores atributos ao uso do *dataset* completo. Outro ponto avaliado, foi o efeito da inclusão do atributo categórico *Dst Port* na performance dos algoritmos entre os avaliados, capazes de lidar com esse tipo de atributo. Através da condução dos experimentos e testes estatísticos, chegou-se à conclusão de que a inclusão desse atributo tem impacto na performance. Também foi observado que após a inclusão do atributo, houve um aumento na estabilidade dos resultados obtidos. Por fim, o terceiro questionamento foi associado à escolha do classificador, onde os autores avaliaram a performance de RF, DT, NB, LR, CatBoost, LightGBM e XGBoost. Com a condução de testes estatísticos, os autores puderam confirmar a influência do classificador na performance obtida. Cabe-se destacar que o LightGBM esteve consistentemente entre os que apresentaram os melhores resultados.

Tang, Zhao, Wu & Wang (2021) propuseram a aplicação do algoritmo LightGBM juntamente a uma função de perda sensível a custo, a um conjunto de dados com algumas características similares ao CIC-IDS2018, como alto desbalanceamento, alta dimensionalidade e a presença de múltiplas classes. O problema consiste na detecção de falhas em caixas de engrenagens de turbinas eólicas. Os *datasets* disponíveis associados à tarefa possuem uma pequena parcela de exemplos com a presença de falhas, com o menor deles possuindo aproximadamente 10% das instâncias com uma classe que representa falhas. Apesar do desbalanceamento ser menor do que o observado no CIC-IDS-2018, o número de instâncias também é menor em ordens de magnitude, podendo provocar um efeito mais significativo, de acordo com as descobertas de Japkowicz & Stephen (2002), apresentadas previamente. Com a abordagem dos autores, demonstrou-se a capacidade do método em reduzir as taxas de classificação incorreta das classes minoritárias, em comparação a outros métodos avançados de detecção de falhas, incluindo Adaboost, GBDT e XGBoost com funções de perda sensíveis a custo. As métricas avaliadas foram a taxa de falsos positivos, taxa de falsos negativos e o coeficiente de correlação de Matthews, com uso de *5-fold cross-validation* para evitar *overfitting* dos modelos.

Songma, Sathuphan & Pamutha (2023) realizaram um estudo amplo em detecção de intrusão com o conjunto de dados CSE-CIC-IDS2018, conduzindo uma investigação em três (3) partes. O estudo contemplou tarefas de pré-processamento como limpeza dos dados, análise exploratória e processos de normalização, além da aplicação das técnicas de redução de dimensionalidade PCA (*Principal Component Analysis*) e seleção de atributos com Random Forest. Os experimentos incluíram a aplicação de 8 (oito) métodos distintos de aprendizado de máquina aos dados após as atividades de pré-processamento realizadas, com coleta abrangente de métricas, tanto estatísticas, trazendo acurácia, precisão, sensibilidade, F-score, acurácia balanceada, ROC AUC, entre outros, quanto o tempo de uso de processador e tamanho dos modelos gerados. Os resultados dos experimentos permitiram a comparação entre as técnicas de redução de atributos, métodos de normalização, e entre os métodos de aprendizado. Devido à ampla disponibilidade de critérios de avaliação, os autores optaram pela métrica ROC AUC, juntamente ao tempo de processador, para fins de avaliação e classificação dos modelos gerados. Os autores puderam concluir que os modelos gerados com o algoritmo XGBoost, e redução de dimensionalidade com a técnica PCA mantendo 11 atributos, foram os que proporcionaram os melhores resultados. Além disso, a escolha da técnica aplicada para normalização não demonstrou mudanças significativas nos resultados.

A avaliação de distintos métodos de aprendizado de máquina aplicados ao *dataset* CIC-IDS2018 também foi realizada por Chimphlee & Chimphlee (2023a), juntamente à aplicação de Random Forest para seleção de atributos e coleta de diversas métricas de avaliação. Além disso, este estudo contemplou técnicas de *undersampling* e SMOTE para endereçar o problema do desbalanceamento encontrado neste conjunto de dados. Os autores aplicaram uma técnica de validação dividindo dos dados em conjuntos de treino, teste e validação. Com a avaliação dos resultados, os mesmos puderam concluir que a técnica MLP, juntamente à aplicação de seleção de atributos e balanceamento, foi a que trouxe o modelo, tendo em vista as métricas precisão,

taxa de falsos positivos e ROC.

No mesmo ano, os autores publicaram outro trabalho utilizando o mesmo conjunto de dados, abordando os desafios do desbalanceamento. Neste, a seleção de atributos por ganho de informação foi aplicada, com a submissão do conjunto de dados com os atributos selecionados a 5 (cinco) classificadores. Também foi realizado balanceamento utilizando a técnica SMOTE, gerando novas instâncias para a classe minoritária, que corresponde aos ataques, deixando-a com o mesmo número de instâncias que a classe *Benign*, que caracteriza comportamento normal. Por meio da coleta e avaliação de quatro (4) métricas estatísticas de ampla utilização, os autores puderam concluir que o método *ensemble* com *bagging* proporcionou a melhor eficácia do ponto de vista da acurácia e F-score. Ademais, a aplicação de seleção dos 19 melhores atributos por ganho de informação demonstra uma melhora nas mesmas métricas (Chimphlee & Chimphlee, 2023b).

A Tabela 3.1 apresenta os principais trabalhos relacionados utilizando o mesmo conjunto de dados, demonstrando as principais características observadas.

**Tabela 3.1:** Trabalhos relacionados.

| *Trabalho* | *Dataset* | *Método* | *Pré-processamento* | *Observações* |
|---|---|---|---|---|
| Fitni & Ramli (2020) | CIC-IDS2018 | Ensemble | Seleção de Atributos | Ensemble composto por 3 métodos, incluindo GBDT e DT. |
| Karatas et al. (2020) | CIC-IDS2018 (Parcial) | LightGBM | SMOTE | Balanceamento proporcionou melhorias. |
| Leevy et al. (2021) | CIC-IDS2018 | Múltiplos | Seleção de Atributos | LightGBM trouxe os melhores resultados. |
| Chimphlee & Chimphlee (2023a) | CIC-IDS2018 | Múltiplos | *Undersampling*, SMOTE e Seleção de Atributos | MLP e XGBoost destacaram-se. |
| Songma et al. (2023) | CIC-IDS2018 (DoS e DDoS) | Múltiplos | Normalização e Seleção de Atributos | XGBoost trouxe os melhores resultados. |
| Chimphlee & Chimphlee (2023b) | CIC-IDS2018 | Múltiplos | SMOTE e Seleção de Atributos | *Ensemble* trouxe os melhores resultados, porém não foi especificada sua composição. |

# Capítulo 4

# Materiais e Métodos

## 4.1 Local de Experimentação

A experimentação do presente trabalho foi conduzida no ambiente de computação em nuvem da Amazon [1]. A escolha da instância EC2 ofereceu a escalabilidade necessária para lidar com o conjunto de dados de grandes dimensões e a capacidade de ajustar os recursos de acordo com as demandas específicas do experimento. Os dados utilizados no estudo foram armazenados de forma segura e eficiente em um *bucket* do serviço S3 [2], proporcionando um ambiente de armazenamento altamente confiável e acessível para as operações de processamento e análise. Essa configuração permitiu a realização de experimentos robustos e a avaliação eficaz das técnicas de detecção de intrusão propostas.

## 4.2 Materiais

Para o desenvolvimento do método proposto, empregou-se o ambiente de desenvolvimento JupyterLab [3], uma ferramenta com interface integrada que permite a análise de dados, modelagem estatística, visualização e programação, por meio da escrita e execução de *script*. A linguagem Python na versão 3.9.16 foi a utilizada para programação das tarefas, juntamente às bibliotecas Scikit-learn 1.2.2 [4], Imblearn 0.10.1 [5], NumPy 1.25.0 [6], LightGBM 3.3.5.99 [7], Pandas 2.0.2 [8], SciPy 1.11.4 [9], Hyperopt 0.2.7 [10].

Devido ao volume de dados e a quantidade de experimentos executados, e levando em

[1] https://aws.amazon.com/
[2] https://aws.amazon.com/s3/
[3] https://jupyter.org/
[4] https://scikit-learn.org/
[5] https://imbalanced-learn.org/
[6] https://numpy.org/
[7] https://lightgbm.readthedocs.io/
[8] https://pandas.pydata.org/
[9] https://docs.scipy.org/doc/scipy/index.html
[10] http://hyperopt.github.io/hyperopt/

consideração o ganho no tempo de treinamento proporcionado pela aceleração com GPU no *framework* LightGBM, foi escolhida uma instância EC2 da Amazon do tipo g4dn.xlarge, que possui 16 GiB de memória RAM, 4 vCPUs Intel(R) Xeon(R) Platinum 8259CL e 1 GPU NVIDIA Tesla T4 com 16GiB de memória. Em complemento, um espaço de *swap* de 64GB foi configurado no servidor. As instâncias G4dn utilizam GPUs NVIDIA T4 e CPU Intel Cascade Lake personalizados, e são otimizadas para inferência de aprendizado de máquina e treinamento em pequena escala.

## 4.3 Método

Após os estudos conduzidos acerca da literatura disponível sobre segurança computacional, detecção de intrusão e aprendizado de máquina, foi possível delinear a proposta da abordagem realizada neste trabalho.

Tendo em conta a carência de estudos englobando sistemas de detecção de intrusão com aprendizado de máquina, utilizando o *dataset* CSE-CIC-IDS2018, que considerem os tipos de ataque, o presente trabalho propõe a abordagem do problema com a perspectiva de classificação multi-classe, evidenciando o efeito provocado pelo desbalanceamento dos dados na performance do classificador. Com o objetivo de oferecer soluções para mitigação de tais problemas, o método deste trabalho compreende o uso de técnicas para o tratamento do desbalanceamento, tanto no pré-processamento, quanto na fase de criação do modelo de detecção.

O método compreende a realização de tarefas de pré-processamento no *dataset* CSE-CIC-IDS2018, construção de um modelo base para comparações, experimentação com aplicação de *oversampling* às classes minoritárias, e utilização de pesos para o treinamento do modelo LightGBM e hiperparâmetros do algoritmo, de forma sequencial. Ao final de cada etapa são coletados os resultados e métricas, para avaliação. A Figura 4.1 retrata as principais tarefas do método proposto, que serão apresentadas em detalhe ao longo das próximas seções.

## 4.4 Pré-processamento

Dentre as atividades envolvidas em tarefas de aprendizado de máquina, o pré-processamento frequentemente é destacado como o que mais envolve esforços. Para a utilização da base de eventos de segurança escolhida, foram necessários diversos ajustes e adequações antes da submissão para treinamento do algoritmo. Esta seção descreve as tarefas realizadas compondo o pré-processamento do método aplicado.

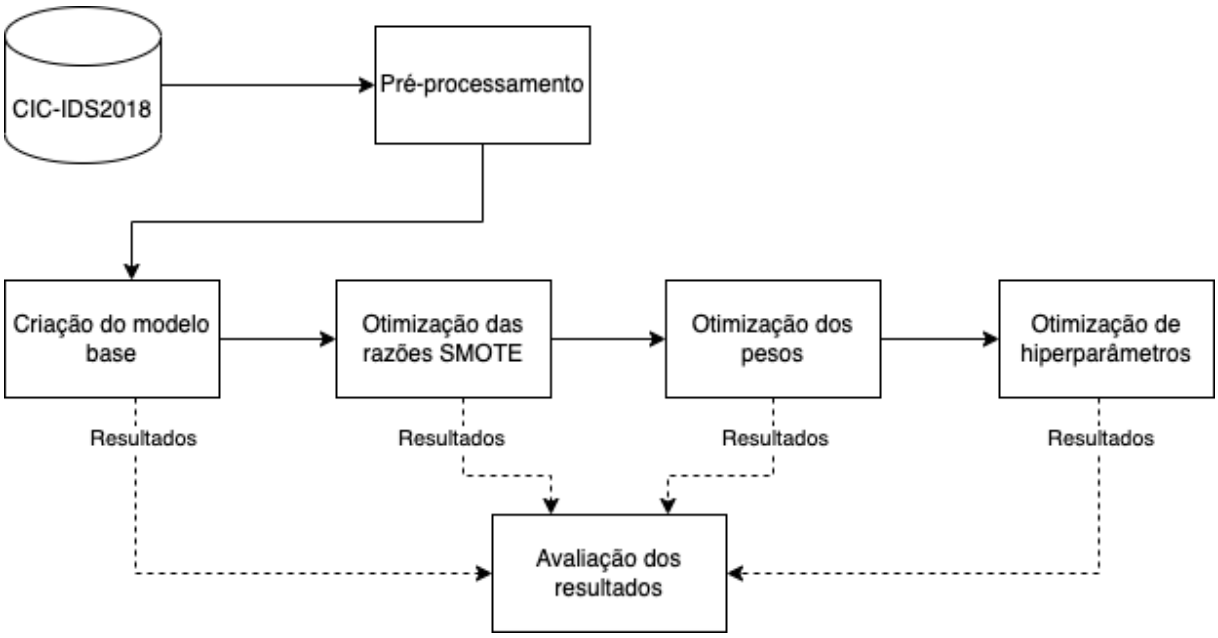


**Figura 4.1:** Método experimental da proposta.
Fonte: Autor

### 4.4.1 Coleta dos Dados

A base de eventos escolhida para a condução deste trabalho, CSE-CIC-IDS2018, está disponível em um *bucket* do serviço S3[11], organizada em dois diretórios distintos. As capturas de tráfego de rede e *logs* não manipulados, estão disponíveis no diretório *Original Network Traffic and Log data*, podendo servir como insumo para a extração dos atributos. A geração de tráfego de rede de comportamento normal, bem como dos ataques, foram conduzidos ao longo de vários dias. Os arquivos de capturas estão individualizados, de acordo com a data.

O *dataset* baseado no tráfego processado, preparado para a utilização em tarefas de aprendizado de máquina, está disponível no diretório *Processed Traffic Data for ML Algorithms*. Do mesmo modo que as capturas de tráfego e *logs*, estes arquivos processados, que possuem atributos extraídos pela ferramenta CICFlowMeter[12], encontram-se identificados pelas datas e possuem formato CSV, utilizando vírgula (,) como separador de colunas. A Tabela 4.1 apresenta os arquivos disponíveis.

Os dados foram obtidos por meio da ferramenta de gerenciamento via linha de comando da AWS [13], especificando a mesma região em que se encontra a instância EC2 utilizada, trazendo agilidade ao processo de obtenção dos dados, que possuem um volume significativo de 6.4GB.

Durante a exploração dos dados, foi identificado que um dos arquivos, *Thuesday-20-*

[11]https://aws.amazon.com/s3/
[12]https://github.com/ahlashkari/CICFlowMeter
[13]https://aws.amazon.com/cli/

**Tabela 4.1:** Arquivos que compõem o *dataset* CIC-IDS2018.

| *Arquivo* |
|---|
| Friday-02-03-2018_TrafficForML_CICFlowMeter.csv |
| Friday-16-02-2018_TrafficForML_CICFlowMeter.csv |
| Friday-23-02-2018_TrafficForML_CICFlowMeter.csv |
| Thuesday-20-02-2018_TrafficForML_CICFlowMeter.csv |
| Thursday-01-03-2018_TrafficForML_CICFlowMeter.csv |
| Thursday-15-02-2018_TrafficForML_CICFlowMeter.csv |
| Thursday-22-02-2018_TrafficForML_CICFlowMeter.csv |
| Wednesday-14-02-2018_TrafficForML_CICFlowMeter.csv |
| Wednesday-21-02-2018_TrafficForML_CICFlowMeter.csv |
| Wednesday-28-02-2018_TrafficForML_CICFlowMeter.csv |

*02-2018_TrafficForML_CICFlowMeter.csv*, possui estrutura distinta dos demais, contendo 4 colunas adicionais: *Flow ID*, *Src IP*, *Src Port*, *Dst IP*. Para a padronização, estas colunas foram eliminadas.

Outro problema encontrado nos dados foi a existência de arquivos com repetição dos cabeçalhos. A fim de evitar problemas com dados inválidos, as respectivas 59 instâncias foram removidas.

Com o objetivo de facilitar a manipulação dos dados, os arquivos foram concatenados em um único. Para tanto, a biblioteca Pandas [14] foi utilizada. Observou-se que a inferência dos tipos de dados realizada pela biblioteca Pandas poderia ser melhorada, buscando redução no uso de memória, devido ao volume elevado de dados. Assim, foi estabelecida tipagem explícita aos atributos, de acordo com o apresentado na Tabela 4.2.

Finalmente, o conteúdo foi gravado em um arquivo no formato de armazenamento colunar Apache Parquet [15] devido à sua eficiência e compatibilidade.

### 4.4.2 Limpeza dos Dados

A segunda etapa do pré-processamento do método proposto, consiste na limpeza dos dados, visando garantir a robustez e confiabilidade do algoritmo de aprendizado. O *dataset* utilizado possui pontos de atenção, como valores inválidos e faltantes.

A primeira modificação efetuada foi a remoção de atributos com valores constantes, que não agregariam ao modelo, além de impactar no tempo de execução. Os atributos removidos estão listados na Tabela 4.3

O atributo *Timestamp*, que identifica a data e hora de início do fluxo, possui muitos valores distintos, se aproximando de um identificador único as instâncias. Além disso, devido ao modo

[14] https://pandas.pydata.org
[15] https://parquet.apache.org/

**Tabela 4.2:** Tipagem definida para a manipulação do *dataset* CIC-IDS2018 com Pandas.

| *Tipo de dados* | *Atributos* |
|---|---|
| pa.int8() | Protocol |
| pa.int16() | Tot Fwd Pkts, Tot Bwd Pkts, Fwd Pkt Len Min, Bwd Pkt Len Min, Down/Up Ratio |
| pa.int32() | Dst Port, Flow Duration, TotLen Fwd Pkts, TotLen Bwd Pkts, Fwd Pkt Len Max, Bwd Pkt Len Max, Fwd Byts/b Avg, Fwd Pkts/b Avg, Fwd Blk Rate Avg, Bwd Byts/b Avg, Bwd Pkts/b Avg, Bwd Blk Rate Avg, Subflow Fwd Pkts, Subflow Fwd Byts, Subflow Bwd Pkts, Subflow Bwd Byts, Init Fwd Win Byts, Init Bwd Win Byts, Fwd Act Data Pkts, Fwd Seg Size Min |
| pa.float32() | Fwd Pkt Len Mean, Fwd Pkt Len Std, Bwd Pkt Len Mean, Bwd Pkt Len Std, Fwd IAT Tot, Fwd IAT Mean, Fwd IAT Std, Fwd IAT Max, Fwd IAT Min, Bwd IAT Tot, Bwd IAT Mean, Bwd IAT Std, Bwd IAT Max, Bwd IAT Min, Fwd Header Len, Bwd Header Len, Fwd Pkts/s, Bwd Pkts/s, Pkt Len Min, Pkt Len Max, Pkt Len Mean, Pkt Len Std, Pkt Len Var, Pkt Size Avg, Fwd Seg Size Avg, Bwd Seg Size Avg, Active Mean, Active Std, Active Max, Active Min, Idle Mean, Idle Std, Idle Max, Idle Min |
| pa.float64() | Flow Byts/s, Flow Pkts/s, Flow IAT Mean, Flow IAT Std, Flow IAT Max, Flow IAT Min |
| pa._bool() | Fwd PSH Flags, Bwd PSH Flags, Fwd URG Flags, Bwd URG Flags, FIN Flag Cnt, SYN Flag Cnt, RST Flag Cnt, PSH Flag Cnt, ACK Flag Cnt, URG Flag Cnt, CWE Flag Count, ECE Flag Cnt |
| pa.string() | Timestamp |
| pa.dictionary(pa.int8(), pa.utf8()) | Label |

**Tabela 4.3:** Atributos com valor constante no *dataset* CIC-IDS2018.

| *Atributo* |
|---|
| Bwd PSH Flags |
| Bwd URG Flags |
| Fwd Byts/b Avg |
| Fwd Pkts/b Avg |
| Fwd Blk Rate Avg |
| Bwd Byts/b Avg |
| Bwd Pkts/b Avg |
| Bwd Blk Rate Avg |

que o tráfego capturado foi gerado, com a execução de geradores de tráfego normal e ataques em datas e horários arbitrários, eventuais padrões de horários ou datas identificados nestes dados, não seriam encontrados em dados reais. Assim, o mesmo foi removido dos dados.

Os valores inválidos *NaN* e *Infinty* encontrados nas colunas *Flow Byts/s* e *Flow Pkts/s*, foram substituídos por 0.

Também foram removidos 15 registros que possuíam valores negativos para o atributo *Flow IAT Min*, pois nos mesmos registros foi observada a ocorrência de valores extremos em outros atributos. Além disso, as marcas de tempo dos respectivos registros não correspondem às datas em que os experimentos foram conduzidos, as portas e protocolos possuem valor zero, assim como diversos outros atributos, indicando que estes exemplos devem se tratar de erros. Por fim, todos os registros com esta característica são da classe *Benign*, que possui a maior representatividade do *dataset*.

### 4.4.3 Transformação dos Dados

Os arquivos fornecidos para tarefas de aprendizado de máquina possuem 15 classes distintas, associadas ao comportamento benigno, além dos diversos ataques efetuados, com a classe identificando a ferramenta ou tipo de ataque executado. Após a coleta e limpeza, uma análise da distribuição de instâncias por classe, apresentada na Tabela 4.4, revelou o desbalanceamento presente no *dataset*, onde pode ser observada a baixa representatividade de 8 classes, que possuem menos de 1% dos exemplos.

**Tabela 4.4:** Exemplos por classe do *dataset* CIC-IDS2018.

| *Classe* | *Instâncias* | *Representatividade* |
|---|---|---|
| Benign | 13484708 | 83,070% |
| DDOS attack-HOIC | 686012 | 4,226% |
| DDoS attacks-LOIC-HTTP | 576191 | 3,550% |
| DoS attacks-Hulk | 461912 | 2,846% |
| Bot | 286191 | 1,763% |
| FTP-BruteForce | 193360 | 1,191% |
| SSH-Bruteforce | 187589 | 1,156% |
| Infilteration | 161934 | 0,998% |
| DoS attacks-SlowHTTPTest | 139890 | 0,862% |
| DoS attacks-GoldenEye | 41508 | 0,256% |
| DoS attacks-Slowloris | 10990 | 0,068% |
| DDOS attack-LOIC-UDP | 1730 | 0,011% |
| Brute Force -Web | 611 | 0,004% |
| Brute Force -XSS | 230 | 0,001% |
| SQL Injection | 87 | 0,001% |

Segundo os desenvolvedores do *dataset*, os ataques foram executados em diversos cenários, de acordo com seus tipos. Por vezes, as ferramentas de ataque são evoluções de outras, como LOIC e HOIC, podendo possuir características e comportamentos semelhantes. Também com o objetivo de reduzir o desbalanceamento verificado, foi efetuado um agrupamento de classes, de acordo com os cenários de ataque, assim como sugerido pelos autores. Após o agrupamento, o número de classes foi reduzido a 7, bem como o número de classes com baixa representatividade, que passaram a 2. A Tabela 4.5 demonstra as substituições realizadas, bem

como os novos quantitativos de instâncias e representatividade.

**Tabela 4.5:** Agrupamento das classes do *dataset* CIC-IDS2018.

| *Nova classe* | *Classes prévias* | *Instâncias* | *Rep.* |
|---|---|---|---|
| Benign | Benign | 13484708 | 83,070% |
| DDoS | DDOS attack-HOIC, DDoS attacks-LOIC-HTTP, DDOS attack-LOIC-UDP | 1263933 | 7,786% |
| DoS | DoS attacks-Hulk, DoS attacks-SlowHTTPTest, DoS attacks-GoldenEye, DoS attacks-Slowloris | 654300 | 4,031% |
| BruteForce | FTP-BruteForce, SSH-Bruteforce | 380949 | 2,347% |
| BotNet | Bot | 286191 | 1,763% |
| Infiltration | Infilteration | 161934 | 0,998% |
| WebAttack | Brute Force -Web, Brute Force -XSS, SQL Injection | 928 | 0,006% |

Devido à maneira como as ferramentas utilizadas aceitam como entrada os dados para manipulação e processamento, com a separação entre os atributos e a classe, o conjunto de dados foi escrito em dois arquivos Parquet distintos, com a classe, ou atributo *Label* no arquivo *y-cicids18.parquet*, e os demais no arquivo *X-cicids18.parquet*.

A seleção dos atributos foi realizada de acordo o estabelecido por Leevy et al. (2021), devido aos resultados apresentados, em que não houve perda significativa na performance do modelo com a utilização dos 15 principais atributos selecionados, em comparação ao uso do conjunto de dados completo. Os atributos que foram mantidos estão apresentados na Tabela 4.6.

O atributo *Dst Port* passou por uma transformação, mantendo os valores originais somente para as portas conhecidas, até 1023. Para as portas registradas, entre 1024 e 49151, os valores foram substituídos por 1024, e para as portas dinâmicas e privadas, acima de 49152, os valores foram substituídos por 49152. Por fim, para a interpretação adequada como um atributo categórico pelo LightGBM, foi realizada uma transformação no tipo de dados desta coluna no *DataFrame*, configurando-a com o tipo *category*.

O conjunto de dados passou pelo processo de *undersampling*, reduzindo a quantidade de instâncias da classe *Benign* a 3 milhões. Para tanto, foi utilizada a ferramenta RandomUnderSampler, da biblioteca imblearn, com o argumento *sampling_strategy* contendo um dicionário de somente um item, que especifica a literal *Benign* como chave e 3000000 como valor. Ademais, para possibilitar a reprodução dos experimentos, assim como em todas ferramentas utilizadas em que há obtenção de valores aleatórios, foi utilizado o parâmetro *random_state*, com o valor 1121218.

Na Tabela 4.7 são apresentadas as classes, seus respectivos quantitativos de instâncias e representatividade após a transformação. Deste modo, após o *undersampling* apenas a classe *WebAttack* permanece com baixa representatividade indicando alto desbalanceamento, com menos de 1% dos exemplos.

**Tabela 4.6:** Atributos selecionados.

| *Atributo* |
|---|
| Fwd Pkt Len Mean |
| Fwd Pkt Len Max |
| Flow IAT Mean |
| TotLen Fwd Pkts |
| Bwd Pkts/s |
| Fwd Pkts/s |
| Flow Byts/s |
| Fwd IAT Max |
| Fwd IAT Tot |
| Flow IAT Std |
| Flow IAT Max |
| Dst Port |
| Fwd Seg Size Min |
| Flow Pkts/s |
| Fwd Header Len |

**Tabela 4.7:** Representatividade das classes após *undersampling*.

| *Classe* | *Instâncias* | *Rep.* |
|---|---|---|
| Benign | 3000000 | 52,190% |
| DDoS | 1263933 | 21,988% |
| DoS | 654300 | 11,383% |
| BruteForce | 380949 | 6,627% |
| BotNet | 286191 | 4,979% |
| Infiltration | 161934 | 2,817% |
| WebAttack | 928 | 0,016% |

Experimentos preliminares foram conduzidos com o conjunto de dados pré-processado, a fim de obter uma referência dos resultados para melhoria. Além do balanceamento por amostragem conduzido na classe majoritária, também foi explorada sua utilização com *oversampling* das classes minoritárias Infiltration e WebAttack, por meio do uso da técnica SMOTE.

Esta técnica consiste na geração de novos exemplos sintéticos, trazendo maior robustez em relação à replicação de registros utilizada na técnica de *oversampling* tradicional. Com a utilização do algoritmo KNN (K-Nearest Neighbors) (Mucherino et al., 2009), são selecionados alguns pontos de dados próximos, em que um subconjunto aleatório destes é utilizado para a criação de novas instâncias similares, por meio de interpolação (Fernández, Garcia, Herrera & Chawla, 2018).

Essa técnica depende da definição de parâmetros, aos quais não foi possível encontrar referências na literatura para o *dataset* e algoritmo utilizado, e tampouco podem ser especificados de maneira arbitrária. Assim, considerando o volume de dados e o esforço para o processamento, foi necessário o uso de uma ferramenta para busca da melhor configuração sem a cobertura de todas combinações possíveis.

A ferramenta de otimização de hiperparâmetros Hyperopt[16] foi empregada para a exploração do espaço de busca, com o objetivo de encontrar a melhor configuração de razões de *oversampling*. Para tanto, foi empregado o algoritmo Tree-based Parzen Esimators, ou TPE, que por meio da definição de limites para cada parâmetro, o mesmo realiza uma busca dentro do espaço, otimizando uma função objetivo.

A perda definida como métrica para otimização foi a média aritmética da AUC entre todas as classes. Como a proposta busca introduzir melhorias à classificação de classes minoritárias, não foram utilizadas médias ponderadas, pois devido à baixa representatividade das classes Infiltration e WebAttack, pequenas variações não seriam detectadas para otimização. A Figura 4.2 apresenta a técnica empregada na busca das melhores razões. O espaço de busca para as melhores razões de SMOTE foi definido como especificado na Tabela 4.8.

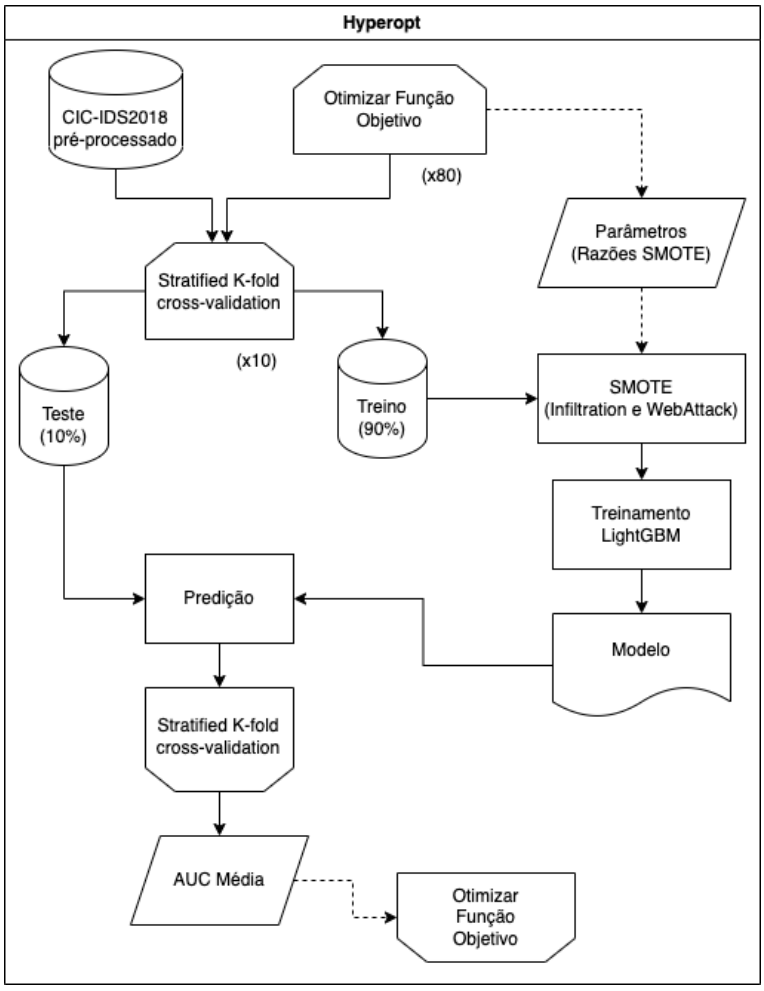


**Figura 4.2:** Otimização das razões SMOTE.
Fonte: Autor

O framework LightGBM na versão utilizada, assim como diversos algoritmos de aprendizado, possuem limitações quanto aos tipos de dados de entrada. Para as classes, originalmente identificadas por literais no *dataset*, foi necessária a aplicação de uma transformação, tornando-

[16]http://hyperopt.github.io/hyperopt/

**Tabela 4.8:** Definição do espaço de busca para otimização das razões SMOTE.

| *Classe* | *Parâmetro* | *Tipo* | *Intevalo* | *Tamanho do passo* |
|---|---|---|---|---|
| Infiltration | Razão SMOTE | Uniforme Quantizada | 1 a 10 | 1 |
| WebAttack | Razão SMOTE | Uniforme Quantizada | 1 a 100 | 1 |

as números inteiros. Para o cumprimento dessa tarefa, foi empregada a ferramenta LabelEncoder, da biblioteca scikit-learn. O mapeamento dos números para as classes foi mantido em um dicionário, a fim de facilitar a interpretação posterior dos resultados.

## 4.5 Processamento

Para possibilitar a avaliação das melhorias propostas no presente trabalho, inicialmente foi realizado o treinamento de um modelo base, sem otimizações e melhorias. Tal tarefa foi realizada com o *dataset* CIC-IDS2018 pré-processado, com exceção do uso de SMOTE, submetendo para treinamento com *10-fold cross-validation* ao LightGBM, fazendo uso apenas dos parâmetros definidos de forma estática, como na Tabela 4.9. A Figura 4.3 retrata as etapas da criação do modelo base.

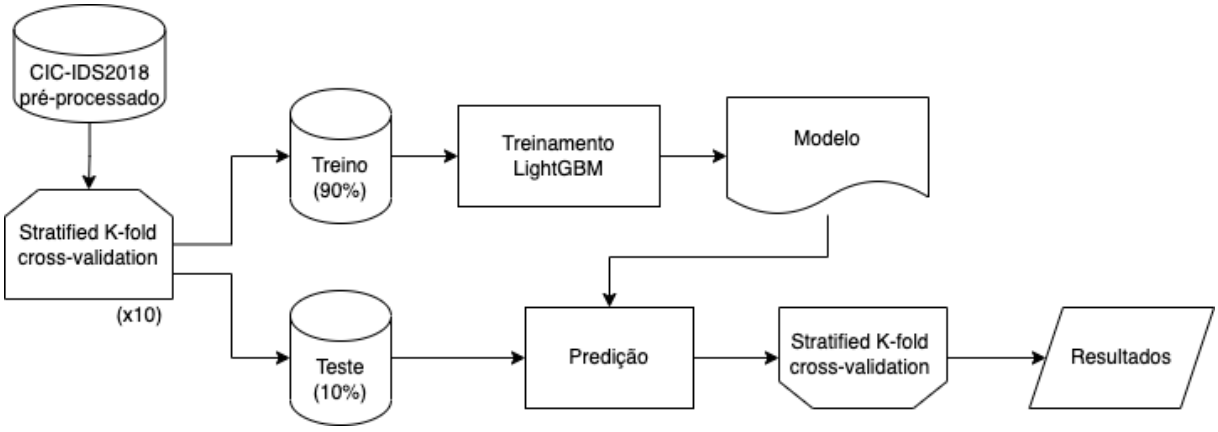


**Figura 4.3:** Criação do modelo base.
Fonte: Autor

Nesta etapa do método proposto, executaram-se os experimentos envolvendo o treinamento sensível a custo do modelo, predição, bem como a coleta de métricas para posterior avaliação. A Figura 4.4 apresenta a definição dos fluxos de processamento do método proposto, que incluem a execução de uma tarefa da fase de pré-processamento.

A interface para *scikit-learn* do LightGBM foi utilizada, por meio da classe LGBMClassifier. Além dos parâmetros que passaram por experimentação e foram fornecidos ao classificador, alguns foram definidos de maneira estática na instanciação, de acordo com o problema apresentado. Os mesmos são apresentados na Tabela 4.9

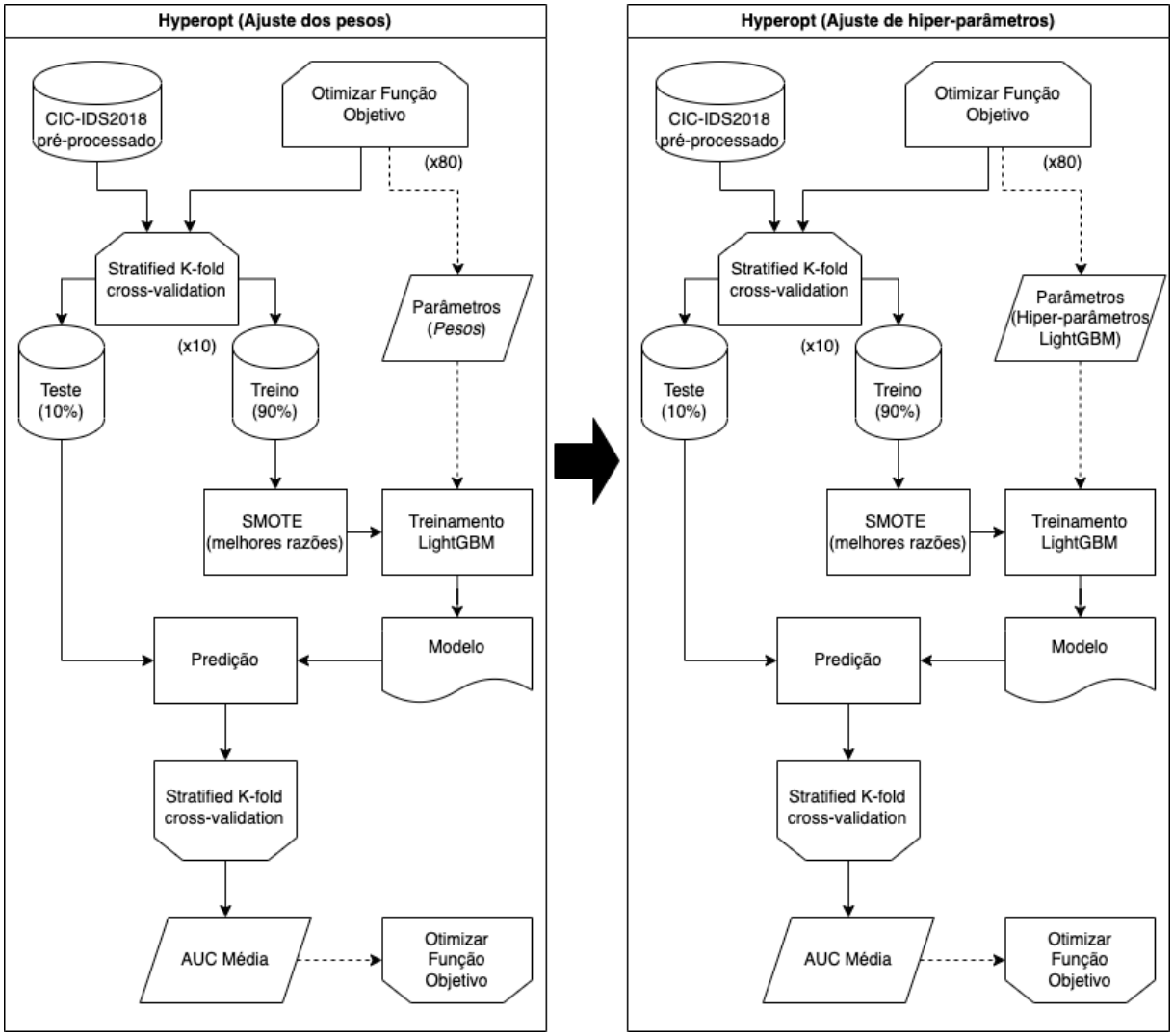


**Figura 4.4:** Processamento.
Fonte: Autor

**Tabela 4.9:** Definição dos parâmetros estáticos do LightGBM.

| *Parâmetro* | *Valor* | *Observação* |
|---|---|---|
| device_type | cuda | Uso de aceleração com CUDA |
| n_jobs | 4 | Quantidade de *threads* em paralelo. A documentação recomenda configurar com a quantidade de núcleos físicos do processador para melhor performance. |
| num_class | 7 | Quantidade de classes |
| objective | multiclassova | Tarefa de aprendizado e objetivo correspondente |
| verbosity | -1 | Desativa as mensagens de *debug* |
| seed | 1121218 | Definição do estado de sequências aleatórias, possibilitando a reprodução dos experimentos |

Além da abordagem para o tratamento de desbalanceamento aplicada na fase de transformação dos dados com as técnicas de balanceamento, durante o treinamento foram realizadas tentativas com a função de perda sensível a custo. No LightGBM, existe a possibilidade de

definição de pesos para classificação incorreta, disponíveis com o uso de um dicionário no parâmetro class_weight.

Do mesmo modo que foi realizada a otimização das razões SMOTE, a busca pela melhor configuração de pesos para o treinamento sensível a custo foi realizada com a ferramenta Hyperopt (Bergstra, Yamins & Cox, 2013). O espaço de busca foi definido de acordo com o especificado na Tabela 4.10.

**Tabela 4.10:** Definição do espaço de busca para otimização dos pesos.

| *Classe* | *Parâmetro* | *Tipo* | *Intevalo* |
|---|---|---|---|
| Infiltration | Peso | Uniforme | 1 a 5 |
| WebAttack | Peso | Uniforme | 1 a 5 |

A realização de *oversampling* e as etapas do processamento foram incluídas na função objetivo, que é executada em um processo iterativo, fornecendo distintos valores para os parâmetros de entrada, buscando convergir para um resultado ótimo da AUC média.

Buscando trazer robustez aos resultados obtidos e reduzir o viés e a variação, foi empregada a técnica de validação *k-fold cross-validation* estratificada ao *dataset* CIC-IDS2018 pré-processado, por meio da implementação StratifiedKFold da bibliteca scikit-learn.

Inicialmente, foram fornecidos ao construtor os parâmetros n_splits, especificando a quantidade de 10 divisões, bem como *shuffle* como verdadeiro, indicando que será realizado um embaralhamento dos registros de cada classe. Na sequência, por meio de chamada da função *split*, fornecendo como argumentos os *dataframes* $X$ e $y$, que correspondem aos atributos independentes e rótulos do conjunto de dados, são retornados os índices para os registros utilizados no conjunto de treinamento e validação. A chamada a essa função se repete por 10 vezes, de acordo com o número de divisões.

O *framework* definido para a criação do modelo foi o LightGBM. O mesmo conta com implementações que empregam o uso de processadores de gráficos, ou GPU, aproveitando do seu paralelismo para aumentar o poder de processamento e reduzir tempos de treinamento. Essas implementações, na versão da ferramenta utilizada neste trabalho, ainda são consideradas experimentais.

Após a utilização dos melhores parâmetros definidos em etapas prévias da experimentação, definindo as razões de SMOTE e os pesos para treinamento sensível a custo, das classes Infiltration e WebAttack, a próxima etapa consiste no ajuste de hiper-parâmetros do LightGBM.

Devido ao volume de dados e complexidade da tarefa de classificação, foram realizadas tentativas de ajuste na quantidade de iterações, que equivale à quantidade de árvores criadas pelo modelo, bem como a taxa de aprendizado, que define a velocidade da convergência do modelo. Para esta etapa, a ferramenta Hyperopt também foi utilizada, com o espaço de busca definido como apresentado na Tabela 4.11.

**Tabela 4.11:** Definição do espaço de busca para ajuste dos hiper-parâmetros.

| *Parâmetro* | *Tipo* | *Intevalo* | *Tamanho do passo* |
|---|---|---|---|
| n_estimators | Uniforme Quantizado | 100 (valor padrão) a 600 | 100 |
| learning_rate | Uniforme Quantizado | 0,01 a 0,1 (valor padrão) | 0,01 |

## 4.6 Pós-processamento

Durante a condução dos experimentos, foram coletadas as métricas AUC de cada um dos *folds*, bem como das classes, a fim de possibilitar a comparação das melhorias do método proposto em relação ao modelo de base.

Para possibilitar a afirmação de que as melhorias observadas são significativas, e não apenas devido ao acaso, foi estabelecida a aplicação de testes estatísticos, para aceitação ou refutação de algumas hipóteses, de acordo com os cenários.

Há uma série de testes estatísticos para aplicação com o objetivo de afirmar a existência de diferenças entre os valores obtidos. Para definição do teste adequado à situação, é necessário estabelecer algumas características, incluindo normalidade da distribuição da amostra, tipo dos dados, presença de paridade e dependência nos dados.

Os dados utilizados nos testes são as AUC obtidas no processo de *cross-validation*. Algumas características podem ser estabelecidas por meio de observação, como as de que os mesmos são numéricos, não pareados, pois não possuem vínculo, e não dependentes.

Para o estabelecimento de que a amostra segue uma distribuição normal, é necessária a aplicação do teste de normalidade de Shapiro-Wilk (SHAPIRO & WILK, 1965). A implementação da biblioteca SciPy [17] foi escolhida para aplicação deste teste. Assim, duas hipóteses foram formuladas:

- $H_0$: A hipótese nula estabelece que a amostra possui distribuição normal;
- $H_1$: A hipótese alternativa estabelece que a amostra não possui distribuição normal.

O nível de confiança de 95%, ou significância de 5%, amplamente utilizado em trabalhos de temas correlatos, foi o adotado no presente trabalho. Assim, para interpretar os resultados do teste, pode-se considerar que se o p-valor obtido seja inferior à significância, ou $p - valor < 0,05$, há uma probabilidade baixa de que a população segue uma distribuição normal. Isto pode ser considerado como evidência contra a hipótese nula, e favorável à hipótese alternativa.

Outro requisito para determinação do teste de diferenças a ser aplicado, é a homogeneidade de variâncias entre as amostras dos grupos. Para tal afirmação, o teste de igualdade de variâncias de Levene (Levene, 1961) pode ser aplicado. Neste trabalho, a implementação disponível na biblioteca SciPy foi utilizada. As hipóteses estabelecidas para a condução do teste

[17] https://docs.scipy.org/doc/scipy/index.html

são:

- $H_0$: As amostras dos grupos possuem variâncias homogêneas;
- $H_1$: As amostras dos grupos possuem variâncias distintas.

O resultado com estes testes, determina qual será conduzido para determinação de que existem diferenças significativas entre os resultados obtidos nos experimentos. Caso sejam observadas distribuições normais e as variâncias sejam iguais, a avaliação pode ser realizada por meio do teste paramétrico Análise de Variância (ANOVA) (Fisher, 1925). No caso da violação de qualquer um destes requisitos, o teste não paramétrico adequado é o de Kruskal-Wallis (Kruskal & Wallis, 1952). Para ambos casos, existem implementações disponíveis na biblioteca SciPy.

Do mesmo modo que para os testes de normalidade e homogeneidade de variâncias, também foram estabelecidas hipóteses para os testes de Kruskal-Wallis e ANOVA, considerando que cada grupo equivale ao método sujeito a comparação:

- $H_0$: As médias de AUC de ambos grupos possuem distribuição igual;
- $H_1$: As médias de AUC de ao menos um dos grupos possui distribuição diferente.

# Capítulo 5

# Resultados e Discussão

## 5.1 Pré-processamento

### 5.1.1 Coleta dos dados

Levando em consideração a constante evolução e mutação dos ataques cibernéticos, o CIC-IDS2018 foi o conjunto de dados escolhido para a condução do trabalho, pois está entre os mais atualizados disponíveis publicamente, o que também é fundamental para a possibilidade de replicação e extensão deste trabalho, além de comparação com outros estudos.

Algumas tarefas da coleta de dados, que implicavam na manipulação de diversos arquivos com os dados brutos, foram intensivas no consumo de memória, o que por vezes ocasionava erros no serviço Jupyter, provocando seu reinício, ou do *kernel* do *notebook* em execução. Para evitar a necessidade de utilização de uma instância de maior capacidade, um espaço adicional para *swap* de 64GB foi configurado no sistema operacional.

### 5.1.2 Limpeza dos dados

As colunas *Flow Byts/s* e *Flow Pkts/s* trazem valores inválidos, como *NaN* e *Infinity* para 95760 registros. Uma das abordagens, frequentemente aplicada, é a remoção dos registros contendo tais valores, principalmente considerando a grande quantidade de exemplos disponíveis no conjunto de dados. Avaliando em maior granularidade, foi observado que destes, 94459 são da classe *Benign*, 1295 da classe *Infilteration* e 6 da classe *FTP-BruteForce*. Como este trabalho aborda o problema do desbalanceamento, foi buscada uma alternativa para manter estes exemplos, tendo em vista que existem registros de uma classe com baixa representatividade, como apresentado anteriormente.

Buscando compreender o motivo dos valores inválidos somente nesses atributos, foi verificado que de acordo com sua especificação, vide Tabela 2.1, eles consistem em taxas por segundo, com indicativos de que são calculadas com base no atributo *Flow Duration*. Deste

modo, foi verificado que os mesmos exemplos que possuíam os atributos com valores inválidos, também guardavam o valor zero no atributo *Flow Duration*. Assim, adotou-se a estratégia de substituir os valores de ambas colunas por zero nestas instâncias.

### 5.1.3 Transformação dos dados

Em vista dos valores apresentados para os atributos *Init Fwd Win Byts* e *Init Bwd Win Byts*, que em uma parcela relevante dos exemplos apresentam valor -1, alguns autores optaram pela remoção dos mesmos antes de qualquer processo de seleção de atributos. O contexto de ambos atributos está restrito a fluxos com o protocolo TCP. Deste modo, por meio da análise exploratória conduzida neste trabalho, foi identificado que o atributo *Init Fwd Win Byts* possui valores positivos para instâncias com *Protocol* 6, e valor -1 para as instâncias com *Protocol* 0 e 17. Deste modo, foi considerado que o valor 6 representa o protocolo TCP. Avaliando também o conteúdo do atributo *Dst Port*, que possui valor 0 para todos registros com *Protocol* 0, foi concluído que o provável é que represente protocolo ICMP, que não utiliza o conceito de portas. Por fim, foi considerado que o *Protocol* 17 representa o protocolo UDP. Quanto ao atributo *Init Bwd Win Byts*, também pode ser observada a presença de valores -1 em fluxos com o protocolo TCP. Entretanto, isso pode ser explicado pela ausência de tráfego no sentido *backward*, o que pode ser verificado observando que o atributo *Tot Bwd Pkts* dos registros correspondentes possui valor 0.

Apesar destas constatações, o contexto deste trabalho não incluiu a realização de experimentos com seleção de atributos, escolhendo como solução a técnica robusta construída por (Leevy et al., 2021), que não inclui os atributos supramencionados. Tal método, consistiu na construção de um *ensemble* com combinação por votação, utilizando tanto técnicas baseadas em medidas estatísticas, quanto algoritmos de aprendizado.

Dentre os métodos que compõem o *ensemble* proposto pelos autores, 3 (três) são baseadas em *ranking* e filtro, compreendendo informação mútua, taxa de ganho e estatística qui-quadrado. Para cada uma das técnicas, os atributos foram classificados em ordem decrescente de acordo com as importâncias obtidas, mantendo os 20 melhores e eliminando importâncias iguais a 0. Outras 4 técnicas baseadas em algoritmos de aprendizado supervisionado foram empregadas, incluindo Random Forest, CatBoost, XGBoost e LightGBM. Após o ajuste dos modelos, foram coletadas as importâncias estabelecidas por cada algoritmo para os atributos. Por fim, para a definição dos atributos selecionados para a composição do grupo, foram escolhidos os que estiveram presentes em ao menos 4 das 7 técnicas aplicadas. A figura 5.1 ilustra o funcionamento do método de seleção de atributos que produziu o conjunto de atributos utilizado neste trabalho.

Considerando que o atributo *Dst Port* é categórico, e que nem todas técnicas tem capacidade de manipular esse tipo de dados sem codificação, o mesmo teria menores chances de

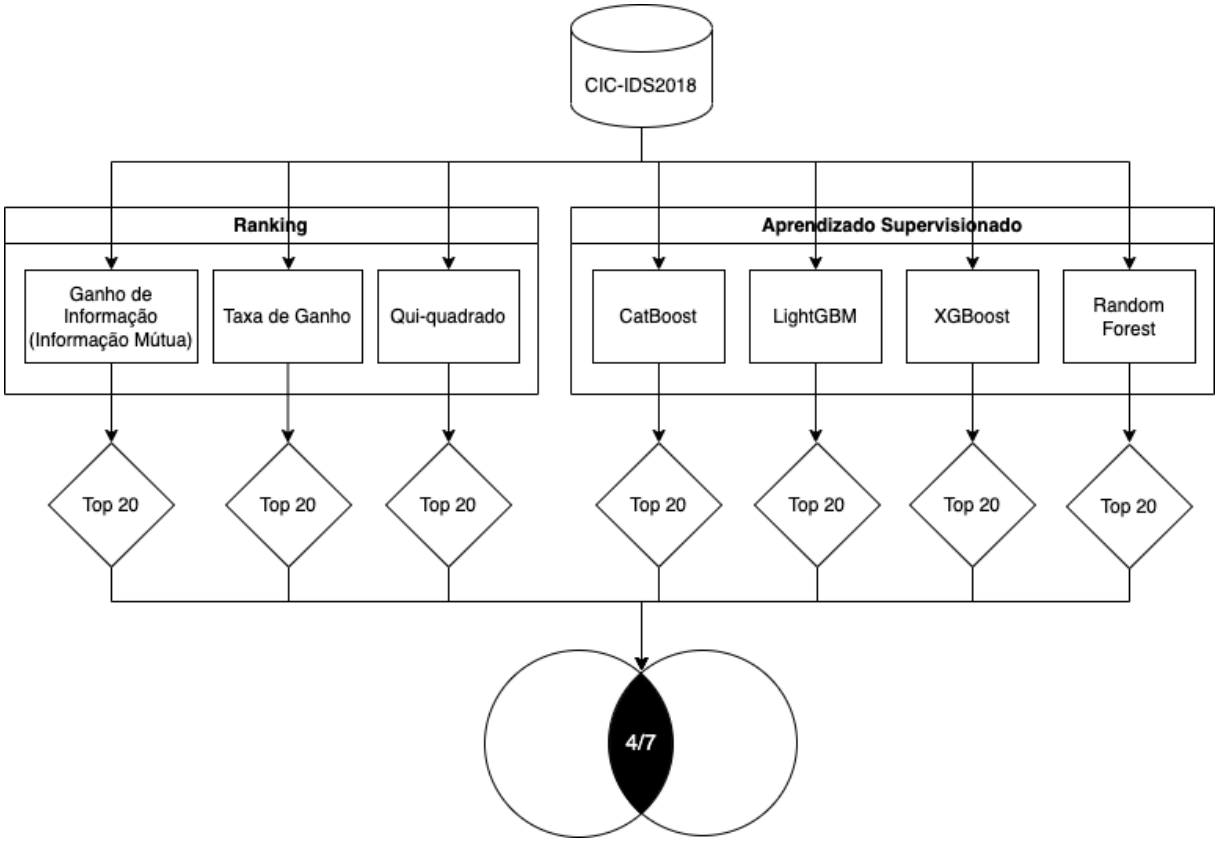


**Figura 5.1:** Seleção de atributos.
Fonte: Autor

ser selecionado. Assim, os autores criaram uma variação do grupo de atributos selecionados, arbitrariamente incluindo-o. Por ser o grupo que apresentou menor perda de performance em comparação ao uso do *dataset* com todos atributos, optou-se por manter o atributo *Dst Port* entre os selecionados. Como o algoritmo LightGBM tem suporte a atributos categóricos, este foi o grupo de atributos selecionados para o presente trabalho.

Devido aos recursos disponíveis para a execução dos experimentos, não foi possível a utilização do atributo *Dst Port* com sua estrutura original. Com o objetivo de manter a maior parte possível da informação disponível no mesmo, devido à sua elevada importância (Chimphlee & Chimphlee, 2023a; Leevy et al., 2021), foi realizada uma transformação, mantendo os valores para portas conhecidas, e agrupando as demais em dois valores, para as dinâmicas e privadas.

No trabalho de Hua (2020), em que o desbalanceamento do *dataset* CIC-IDS2018 foi abordado, houveram experimentações avaliando os efeitos provocados pelo *undersampling* da classe majoritária. Partindo de um subconjunto com 70% das instâncias, usado para treinamento, os autores aplicaram *undersampling* aleatório, reduzindo a quantidade de exemplos da classe *Benign* até 1 milhão. Com a coleta do tempo de treinamento, tempo para predição e acurácia, os autores puderam concluir que com a redução até aproximadamente 3 milhões de instâncias da classe *Benign*, não foi percebida redução significativa na acurácia, enquanto os tempos sofreram grande redução. Reduções maiores provocaram queda expressiva na acurácia

do modelo.

Assim, com o objetivo de auxiliar na diminuição do desbalanceamento do CIC-IDS2018, além de possibilitar redução no custo computacional para treinamento e predição, o método empregado neste trabalho realizou *undersampling* da classe *Benign* com a mesma razão utilizada pelos autores, de 22.25%, mantendo 3 milhões de instâncias.

### 5.1.4 Construção do modelo base

O modelo base foi construído com o objetivo de possibilitar comparações posteriores às melhorias propostas no método deste trabalho. Na Tabela 5.1 são apresentadas as métricas AUC para cada um dos *folds*. As mesmas estão organizadas por classe, e foram computadas com o método um-versus-outros. Na última coluna, é exibida a média aritmética das AUCs do modelo gerado em cada *fold*, a qual foi priorizada em relação à média ponderada, para não ocultar a influência das classes minoritárias no resultado. Nas últimas linhas, são apresentadas as médias ($\bar{x}$) e desvio padrão ($\sigma$) entre todos os *folds*.

**Tabela 5.1:** AUCs do modelo base.

| *Fold* | *Benign* | *BotNet* | *BruteForce* | *DDoS* | *DoS* | *Infiltration* | *WebAttack* | *Média* |
|---|---|---|---|---|---|---|---|---|
| 1 | 98,785% | 99,996% | 99,766% | 99,977% | 99,115% | 89,725% | 52,568% | 91,419% |
| 2 | 98,782% | 100,000% | 99,781% | 99,975% | 99,842% | 89,521% | 66,481% | 93,483% |
| 3 | 98,730% | 99,993% | 99,418% | 99,975% | 99,291% | 89,637% | 65,309% | 93,193% |
| 4 | 98,811% | 99,996% | 99,794% | 99,974% | 99,722% | 89,915% | 53,822% | 91,719% |
| 5 | 98,692% | 99,996% | 99,801% | 99,976% | 98,129% | 89,762% | 58,397% | 92,107% |
| 6 | 98,761% | 99,975% | 99,780% | 99,976% | 99,300% | 89,527% | 51,388% | 91,244% |
| 7 | 98,825% | 100,000% | 99,499% | 99,969% | 99,803% | 89,895% | 53,585% | 91,654% |
| 8 | 98,777% | 99,996% | 99,787% | 99,968% | 99,833% | 89,805% | 52,746% | 91,559% |
| 9 | 98,798% | 99,996% | 99,791% | 99,976% | 98,931% | 89,885% | 49,068% | 90,921% |
| 10 | 98,792% | 99,996% | 99,796% | 99,975% | 99,381% | 89,922% | 58,479% | 92,334% |
| $\bar{x}$ | 98,775% | 99,995% | 99,721% | 99,974% | 99,335% | 89,759% | 56,184% | 91,963% |
| $\sigma$ | 0,039% | 0,007% | 0,140% | 0,003% | 0,532% | 0,154% | 5,872% | 0,830% |

Pode ser observado que o modelo base apresentou baixa performance nas classes Infiltration e WebAttack, em que as médias de AUC foram 89,759% e 56,184%, com desvios padrão de 0,154% e 5,872%, respectivamente. Considerando que a classificação aleatória dos exemplos proporcionaria uma AUC de 50%, pode-se concluir que para a classe WebAttack, o modelo possui capacidade preditiva quase nula. Tal fato auxiliou no direcionamento do foco em melhoria de performance para estas classes, sem perder de vista a possibilidade de efeitos negativos às demais.

### 5.1.5 Experimentos para definição das razões SMOTE

A tarefa de definição das razões de *oversampling* SMOTE para cada classe pode trazer um elevado custo computacional, caso as possibilidades sejam avaliadas por meio do treinamento do modelo e avaliação dos resultados.

Entre as estratégias disponíveis, a *grid search*, explora extensivamente todo o espaço de busca definido, encontrando a melhor configuração absoluta. Entretanto, devido ao custo computacional para realização de tal tarefa, a tornaria inviável. Outra alternativa conhecida, é a exploração aleatória dentro do espaço de busca, que pode se aproximar ao resultado ótimo após um número de tentativas.

Seo et al. (2018) propuseram uma abordagem alternativa, trazendo uma metodologia para aplicação de regressão com SVM, a fim de encontrar razões ótimas. Apesar dos resultados apresentados pelos autores, tal abordagem consiste em uma proposta nova, também não foram encontradas aplicações ou abordagens seguindo a mesma estratégia na literatura para validação de sua eficácia em outros casos de uso.

Assim, buscando balancear o custo computacional e a qualidade do resultado obtido na otimização, foi escolhida a abordagem Tree-structured Parzen Estimator (TPE), que consiste em um algoritmo de aprendizado de máquina para otimização Bayesiana (Bergstra et al., 2013), projetado para a exploração de espaços de busca. Seu propósito é encontrar um conjunto ótimo de hiper-parâmetros para um modelo de aprendizado de máquina. O mesmo trabalha iterativamente, construindo modelos probabilísticos de uma função objetivo, para definir os parâmetros mais promissores para a próxima avaliação, mantendo o foco em regiões com maior probabilidade de ganho do espaço de busca.

A busca pelas razões SMOTE foi conduzida para as duas classes com menor representatividade, devido à baixa performance apresentada para as mesmas no modelo base, construído sem as otimizações propostas na experimentação deste trabalho. Na Tabela 5.2 são apresentadas as métricas coletadas com as melhores razões de SMOTE encontradas, seguindo a mesma estrutura da tabela do modelo base.

A busca foi realizada ao longo de 80 iterações, porém a combinação de parâmetros ótima aconteceu na 47ª iteração. A Figura 5.2 apresenta as AUC médias obtidas ao longo da execução.

O melhor resultado encontrado foi a razão de 83 vezes a quantidade de registros para a classe WebAttack e 1 vez para a classe Infiltration, ou seja, o uso de *oversampling* SMOTE apenas provocou melhoria quando aplicado à classe WebAttack. Na Figura 5.3 podem ser visualizadas as regiões com os melhores resultados, em função das razões aplicadas a ambas as classes. Por meio desta, é possível observar que a variação na razão da classe WebAttack está associada a uma melhoria da AUC, entretanto a classe Infiltration aparenta ter pouco efeito. Isso pode ser explicado pelo fato de que a classe WebAttack possuía baixíssima representatividade, com menos de 1% das instâncias. Já a classe Infiltration, apesar de ser a segunda com menor

**Tabela 5.2:** AUCs com a aplicação de SMOTE.

| *Fold* | *Benign* | *BotNet* | *BruteForce* | *DDoS* | *DoS* | *Infiltration* | *WebAttack* | *Média* |
|---|---|---|---|---|---|---|---|---|
| 1 | 98,738% | 100,000% | 99,625% | 99,977% | 99,775% | 89,571% | 99,585% | 98,182% |
| 2 | 98,734% | 99,996% | 99,792% | 99,975% | 99,200% | 89,783% | 99,644% | 98,161% |
| 3 | 98,806% | 100,000% | 99,526% | 99,975% | 99,855% | 89,990% | 99,665% | 98,260% |
| 4 | 98,813% | 100,000% | 99,777% | 99,974% | 99,791% | 90,073% | 99,444% | 98,268% |
| 5 | 98,816% | 99,999% | 99,743% | 99,970% | 99,847% | 89,992% | 99,421% | 98,256% |
| 6 | 98,719% | 100,000% | 99,068% | 99,973% | 99,414% | 89,666% | 99,703% | 98,078% |
| 7 | 98,796% | 99,996% | 99,805% | 99,975% | 99,837% | 89,846% | 99,764% | 98,288% |
| 8 | 98,776% | 100,000% | 99,772% | 99,973% | 99,667% | 89,742% | 99,508% | 98,206% |
| 9 | 98,778% | 100,000% | 99,803% | 99,975% | 99,727% | 89,760% | 99,704% | 98,250% |
| 10 | 98,786% | 100,000% | 99,785% | 99,975% | 99,838% | 89,865% | 97,698% | 97,992% |
| $\bar{x}$ | 98,776% | 99,999% | 99,670% | 99,974% | 99,695% | 89,829% | 99,414% | 98,194% |
| $\sigma$ | 0,035% | 0,001% | 0,230% | 0,002% | 0,219% | 0,157% | 0,614% | 0,095% |

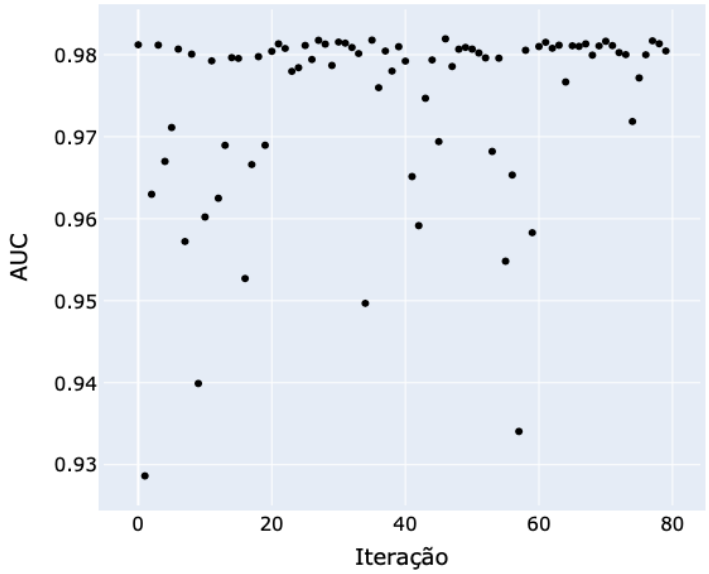


**Figura 5.2:** Evolução da otimização de razões SMOTE.
Fonte: Autor

representatividade, possui um número maior de instâncias, e a razão para baixa performance do modelo em classificá-la corretamente pode ter outros motivos.

Os resultados obtidos com a aplicação desta técnica tiveram características similares às do trabalho de (Seo et al., 2018), em que somente para algumas das classes o uso de *oversampling* proporcionou melhorias, apesar da base de dados e metodologia de otimização distintas.

O teste de Shapiro-Wilk foi aplicado para as métricas AUC médias de cada *fold*, tanto para as coletadas do modelo base, quanto para o modelo com *oversampling* SMOTE. Os resultados obtidos para $p - valor$ em ambos casos foram superiores à significância definida de 5%, o que não permite rejeitar a hipótese nula. Este resultado, bem como os demais resultados dos testes estatísticos aplicados durante os experimentos desta subseção, podem ser consultados na Tabela

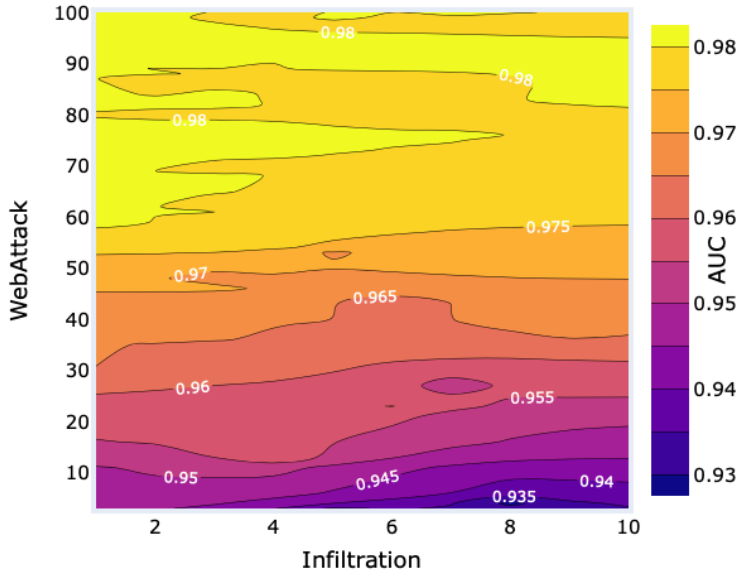


**Figura 5.3:** AUC média em função das razões SMOTE.
Fonte: Autor

**Tabela 5.3:** Testes estatísticos entre o modelo base e com SMOTE.

| *Métrica* | *Teste* | *Modelo* | $p - valor$ |
|---|---|---|---|
| AUC Média | Shapiro-Wilk | Base | 0,29217 |
| | Shapiro-Wilk | SMOTE | 0,08015 |
| | Levene | Base vs. SMOTE | 0,01284 |
| | Kruskal-Wallis | Base vs. SMOTE | 0,00016 |
| AUC Infiltration | Shapiro-Wilk | Base | 0,13700 |
| | Shapiro-Wilk | SMOTE | 0,91402 |
| | Levene | Base vs. SMOTE | 0,98446 |
| | ANOVA | Base vs. SMOTE | 0,33076 |
| AUC WebAttack | Shapiro-Wilk | Base | 0,13602 |
| | Shapiro-Wilk | SMOTE | 0,00001 |
| | Levene | Base vs. SMOTE | 0,01228 |
| | Kruskal-Wallis | Base vs. SMOTE | 0,00016 |

5.3. Outro teste aplicado para verificação de que um teste paramétrico pode ser aplicado, foi o de Levene, comparando as variâncias do grupo do modelo base e modelo com SMOTE. Para este teste, o $p - valor$ obtido foi inferior à significância, fornecendo evidência suficiente para a rejeição da hipótese $H_0$.

Assim, como constatado que os dados analisados não são paramétricos, o teste adequado para verificação de que há diferença significativa entre as médias dos grupos é o de Kruskal-Wallis. O mesmo foi aplicado, e os resultados demonstraram que a hipótese nula pode ser rejeitada, concluindo que as melhorias obtidas com a aplicação de SMOTE são estatisticamente significativas.

Cabe-se ressaltar que apesar de não serem verificadas diferenças notáveis entre as AUC

médias de cada classe com a aplicação de SMOTE, no caso da classe DoS houve uma diminuição no desvio padrão.

Testes adicionais foram realizados com as AUC obtidas nas classes Infiltration e WebAttack, para a verificação se os dados são paramétricos ou não. Para os resultados da primeira classe, foi possível aplicar o teste paramétrico ANOVA, em que não foi possível comprovar que as diferenças encontradas são estatisticamente significativas. Assim, confirma-se que o modelo com a aplicação de SMOTE, não trouxe melhorias para a classificação da classe Infiltration. Já para a classe WebAttack, os dados não são paramétricos, e o teste de Kruskal-Wallis foi aplicado, trazendo evidências que confirmam que as melhorias encontradas são significativas.

## 5.2 Processamento

Em um primeiro momento, foram realizadas tentativas de utilização da construção original do LightGBM baseada em GPU, com o uso de OpenCL. Entretanto, devido a problemas encontrados ao longo da execução do treinamento, foi utilizada a implementação alternativa baseada em CUDA [1], uma plataforma de computação paralela proprietária que permite o uso de processadores gráficos para realização de tarefas de computação tradicional. Para possibilitar sua utilização, foi necessário obter o código fonte do LightGBM, compilar e construir, além da instalação de algumas dependências no sistema hospedeiro, processo mais trabalhoso do que a instalação padrão, que pode ser realizada com simplicidade fazendo uso de ferramentas como o instalador de pacotes do Python, pip.

Durante a etapa de processamento, foram observadas algumas peculiaridades nas configurações do algoritmo LightGBM. Apesar de ser um dos principais recursos da proposta que concebeu este algoritmo, a técnica Gradient-based One-Side Sampling não pode ser aplicada com o uso de aceleração por CUDA, pois provocava o erro *LightGBMError: [CUDA] invalid argument* ao habilitá-la, por meio do parâmetro *boosting* = *goss*. Maiores detalhes sobre o erro podem ser visualizados no Apêndice A.1

É possível que esta limitação esteja associada ao desenvolvimento ainda experimental da implementação com suporte a aceleração CUDA na versão utilizada do LightGBM. Entretanto, apesar deste recurso prometer uma redução nos tempos de treinamento, foi optado por utilizar a aceleração CUDA por apresentar ganhos mais significativos nos tempos, em relação à utilização do LightGBM com GOSS e somente uso de CPU.

[1] https://developer.nvidia.com/cuda-toolkit

### 5.2.1 Experimentos para definição dos pesos das classes

A experimentação aplicando *oversampling* não foi capaz de proporcionar melhorias à performance do modelo para a classe Infiltration, como evidenciado pelos resultados apresentados anteriormente, e pelos testes estatísticos conduzidos do mesmo modo que para o modelo geral, vide Tabela 5.3. Unindo-se ao fato de que no trabalho de Tang et al. (2021), o uso de função de perda sensível a custo proporcionou melhorias ao modelo, uma estratégia similar foi adotada neste trabalho. As duas classes que tiveram os piores resultados foram escolhidas para otimização por meio da configuração de diferentes pesos.

**Tabela 5.4:** AUC com o ajuste dos pesos das classes.

| *Fold* | *Benign* | *BotNet* | *BruteForce* | *DDoS* | *DoS* | *Infiltration* | *WebAttack* | *Média* |
|---|---|---|---|---|---|---|---|---|
| 1 | 98,770% | 99,993% | 99,782% | 99,973% | 99,694% | 89,892% | 99,597% | 98,243% |
| 2 | 98,731% | 100,000% | 99,801% | 99,973% | 99,189% | 89,791% | 99,721% | 98,172% |
| 3 | 98,786% | 99,993% | 99,781% | 99,973% | 99,796% | 90,018% | 99,657% | 98,286% |
| 4 | 98,811% | 100,000% | 99,590% | 99,973% | 99,818% | 90,179% | 99,484% | 98,265% |
| 5 | 98,788% | 99,999% | 99,500% | 99,976% | 99,814% | 89,952% | 99,436% | 98,209% |
| 6 | 98,767% | 100,000% | 99,201% | 99,974% | 99,751% | 89,754% | 99,677% | 98,161% |
| 7 | 98,788% | 99,986% | 99,794% | 99,969% | 99,821% | 89,941% | 99,728% | 98,290% |
| 8 | 98,786% | 100,000% | 99,804% | 99,965% | 99,869% | 89,870% | 99,530% | 98,261% |
| 9 | 98,802% | 99,996% | 99,807% | 99,972% | 99,875% | 90,006% | 99,806% | 98,324% |
| 10 | 98,796% | 100,000% | 99,718% | 99,973% | 99,703% | 90,035% | 99,410% | 98,233% |
| $\bar{x}$ | 98,783% | 99,997% | 99,678% | 99,972% | 99,733% | 89,944% | 99,604% | 98,244% |
| $\sigma$ | 0,022% | 0,005% | 0,197% | 0,003% | 0,201% | 0,125% | 0,135% | 0,052% |

A configuração que foi capaz de produzir os melhores resultados para AUC média do modelo, foi o peso 3,02785 para a classe Infiltration e 4,131125 para WebAttack. Tais resultados são apresentados na tabela 5.4.

Foram realizadas 80 tentativas durante a exploração dos valores ótimos para o parâmetro class_weight nas classes Infiltration e WebAttack. A Figura 5.4 apresenta a evolução da AUC média do modelo em função das iterações. A otimização encontrou a melhor configuração dos parâmetros na 40ª iteração. Foi possível observar uma menor amplitude de valores para AUC, em relação à busca realizada pelas razões SMOTE.

A Figura 5.5 demonstra a variação nos resultados obtidos para a AUC média do modelo, associada aos valores utilizados para class_weight nas classes Infiltration e WebAttack. Ao contrário do observado na otimização das razões SMOTE, são pequenas as regiões que se aproximam do melhor valor encontrado para a AUC.

Os resultados obtidos com os melhores pesos e aplicação de SMOTE demonstraram uma leve melhora nos resultados para a AUC média do modelo, em relação ao modelo somente com a aplicação de SMOTE. Também houve melhora nas médias das AUC de cada classe e redução no desvio padrão, com exceção das classes BotNet e DDoS.

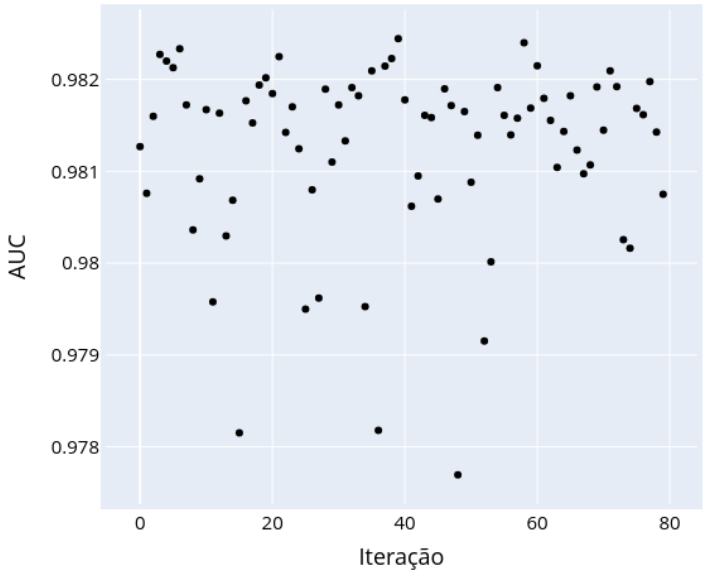


**Figura 5.4:** Evolução da otimização dos pesos.
Fonte: Autor

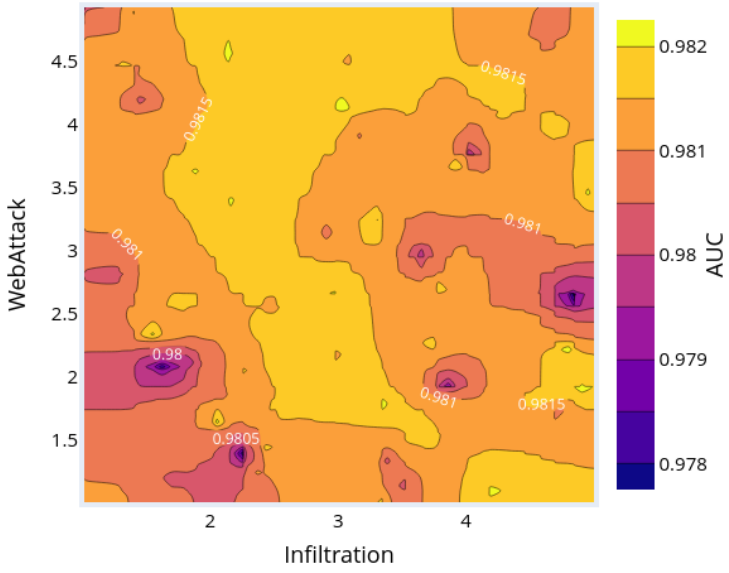


**Figura 5.5:** AUC média em função dos pesos.
Fonte: Autor

As AUC médias, bem como das classes Infiltration e WebAttack passaram pelos testes estatísticos de normalidade e igualdade de variâncias, a fim de determinar qual teste para avaliar as diferenças encontradas. As AUC médias e AUC da classe Infiltration apresentaram distribuição normal e variâncias iguais, quando comparando entre as métricas do modelo SMOTE e com pesos, enquanto as AUC da classe WebAttack não possuem distribuição normal. Os resultados dos respectivos testes estatísticos estão disponíveis na Tabela 5.5, onde o modelo com uso de SMOTE e definição de pesos é referenciado apenas como Pesos. Como os testes de normali-

dade para o modelo SMOTE já haviam sido conduzidos nos experimentos da seção anterior, seus resultados foram omitidos nessa tabela.

**Tabela 5.5:** Testes estatísticos entre o modelo com SMOTE e modelo com SMOTE e pesos.

| *Métrica* | *Teste* | *Modelo* | $p - valor$ |
|---|---|---|---|
| AUC Média | Shapiro-Wilk | Pesos | 0,83409 |
| | Levene | SMOTE vs. Pesos | 0,23072 |
| | ANOVA | SMOTE vs. Pesos | 0,15767 |
| AUC Infiltration | Shapiro-Wilk | Pesos | 0,93481 |
| | Levene | SMOTE vs. Pesos | 0,42138 |
| | ANOVA | SMOTE vs. Pesos | 0,08625 |
| AUC WebAttack | Shapiro-Wilk | Pesos | 0,67709 |
| | Levene | SMOTE vs. Pesos | 0,36963 |
| | Kruskal-Wallis | SMOTE vs. Pesos | 0,65015 |

Do mesmo modo que efetuado nos experimentos com a aplicação de SMOTE, testes de normalidade e igualdade de variâncias foram realizados para a identificação dos respectivos testes de comparação adequados, paramétricos ou não. Para as métricas AUC média e AUC da classe Infiltration, o teste paramétrico ANOVA foi aplicado, trazendo resultados acima da significância, em ambos os casos. Para a AUC da classe WebAttack, o teste não-paramétrico de Kruskal-Wallis foi aplicado, em que o $p - valor$ obtido também foi acima de 5%. Assim, não foi possível fornecer evidências de que a definição de pesos para o treinamento do modelo provocaram melhorias significativas nos resultados avaliados.

Ademais, apesar de não comprovadas melhorias significativas nos resultados, a definição dos pesos demonstrou aumento na estabilidade dos modelos durante o processo de *cross-validation*, reduzindo o desvio padrão apresentado nos resultados, principalmente na AUC da classe WebAttack e na média de todas as classes. É notável o incremento de aproximadamente 2% na AUC da classe WebAttack no *fold* 10, que apresentou o menor dos resultados antes da definição dos pesos.

### 5.2.2 Experimentos para ajuste de hiper-parâmetros

Apesar da performance de estado da arte proporcionada pelo uso de algoritmos de Gradient Boosting, como o LightGBM, a sua implementação permite a configuração de alguns hiper-parâmetros, que podem impactar no ajuste do modelo criado, sua complexidade, bem como no tempo de treinamento e classificação (Anghel, Papandreou, Parnell, Palma & Pozidis, 2018). Buscando maximizar o poder preditivo do modelo criado, foi realizada otimização de dois (2) hiper-parâmetros do algoritmo, que trazem impactos nos resultados obtidos e custo computacional.

A otimização de hiper-parâmetros foi realizada por 60 iterações, em que o melhor re-

sultado foi obtido na 35ª iteração, com o parâmetro n_estimators definido como 200 e learning_rate como 0,04, conforme evolução apresentada na Figura 5.6. Com o modelo gerado a partir da melhor configuração de hiper-parâmetros, os resultados foram próximos ao modelo criado sem especificar os parâmetros n_estimators e learning_rate, que possuem os valores padrão de $100$ e $0,1$, respectivamente. As métricas coletadas estão disponíveis na Tabela 5.6. As diferenças encontradas foram mais notáveis nas AUC de cada classe, podendo ser observadas nas médias de todos os *folds*, ao contrário da média geral, que com a escala apresentada nos dados coletados, se manteve igual ao modelo com uso de SMOTE e pesos. Ademais, o modelo com os hiper-parâmetros otimizados apresentou melhor estabilidade ao longo do processo de *cross-validation*, apresentando uma redução no desvio padrão, com exceção às classes *Benign* e *Infiltration*, em que houve um aumento pequeno.

**Tabela 5.6:** AUCs com o ajuste de hiper-parâmetros.

| *Fold* | *Benign* | *BotNet* | *BruteForce* | *DDoS* | *DoS* | *Infiltration* | *WebAttack* | *Média* |
|---|---|---|---|---|---|---|---|---|
| 1 | 98,745% | 99,993% | 99,805% | 99,975% | 99,671% | 89,704% | 99,666% | 98,223% |
| 2 | 98,748% | 100,000% | 99,795% | 99,972% | 99,876% | 89,667% | 99,735% | 98,256% |
| 3 | 98,780% | 100,000% | 99,787% | 99,973% | 99,873% | 89,855% | 99,672% | 98,277% |
| 4 | 98,792% | 99,996% | 99,801% | 99,972% | 99,874% | 90,011% | 99,476% | 98,275% |
| 5 | 98,707% | 99,995% | 99,764% | 99,972% | 99,842% | 89,708% | 99,451% | 98,206% |
| 6 | 98,739% | 100,000% | 99,795% | 99,973% | 99,843% | 89,545% | 99,693% | 98,227% |
| 7 | 98,744% | 99,996% | 99,791% | 99,970% | 99,762% | 89,695% | 99,719% | 98,240% |
| 8 | 98,733% | 100,000% | 99,795% | 99,969% | 99,834% | 89,648% | 99,517% | 98,214% |
| 9 | 98,772% | 99,996% | 99,799% | 99,973% | 99,877% | 89,775% | 99,792% | 98,284% |
| 10 | 98,781% | 100,000% | 99,788% | 99,973% | 99,862% | 89,830% | 99,454% | 98,241% |
| $\bar{x}$ | 98,754% | 99,998% | 99,792% | 99,972% | 99,832% | 89,744% | 99,618% | 98,244% |
| $\sigma$ | 0,026% | 0,003% | 0,011% | 0,002% | 0,066% | 0,130% | 0,129% | 0,028% |

A visualização disponível na Figura 5.7 permite verificar a variação da AUC média obtida, relacionando aos parâmetros em que foi conduzida a otimização. É possível observar que valores elevados para ambos os parâmetros provocam uma redução na AUC média, além de que valores extremamente baixos para os mesmos também demonstram uma região de redução da performance.

Seguindo o protocolo adotado para os demais experimentos, foram conduzidos os testes estatísticos necessários para identificar a presença ou ausência de diferenças significativas do modelo construído, em relação ao modelo com uso de SMOTE e definição de pesos. Os testes realizados, em que os resultados podem ser consultados na Tabela 5.7, direcionaram à aplicação do teste paramétrico ANOVA. Por sua vez, seus resultados demonstraram que para as AUC médias do modelo e AUC da classe WebAttack, não existem evidências que indiquem a presença de diferenças significativas para o uso da configuração ótima de hiper-parâmetros obtida. Ademais, o teste indicou que a redução observada das AUC da classe Infiltration é significativa.

Deste modo, pode-se estabelecer que é favorável a utilização dos valores padrão para ambos hiper-parâmetros. Isso justifica-se pois além de não possibilitar melhoria para a per-

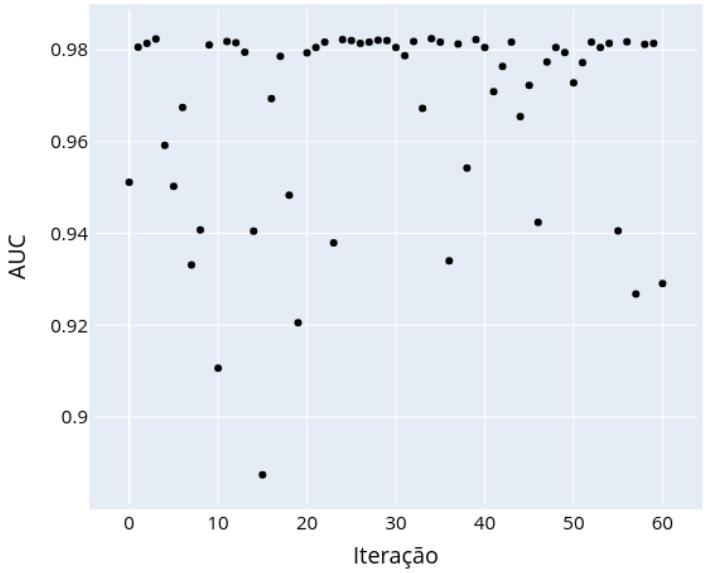


**Figura 5.6:** Evolução da otimização de hiper-parâmetros.
Fonte: Autor

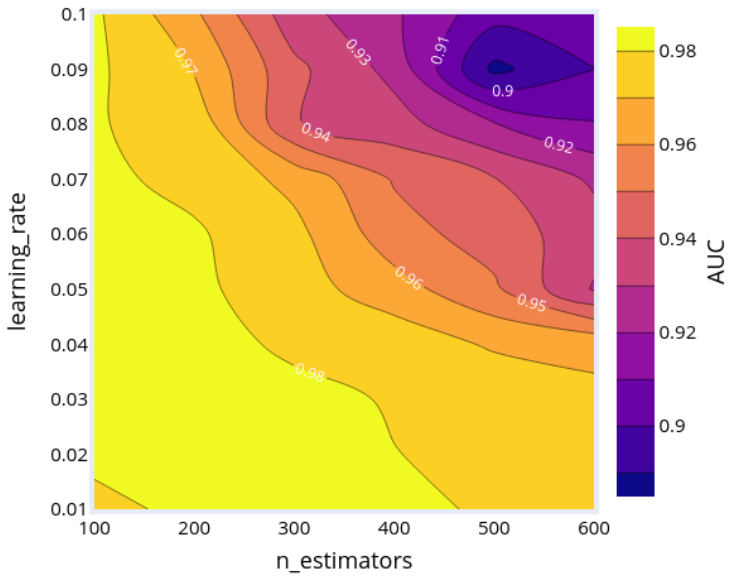


**Figura 5.7:** AUC média em função dos hiper-parâmetros.
Fonte: Autor

formance geral do modelo avaliada pela AUC, a mudança nos hiper-parâmetros provocou uma redução na AUC da classe Infiltration. Ademais, o aumento na quantidade de árvores do modelo, dada pelo parâmetro n_estimators, provoca um acréscimo na complexidade do mesmo, elevando o custo computacional das tarefas de treinamento e predição.

**Tabela 5.7:** Testes estatísticos entre AUCs médias do modelo com SMOTE e pesos e modelo com ajuste de hiper-parâmetros.

| *Métrica* | *Teste* | *Modelo* | $p - valor$ |
|---|---|---|---|
| AUC Média | Shapiro-Wilk | Hiper-parâmetros | 0,49102 |
| | Levene | Pesos vs. Hiper-parâmetros | 0,10650 |
| | ANOVA | Pesos vs. Hiper-parâmetros | 0,98949 |
| AUC Infiltration | Shapiro-Wilk | Hiper-parâmetros | 0,68656 |
| | Levene | Pesos vs. Hiper-parâmetros | 0,95175 |
| | ANOVA | Pesos vs. Hiper-parâmetros | 0,00250 |
| AUC WebAttack | Shapiro-Wilk | Hiper-parâmetros | 0,10718 |
| | Levene | Pesos vs. Hiper-parâmetros | 0,80671 |
| | ANOVA | Pesos vs. Hiper-parâmetros | 0,82761 |

## 5.3 Comparação

Para todos os efeitos, o melhor modelo resultante deste trabalho, é considerado o que envolve o uso de SMOTE para *oversampling* da classe WebAttack, com a razão de 83 vezes, bem como o treinamento utilizando a definição de pesos para as classes, com o valor 3,02785 associado à classe Infiltration e 4,131125 à classe WebAttack, sem a utilização dos hiper-parâmetros otimizados, mantendo os valores padrão de 100 para n_estimators e 0,1 para learning_rate.

A fim de permitir comparações com trabalhos relacionados, as métricas precisão, sensibilidade e AUC foram computadas de maneira distinta em relação aos experimentos realizados, com uso de médias ponderadas, de acordo com a participação de cada classe, assim como é observado nas demais publicações. Ademais, com exceção da métrica AUC, as demais foram computadas considerando uma limítrofe padrão para separação das classes. A saída da classificação produzida pelo modelo possui um formato de vetor, com os *scores* obtidos para cada uma das classes. É considerado como resultado da classificação, o maior entre estes. Os resultados podem ser observados na Tabela 5.8.

Há um elevado número de estudos utilizando os *datasets* CIC-IDS, entretanto muitos deles não dão a devida importância a aspectos como desbalanceamento e utilização de métricas que permitam avaliar sua performance com os ataques de classes minoritárias, muitas vezes apresentando classificadores com acurácias altíssimas, porém sem explorar a possibilidade de ocorrência de viés e *overfitting*.

Fitni & Ramli (2020) comparam os resultados de vários algoritmos conhecidos de aprendizado de máquina aplicados ao *dataset* CIC-IDS-2018, apresentando resultados da classificação de cada uma das classes, sem a utilização de técnicas de balanceamento. Os resultados gerais dos modelos demonstraram bons, com acurácia superior ao presente trabalho. Entretanto, as classes minoritárias possuem altas taxas de classificação incorreta. Para a classe *Infiltration*, em que o modelo do presente trabalho também demonstrou dificuldades na classificação, foram apresentadas taxas de classificação incorreta de 93,6%, o que equivale a uma acurácia baixís-

sima de 0,064%. Do mesmo modo, as classes *SQL Injection*, *Brute Force - Web* e *Brute Force - XSS*, apresentaram acurácias de 29,4%, 45,1% e 63,0%. Estas classes foram agrupadas para a composição da classe *WebAttack* neste trabalho, que também apresentou baixos resultados.

Nos resultados apresentados no trabalho de Faker & Dogdu (2019), a única métrica apresentada foi a acurácia global média do processo de *cross-validation*, sem o desvio padrão. Além disso, não foi demonstrada a realização de testes estatísticos a fim de comprovar que as diferenças identificadas sejam significativas. Do mesmo modo, outros trabalhos não levaram em consideração a necessidade de comprovação com testes estatísticos (Fitni & Ramli, 2020; Tang et al., 2021).

**Tabela 5.8:** Comparação dos resultados de trabalhos relacionados.

| *Trabalho* | *Tipo* | *Algoritmo* | *AUC* | *ACC* | *PR* | *REC* | *F1* |
|---|---|---|---|---|---|---|---|
| Kanimozhi & Jacob (2019) | Binária[2] | MLP | 0,999 | 0,999 | - | - | 0,999 |
| Chimphlee & Chimphlee (2023a) | Binária | MLP | 0,994 | 0,995 | 0,993 | 0,995 | 0,995 |
| Songma et al. (2023) | Binária[2] | XGBoost | 0,993 | 0,998 | 0,921 | 0,989 | 0,949 |
| Este trabalho | Multiclasse | LightGBM | 0,990 | 0,954 | 0,954 | 0,954 | 0,952 |
| Fitni & Ramli (2020) | Binária | Ensemble[3] | 0,941 | 0,988 | 0,988 | 0,971 | 0,979 |
| Karatas et al. (2020) | Binária[2] | LightGBM | - | 0,994 | 0,993 | 0,993 | 0,993 |
| Chimphlee & Chimphlee (2023b) | Binária | Ensemble | - | 0,984 | 0,982 | 0,984 | 0,980 |

Comparando os resultados obtidos neste trabalho com outros que utilizam o mesmo *dataset*, apenas foi possível observar uma melhor AUC em relação ao trabalho de Fitni & Ramli (2020). Entretanto, cabe ressaltar que a composição dos resultados pode ser impactada por diversas atividades no processo de mineração de dados. Até onde sabemos, em relação aos trabalhos avaliados e comparados, este foi o que levou com maior rigor o uso de técnicas para obtenção de resultados com robustez e confiabilidade, abrangendo com detalhe as tarefas de pré-processamento, processamento e pós-processamento, possibilitando a reprodução dos experimentos, bem como incorporando técnicas de validação e testes estatísticos.

Trabalhos envolvendo o *dataset* CIC-IDS-2018 ainda são escassos, e alguns entre os utilizados no comparativo de resultados possuem características que podem limitar a comparação. Tais diferenças e deficiências encontradas nos trabalhos correlatos são discutidas nos próximos parágrafos.

Alguns autores optaram por não utilizar o *dataset* CSE-CIC-IDS2018 com todas as instâncias disponíveis. Songma et al. (2023) limitaram o trabalho à classificação de ataques do tipo DoS e DDoS, justificando com a sua forma resistente e de mitigação difícil. Karatas et al. (2020) utilizaram apenas uma parte das classes disponíveis, compreendendo *Benign*, *Bot*, *Brute Force*, *DoS*, *Infiltration* e *SQL injection*, em um total de aproximadamente 4,5 milhões de instâncias,

[2] *Dataset* parcial

[3] Ensemble composto por GBDT, DT e LR

em relação às 16,2 milhões disponíveis.

Já Kanimozhi & Jacob (2019) direcionaram seus esforços à classificação de ataques do tipo *BotNet*. Apesar das elevadas métricas reportadas, seu trabalho abrange poucos detalhes acerca da metodologia aplicada. São destacados alguns parâmetros configurados para utilização do algoritmo, mencionando que houve otimização de hiper-parâmetros, porém sem apresentar os resultados e o espaço de busca. Também não foram fornecidos detalhes relacionados ao pré-processamento. Deste modo, não é possível a reprodução dos experimentos conduzidos pelos autores.

O trabalho de Chimphlee & Chimphlee (2023b) realiza comparações entre alguns algoritmos, e sugere que a aplicação de seleção de atributos com o método de ganho de informação proporciona melhorias aos resultados. Entretanto, os próprios autores destacam que uma análise mais completa pode se fazer necessária para validar que as diferenças encontradas sejam significativas.

# Capítulo 6

# Conclusão

Entre as características dos conjuntos de dados que podem ser prejudiciais ao processo de aprendizado de máquina, está o desbalanceamento. O mesmo está presente na base de dados de tráfego de rede utilizada neste trabalho, o CSE-CIC-IDS2018. Tal *dataset* foi selecionado no presente trabalho de detecção de intrusão por trazer dados realísticos, além de ser mais atualizada, em comparação aos *datasets* clássicos utilizados mais amplamente.

Visando fornecer um estudo ampliando a compreensão acerca dos efeitos de métodos endereçando o desbalanceamento, este trabalho abordou um método combinando o uso de técnicas de amostragem e treinamento sensível a custo, incluindo técnicas de validação, coleta de métricas robustas, e aplicação de testes estatísticos. A construção do método experimental também foi delineada tendo em vista os resultados apresentados em trabalhos relacionados ao tema, definidos durante a revisão bibliográfica, que demonstraram efetividade na aplicação das técnicas para balanceamento, sugerindo a possibilidade de aplicação com resultados positivos neste estudo.

Além disso, foi identificada uma escassez de trabalhos com o *dataset* CSE-CIC-IDS2018, direcionada a classificação multi-classe, o que pode evidenciar a baixa performance na identificação de alguns tipos de ataques. Deste modo, a utilização desse método de classificação proporciona uma ampliação no entendimento relacionado à performance em distintos tipos de ataque, também fornecendo uma base para comparação a trabalhos futuros.

Os resultados obtidos com a aplicação da técnica *oversampling* SMOTE, concordam com descobertas prévias obtidas por outros autores, que sugerem melhoras de performance com sua utilização. Assim, foi possível concluir que para classificadores baseados em LightGBM, com a base de dados CIC-IDS2018, a aplicação de SMOTE foi capaz de aumentar a eficácia nas predições, avaliada pela métrica AUC.

Quanto à definição de pesos para treinamento sensível a custo, os resultados alcançados demonstraram que foi proporcionada uma melhoria na estabilidade dos modelos gerados no processo de *cross-validation*, e pequenas diferenças na performance de classificação de algumas classes. Entretanto, o resultado global do modelo, mensurado pela métrica AUC, não demonstrou ganhos estatisticamente significativos.

Este trabalho não abordou a calibração de probabilidades dos modelos gerados, o que

pode proporcionar aumento na confiança das predições obtidas. Além disso, melhorias podem ser proporcionadas em métricas como acurácia, precisão, sensibilidade e *F-score*. Trabalhos futuros podem vislumbrar a implementação e estudo dessa melhoria.

No método proposto no presente trabalho, a aplicação das técnicas foi realizada de forma sequencial, com a otimização de razões SMOTE, e posterior otimização de pesos para treinamento sensível a custo. Estudos adicionais podem ser realizados avaliando os efeitos de cada uma destas tarefas de maneira individual.

# Referências Bibliográficas


Ahmad, T. & Aziz, M. N. (2019). Data preprocessing and feature selection for machine learning intrusion detection systems, *ICIC Express Letters* **13**: 93–101. Citado na página 48.

Alpaydin, E. (2014). *Introduction to Machine Learning*, Adaptive Computation and Machine Learning, 3 edn, MIT Press, Cambridge, MA. Citado 3 vezes nas páginas 31, 39 e 40.

Alpaydin, E. (2021). *Machine Learning*, The MIT Press.
**URL:** *https://doi.org/10.7551/mitpress/13811.001.0001* Citado 4 vezes nas páginas 9, 31, 33 e 34.

Amin, A., Anwar, S., Adnan, A., Nawaz, M., Howard, N., Qadir, J., Hawalah, A. & Hussain, A. (2016). Comparing oversampling techniques to handle the class imbalance problem: A customer churn prediction case study, *IEEE Access* **4**: 7940–7957. Citado 2 vezes nas páginas 16 e 17.

Anghel, A., Papandreou, N., Parnell, T. P., Palma, A. D. & Pozidis, H. (2018). Benchmarking and optimization of gradient boosted decision tree algorithms, *CoRR* **abs/1809.04559**.
**URL:** *http://arxiv.org/abs/1809.04559* Citado na página 77.

Beaman, C., Barkworth, A., Akande, T. D., Hakak, S. & Khan, M. K. (2021). Ransomware: Recent advances, analysis, challenges and future research directions, *Computers & Security* **111**: 102490.
**URL:** *https://www.sciencedirect.com/science/article/pii/S016740482100314X* Citado na página 28.

Bergstra, J., Yamins, D. & Cox, D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, *in* S. Dasgupta & D. McAllester (eds), *Proceedings of the 30th International Conference on Machine Learning*, Vol. 28 of *Proceedings of Machine Learning Research*, PMLR, Atlanta, Georgia, USA, pp. 115–123.
**URL:** *https://proceedings.mlr.press/v28/bergstra13.html* Citado 2 vezes nas páginas 64 e 71.

Bissell, K., Fox, J., LaSalle, R. M. & Dal Cin, P. (2021). State of cybersecurity report, *Technical report*, Accenture. Citado na página 14.

Chawla, N. V., Bowyer, K. W., Hall, L. O. & Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique, *J. Artif. Int. Res.* **16**(1): 321–357. Citado na página 16.

Chicco, D. & Jurman, G. (2020). The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation, *BMC Genomics* **21**. Citado na página 44.

Chimphlee, S. & Chimphlee, W. (2023a). Machine learning to improve the performance of anomaly-based network intrusion detection in big data, *Indonesian Journal of Electrical Engineering and Computer Science* .
**URL:** *https://api.semanticscholar.org/CorpusID:257034141* Citado 4 vezes nas páginas 51, 52, 69 e 81.

Chimphlee, W. & Chimphlee, S. (2023b). Intrusion detection system(ids) development using tree-based machine learning algorithms, *International journal of Computer Networks & Communications* **15**: 93–109. Citado 3 vezes nas páginas 52, 81 e 82.

Crosbie, M. & Spafford, E. H. (1995). Defending a computer system using autonomous agents. Citado na página 14.

Demăr, J. (2008). On the Appropriateness of Statistical Tests in Machine Learning . **URL:** *http://www.site.uottawa.ca/ICML08WS/papers/J_Demsar.pdf* Citado na página 46.

Denning, D. & Neumann, P. G. (1985). Requirements and model for ides - a real-time intrusion-detection expert system, *Technical report*, SRI International. Citado 2 vezes nas páginas 15 e 21.

D'Hooge, L., Wauters, T., Volckaert, B. & De Turck, F. (2020). Inter-dataset generalization strength of supervised machine learning methods for intrusion detection, *Journal of Information Security and Applications* **54**: 102564. Citado 2 vezes nas páginas 15 e 22.

Elkan, C. (2001). The foundations of cost-sensitive learning, *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'01, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, p. 973–978. Citado na página 17.

Encyclopaedia Britannica, i. (1991). *The New Encyclopaedia Britannica*, number pt. 1 in *The New Encyclopaedia Britannica*, Encyclopaedia Britannica. **URL:** *https://books.google.com.br/books?id=0m8oAQAAIAAJ* Citado na página 30.

Ertel, W. (2018). *Introduction to artificial intelligence*, Springer. Citado 3 vezes nas páginas 30, 41 e 42.

Faker, O. & Dogdu, E. (2019). Intrusion detection using big data and deep learning techniques, *Proceedings of the 2019 ACM Southeast Conference*, ACM SE '19, Association for Computing Machinery, New York, NY, USA, p. 86–93. **URL:** *https://doi.org/10.1145/3299815.3314439* Citado 2 vezes nas páginas 49 e 81.

Fawcett, T. (2006). An introduction to roc analysis, *Pattern Recognition Letters* **27**(8): 861–874. ROC Analysis in Pattern Recognition. **URL:** *https://www.sciencedirect.com/science/article/pii/S016786550500303X* Citado na página 44.

Fayyad, U., Piatetsky-Shapiro, G. & Smyth, P. (1996). From data mining to knowledge discovery in databases, *AI Magazine* **17**(3): 37. **URL:** *https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1230* Citado 2 vezes nas páginas 32 e 39.

Feily, M., Shahrestani, A. & Ramadass, S. (2009). A survey of botnet and botnet detection, *2009 Third International Conference on Emerging Security Information, Systems and Technologies*, pp. 268–273. Citado na página 28.

Fernández, A., Garcia, S., Herrera, F. & Chawla, N. (2018). Smote for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary, *Journal of Artificial Intelligence Research* **61**: 863–905. Citado na página 60.

Fisher, R. A. (1925). *Statistical Methods for Research Workers*, Oliver and Boyd. Citado na página 66.

Fitni, Q. R. S. & Ramli, K. (2020). Implementation of ensemble learning and feature selection

for performance improvements in anomaly-based intrusion detection systems, *2020 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*, pp. 118–124. Citado 5 vezes nas páginas 16, 50, 52, 80 e 81.

Govindarajan, M. & Chandrasekaran, R. (2011). Intrusion detection using neural based hybrid classification methods, *Computer networks* **55**(8): 1662–1671. Citado 2 vezes nas páginas 15 e 22.

Guo, H., Zhou, J. & Wu, C.-a. (2020). Ensemble learning via constraint projection and undersampling technique for class-imbalance problem, *Soft Computing* **24**(7): 4711–4727.
**URL:** *https://doi.org/10.1007/s00500-019-04501-6* Citado 2 vezes nas páginas 16 e 17.

Gupta, B. B. & Badve, O. (2017). Taxonomy of dos and ddos attacks and desirable defense mechanism in a cloud computing environment, *Neural Computing and Applications* **28**. Citado na página 28.

He, H. & Garcia, E. A. (2009). Learning from imbalanced data, *IEEE Transactions on Knowledge and Data Engineering* **21**(9): 1263–1284. Citado na página 16.

Heady, R., Luger, G., Maccabe, A. & Servilla, M. (1990). The architecture of a network level intrusion detection system, *Technical report*, Los Alamos National Lab., NM (United States); New Mexico Univ., Albuquerque. Citado na página 14.

Hoque, N., Bhuyan, M. H., Baishya, R., Bhattacharyya, D. & Kalita, J. (2014). Network attacks: Taxonomy, tools and systems, *Journal of Network and Computer Applications* **40**: 307–324.
**URL:** *https://www.sciencedirect.com/science/article/pii/S1084804513001756* Citado na página 29.

Hua, Y. (2020). An efficient traffic classification scheme using embedded feature selection and lightgbm, *2020 Information Communication Technologies Conference (ICTC)*, pp. 125–130. Citado 2 vezes nas páginas 18 e 69.

Huang, J. & Ling, C. (2005). Using auc and accuracy in evaluating learning algorithms, *IEEE Transactions on Knowledge and Data Engineering* **17**(3): 299–310. Citado na página 45.

Igure, V. M. & Williams, R. D. (2008). Taxonomies of attacks and vulnerabilities in computer systems, *IEEE Communications Surveys & Tutorials* **10**(1): 6–19. Citado 2 vezes nas páginas 27 e 29.

Japkowicz, N. & Stephen, S. (2002). The class imbalance problem: A systematic study, *Intell. Data Anal.* **6**(5): 429–449. Citado 3 vezes nas páginas 17, 47 e 51.

Kanimozhi, V. & Jacob, P. (2019). Artificial intelligence based network intrusion detection with hyper-parameter optimization tuning on the realistic cyber dataset cse-cic-ids2018 using cloud computing, pp. 0033–0036. Citado 3 vezes nas páginas 49, 81 e 82.

Kantardzic, M. (2019). *Data Mining*, Wiley.
**URL:** *https://doi.org/10.1002/9781119516057* Citado na página 40.

Karatas, G., Demir, O. & Sahingoz, O. K. (2020). Increasing the performance of machine learning-based idss on an imbalanced and up-to-date dataset, *IEEE Access* **8**: 32150–32162. Citado 7 vezes nas páginas 14, 15, 16, 22, 49, 52 e 81.

Kaur, P. & Gosain, A. (2018). Issues and challenges of class imbalance problem in classification, *International Journal of Information Technology* . Citado 2 vezes nas páginas 16 e

17.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q. & Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree, *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, Curran Associates Inc., Red Hook, NY, USA, p. 3149–3157. Citado 2 vezes nas páginas 16 e 38.

Kruskal, W. H. & Wallis, W. A. (1952). Use of ranks in one-criterion variance analysis, *Journal of the American Statistical Association* **47**: 583–621.
**URL:** *https://api.semanticscholar.org/CorpusID:51902974* Citado na página 66.

Kulkarni, A., Chong, D. & Batarseh, F. A. (2020). 5 - foundations of data imbalance and solutions for a data democracy, *in* F. A. Batarseh & R. Yang (eds), *Data Democracy*, Academic Press, pp. 83–106.
**URL:** *https://www.sciencedirect.com/science/article/pii/B9780128183663000058* Citado 2 vezes nas páginas 16 e 44.

Kyatam, S., Alhayajneh, A. & Hayajneh, T. (2017). Heartbleed attacks implementation and vulnerability, *2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pp. 1–6. Citado na página 27.

Leevy, J. L., Hancock, J., Zuech, R. & Khoshgoftaar, T. M. (2021). Detecting cybersecurity attacks across different network features and learners, *Journal of Big Data* **8**(1): 38.
**URL:** *https://doi.org/10.1186/s40537-021-00426-w* Citado 7 vezes nas páginas 15, 22, 50, 52, 59, 68 e 69.

Leevy, J. L. & Khoshgoftaar, T. M. (2020). A survey and analysis of intrusion detection models based on CSE-CIC-IDS2018 big data, *Journal of Big Data* **7**(1): 104. Citado 2 vezes nas páginas 16 e 48.

Levene, H. (1961). Robust tests for equality of variances.
**URL:** *https://api.semanticscholar.org/CorpusID:117424234* Citado na página 65.

Lippmann, R., Haines, J. W., Fried, D. J., Korba, J. & Das, K. (2000). The 1999 darpa off-line intrusion detection evaluation, *Computer Networks* **34**(4): 579–595. Recent Advances in Intrusion Detection Systems.
**URL:** *https://www.sciencedirect.com/science/article/pii/S1389128600001390* Citado 2 vezes nas páginas 15 e 23.

Machado, R. B. (2005). *Uma abordagem de detecção de intrusão baseada em sistemas imunológicos artificiais e agentes móveis*, Master's thesis, UNIVERSIDADE FEDERAL DE SANTA CATARINA. Citado 2 vezes nas páginas 15 e 22.

Mahoney, M. V. & Chan, P. K. (2003). An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection, *in* G. Vigna, C. Kruegel & E. Jonsson (eds), *Recent Advances in Intrusion Detection*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 220–237. Citado na página 23.

McHugh, J. (2000). Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory, *ACM Trans. Inf. Syst. Secur.* **3**(4): 262–294.
**URL:** *https://doi.org/10.1145/382912.382923* Citado na página 23.

Mitchell, T. (1997). *Machine Learning*, McGraw-Hill International Editions, McGraw-Hill.
**URL:** *https://books.google.com.br/books?id=EoYBngEACAAJ* Citado na página 31.

Morgan, S. & Braue, D. (2022). Boardroom cybersecurity report, *Technical report*, Cybersecurity Ventures. Citado 2 vezes nas páginas 9 e 15.

Morgan, S. & Osborne, C. (2023). Boardroom cybersecurity report, *Technical report*, Cybersecurity Ventures. Citado na página 14.

Mucherino, A., Papajorgji, P. J. & Pardalos, P. M. (2009). *k-Nearest Neighbor Classification*, Springer New York, New York, NY, pp. 83–106.
**URL:** *https://doi.org/10.1007/978-0-387-88615-2_4* Citado 2 vezes nas páginas 50 e 60.

Norvig, P. & Russell, S. J. (2020). *Artificial Intelligence: A Modern Approach*, Pearson series in artificial intelligence, fourth edition edn, Pearson.
**URL:** *libgen.li/file.php?md5=188ec1f979f92af02697f4066f3dd93a* Citado 3 vezes nas páginas 30, 36 e 37.

Orebaugh, A. & Pinkard, B. (2011). *Nmap in the Enterprise: Your Guide to Network Scanning*, Elsevier Science.
**URL:** *https://books.google.com.br/books?id=VjgezB784XIC* Citado na página 30.

Panigrahi, R. & Borah, S. (2018). A detailed analysis of cicids2017 dataset for designing intrusion detection systems, *International Journal of Engineering & Technology* **7**: 479–482. Citado na página 16.

Pears, R., Finlay, J. & Connor, A. M. (2014). Synthetic minority over-sampling technique(smote) for predicting software build outcomes.
**URL:** *https://arxiv.org/abs/1407.2330* Citado 2 vezes nas páginas 16 e 17.

Pfahringer, B. (2000). Winning the kdd99 classification cup: Bagged boosting, *SIGKDD Explor. Newsl.* **1**(2): 65–66.
**URL:** *http://doi.acm.org/10.1145/846183.846200* Citado 2 vezes nas páginas 15 e 22.

Piatetsky-Shapiro, G., Matheus, C., Smyth, P. & Uthurusamy, R. (1994). Kdd-93: Progress and challenges in knowledge discovery in databases, *AI Magazine* **15**(3): 77.
**URL:** *https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1103* Citado na página 32.

Pietro, R. D. & Mancini, L. V. (2008). Intrusion detection systems, ISBN 1441945857. Citado na página 20.

Quinlan, R. (2001). *C5.0: An Informal Tutorial*.
**URL:** *http://www.rulequest.com/see5-unix.html* Citado na página 47.

Rich, E. (1983). *Artificial Intelligence*, McGraw-Hill, Inc., USA. Citado na página 30.

Saxena, A., Sinha, S. & Shukla, P. (2017). General study of intrusion detection system and survey of agent based intrusion detection system, pp. 471–421. Citado na página 14.

Seo, J.-H., Kim, Y.-H. & Lo Bosco, G. (2018). Machine-learning approach to optimize smote ratio in class imbalance dataset for intrusion detection, **2018**.
**URL:** *https://doi.org/10.1155/2018/9704672* Citado 3 vezes nas páginas 48, 71 e 72.

SHAPIRO, S. S. & WILK, M. B. (1965). An analysis of variance test for normality (complete samples)†, *Biometrika* **52**(3-4): 591–611.
**URL:** *https://doi.org/10.1093/biomet/52.3-4.591* Citado na página 65.

Sharafaldin, I., Lashkari, A. H. & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization., *ICISSP*, pp. 108–116. Citado 3

vezes nas páginas 15, 18 e 23.

Shiravi, A., Shiravi, H., Tavallaee, M. & Ghorbani, A. A. (2012). Toward developing a systematic approach to generate benchmark datasets for intrusion detection, *Computers & Security* **31**(3): 357–374.
**URL:** *https://www.sciencedirect.com/science/article/pii/S0167404811001672* Citado 2 vezes nas páginas 15 e 23.

Siddique, K., Akhtar, Z., Aslam Khan, F. & Kim, Y. (2019). Kdd cup 99 data sets: A perspective on the role of data sets in network intrusion detection research, *Computer* **52**(2): 41–51. Citado na página 23.

Sohi, S. M., Seifert, J.-P. & Ganji, F. (2021). Rnnids: Enhancing network intrusion detection systems through deep learning, *Comput. Secur.* **102**(C).
**URL:** *https://doi.org/10.1016/j.cose.2020.102151* Citado na página 14.

Sommer, R. & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection, *2010 IEEE Symposium on Security and Privacy*, pp. 305–316. Citado 2 vezes nas páginas 15 e 23.

Songma, S., Sathuphan, T. & Pamutha, T. (2023). Optimizing intrusion detection systems in three phases on the cse-cic-ids-2018 dataset, *Computers* **12**(12).
**URL:** *https://www.mdpi.com/2073-431X/12/12/245* Citado 3 vezes nas páginas 51, 52 e 81.

Souza, C. A. d. (2018). *Método híbrido de detecção de intrusão aplicando inteligência artificial*, Master's thesis, Universidade Estadual do Oeste do Paraná. Citado 2 vezes nas páginas 15 e 22.

Spirakis, M. D. D. K. D. G. P. (1994). Intrusion detection: Approach and performance issues of the securenet system, *Computers & Security vol. 13 iss. 6* **13**.
**URL:** *libgen.li/file.php?md5=727295c14d51592c6f881939b92c9fc7* Citado 2 vezes nas páginas 15 e 22.

Tahir, M. A. U. H., Asghar, S., Manzoor, A. & Noor, M. A. (2019). A classification model for class imbalance dataset using genetic programming, *IEEE Access* **7**: 71013–71037. Citado 2 vezes nas páginas 17 e 49.

Tang, M., Zhao, Q., Wu, H. & Wang, Z. (2021). Cost-sensitive lightgbm-based online fault detection method for wind turbine gearboxes, *Frontiers in Energy Research* **9**.
**URL:** *https://www.frontiersin.org/articles/10.3389/fenrg.2021.701574* Citado 3 vezes nas páginas 50, 75 e 81.

Tavallaee, M., Bagheri, E., Lu, W. & Ghorbani, A. A. (2009). A detailed analysis of the kdd cup 99 data set, *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pp. 1–6. Citado 2 vezes nas páginas 15 e 23.

Thakkar, A. & Lohiya, R. (2020). A review of the advancement in intrusion detection datasets, *Procedia Computer Science* **167**: 636–645. International Conference on Computational Intelligence and Data Science.
**URL:** *https://www.sciencedirect.com/science/article/pii/S1877050920307961* Citado na página 15.

THE WORLD BANK (2022). Individuals using the internet (% of population).
**URL:** *https://data.worldbank.org/indicator/IT.NET.USER.ZS* Citado na página 14.

TURING, A. M. (1950). I.—COMPUTING MACHINERY AND INTELLIGENCE, *Mind* **LIX**(236): 433–460.
**URL:** *https://doi.org/10.1093/mind/LIX.236.433* Citado na página 30.

University of New Brunswick (2018). CIC-IDS-2018 dataset, https://www.unb.ca/cic/datasets/ids-2018.html. Citado 4 vezes nas páginas 9, 24, 26 e 29.

Van Calster, B., Van Belle, V., Condous, G., Bourne, T., Timmerman, D. & Van Huffel, S. (2008). Multi-class auc metrics and weighted alternatives, *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pp. 1390–1396. Citado na página 45.

Witten, I., Frank, E., Hall, M. & Pal, C. (2016). *Data Mining: Practical Machine Learning Tools and Techniques*, The Morgan Kaufmann Series in Data Management Systems, Elsevier Science.
**URL:** *https://books.google.com.br/books?id=1SylCgAAQBAJ* Citado 11 vezes nas páginas 9, 16, 32, 33, 34, 35, 36, 37, 41, 42 e 45.

Wu, S. X. & Banzhaf, W. (2010). The use of computational intelligence in intrusion detection systems: A review, *Applied Soft Computing* **10**(1): 1–35.
**URL:** *http://www.sciencedirect.com/science/article/pii/S1568494609000908* Citado 4 vezes nas páginas 9, 20, 21 e 22.

# Apêndice A

# Apêndices

## A.1 Erro com utilização de GOSS e CUDA

O bloco abaixo apresenta o *log* de erro quando configurado para utilização de aceleração com CUDA e *boosting* com GOSS. Foram acrescentadas quebras de linha a fim de permitir que a largura do conteúdo se adeque ao espaço disponível na página. O caminho apresentado */home/ec2-user/jupyter-lab/LightGBM/* é onde foi realizada a instalação do LightGBM no servidor em uso.

```
File ~/.local/lib/python3.9/site-packages/lightgbm/sklearn.py:1135,
    in LGBMClassifier.fit(self, X, y, sample_weight,  init_score,
    eval_set, eval_names, eval_sample_weight, eval_class_weight,
    eval_init_score, eval_metric, feature_name, categorical_feature,
    callbacks, init_model)
   1132         else:
   1133             valid_sets.append((valid_x,
   self._le.transform(valid_y)))
-> 1135 super().fit(
   1136     X,
   1137     _y,
   1138     sample_weight=sample_weight,
   1139     init_score=init_score,
   1140     eval_set=valid_sets,
   1141     eval_names=eval_names,
   1142     eval_sample_weight=eval_sample_weight,
   1143     eval_class_weight=eval_class_weight,
   1144     eval_init_score=eval_init_score,
   1145     eval_metric=eval_metric,
   1146     feature_name=feature_name,
   1147     categorical_feature=categorical_feature,
   1148     callbacks=callbacks,
   1149     init_model=init_model
```

```
   1150 )
   1151 return self

File ~/.local/lib/python3.9/site-packages/lightgbm/sklearn.py:839,
    in LGBMModel.fit(self, X, y, sample_weight, init_score, group,
    eval_set, eval_names, eval_sample_weight, eval_class_weight,
    eval_init_score, eval_group, eval_metric, feature_name,
    categorical_feature, callbacks, init_model)
    836 evals_result: _EvalResultDict = {}
    837 callbacks.append(record_evaluation(evals_result))
--> 839 self._Booster = train(
    840     params=params,
    841     train_set=train_set,
    842     num_boost_round=self.n_estimators,
    843     valid_sets=valid_sets,
    844     valid_names=eval_names,
    845     feval=eval_metrics_callable,  # type: ignore[arg-type]
    846     init_model=init_model,
    847     feature_name=feature_name,
    848     callbacks=callbacks
    849 )
    851 self._evals_result = evals_result
    852 self._best_iteration = self._Booster.best_iteration

File ~/.local/lib/python3.9/site-packages/lightgbm/engine.py:266,
in train(params, train_set, num_boost_round, valid_sets, valid_names,
feval, init_model, feature_name, categorical_feature,
keep_training_booster, callbacks)
    258 for cb in callbacks_before_iter:
    259     cb(callback.CallbackEnv(model=booster,
    260                             params=params,
    261                             iteration=i,
    262                             begin_iteration=init_iteration,
    263                             end_iteration=init_iteration
        + num_boost_round,
    264                             evaluation_result_list=None))
--> 266 booster.update(fobj=fobj)
    268 evaluation_result_list:
        List[_LGBM_BoosterEvalMethodResultType] = []
    269 # check evaluation result.

File ~/.local/lib/python3.9/site-packages/lightgbm/basic.py:3553,
in Booster.update(self, train_set, fobj)
```

```
   3551 if self.__set_objective_to_none:
   3552     raise LightGBMError('Cannot update due to
        null objective function.')
-> 3553 _safe_call(_LIB.LGBM_BoosterUpdateOneIter(
   3554     self.handle,
   3555     ctypes.byref(is_finished)))
   3556 self.__is_predicted_cur_iter =
        [False for _ in range(self.__num_dataset)]
   3557 return is_finished.value == 1

File ~/.local/lib/python3.9/site-packages/lightgbm/basic.py:237,
    in _safe_call(ret)
    229 """Check the return value from C API call.
    230
    231 Parameters
   (...)
    234     The return value from C API calls.
    235 """
    236 if ret != 0:
--> 237     raise LightGBMError(_LIB.LGBM_GetLastError()
        .decode('utf-8'))

LightGBMError: [CUDA] invalid argument
/home/ec2-user/jupyter-lab/LightGBM/src/boosting/goss.hpp 63
```