UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de São José do Rio Preto

LEANDRO MARCOS DA SILVA

Classificação de domínios recém-registrados por meio de DNS passivo aplicando técnicas de aprendizado de máquina

São José do Rio Preto
2022

Leandro Marcos da Silva

Classificação de domínios recém-registrados por meio de DNS passivo aplicando técnicas de aprendizado de máquina

Trabalho de Conclusão de Curso apresentado ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Financiador: NIC.br - Proc. 2764/2018

Orientador: Prof. Dr. Adriano Mauro Cansian

São José do Rio Preto

2022

S586c

Silva, Leandro Marcos da

Classificação de domínios recém-registrados por meio de DNS passivo aplicando técnicas de aprendizado de máquina / Leandro Marcos da Silva. -- São José do Rio Preto, 2022

75 p. : il., tabs.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (Unesp), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto

Orientador: Adriano Mauro Cansian

1. Computadores Medidas de segurança. 2. Inteligência artificial. 3. Nomes de domínio na Internet. 4. Fraude na Internet. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Leandro Marcos da Silva

Classificação de domínios recém-registrados por meio de DNS passivo aplicando técnicas de aprendizado de máquina

Trabalho de Conclusão de Curso apresentado ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Financiador: NIC.br - Proc. 2764/2018

Banca Examinadora:

Prof. Dr. Adriano Mauro Cansian
UNESP – São José do Rio Preto
Orientador

Prof. Dr. Geraldo Francisco Donegá Zafalon
UNESP – São José do Rio Preto

Profa. Dra. Rogéria Cristiane Gratão de Souza
UNESP – São José do Rio Preto

São José do Rio Preto

2022

Agradecimentos

Agradeço, primeiramente, a Deus por toda minha vida e por me permitir ultrapassar todos os obstáculos ao longo da realização deste trabalho.

Agradeço aos meus pais Ivone de Fátima Pavan Silva e João Batista da Silva, além da minha irmã Letícia Pavan da Silva por estarem sempre presentes na minha vida, acreditarem em mim, apoiarem em todos os momentos e entenderem a minha ausência enquanto eu me dedicava a realizar este trabalho.

Agradeço ao meu orientador Prof. Dr. Adriano Mauro Cansian por contribuir na minha formação acadêmica com o seu conhecimento, acompanhar e auxiliar no projeto, além da oportunidade que me deu de fazer parte do Laboratório ACME!, o qual foi um grande diferencial a minha graduação.

Agradeço aos membros do Laboratório ACME!, principalmente o "Team DNS", composto por Marcos Rogério Silveira, Victor Fernandes Gardini e Vinicios Barretos Batista, por todos ensinamentos e cooperação, o que me permitiu desenvolver e concluir este presente trabalho.

Agradeço aos meus amigos Paulo André Pimenta Aragão e Victor Fernandes Gardini por toda amizade no decorrer da graduação que tornou possível a superação de todos os obstáculos. Agradeço a Patricia Giriolo Forte por me apoiar em todos os momentos e acreditar em mim, sempre me motivando a não desistir dos meus sonhos.

Agradeço a Universidade Estadual Paulista "Júlio de Mesquita Filho" (Unesp) pela existência do curso de Bacharelado em Ciência da Computação e aos professores do Departamento de Ciências de Computação e Estatística por todos os conhecimentos passados ao longo desses quatro anos.

Enfim, agradeço a todos aqueles que contribuíram, indiretamente ou diretamente, para o desenvolvimento deste trabalho de pesquisa, incrementando de alguma forma o meu processo de aprendizado.

Este trabalho foi financiado por intermédio do Convênio de Pesquisa e Inovação com o Núcleo de Informação e Coordenação do Ponto BR (NIC.br), processo Fundação para o Desenvolvimento da UNESP (Fundunesp), N.º 2764/2018.

Não importa quanto a vida possa ser ruim, sempre existe algo que você pode fazer, e triunfar. Enquanto há vida, há esperança.

(Stephen Hawking)

Resumo

Uma grande quantidade de nomes de domínios é registrada diariamente, incluindo domínios maliciosos, os quais são aplicados para os mais diversos ataques com o objetivo de roubar informações e contaminar máquinas. Neste cenário, soluções que usam técnicas de aprendizado de máquina surgem para a classificação de domínios com abordagens que analisam características textuais, DNS ativo e passivo. Contudo, existem carências na classificação de domínios, principalmente de domínios recém-registrados. Diante disso, este trabalho propõe um sistema para a classificação de domínios recém-registrados por meio do DNS passivo, o qual aplica cinco modelos para monitorar o primeiro mês de vida dos domínios após a primeira *query*, sendo cada modelo referente a um certo intervalo de tempo. Os dados são coletados do servidor autoritativo de um *Top-Level Domain*, e, com isso, 20 características são extraídas do DNS passivo com informações de geolocalização. Além disso, os métodos *Cluster Centroids* e K-*Means* SMOTE foram combinados para o balanceamento de classes, dado que há mais domínios legítimos do que maliciosos. Para o treinamento dos modelos, comparou-se cinco algoritmos de aprendizado de máquina baseados em árvores, dos quais o LightGBM obteve os melhores resultados. Por fim, dois modelos do sistema foram validados com novos dados, obtendo uma Taxa de Verdadeiro Positivo média de 0,8669.

Palavras-chave: DNS Passivo, Domínios Recém-Registrados, Domínios Maliciosos, Aprendizado de Máquina, Balanceamento de Classes.

Abstract

Many domain names are registered daily, including malicious domains which are applied to the most diverse attacks to steal information and infect machines. In this scenario, solutions that use machine learning techniques emerge to classify domains with approaches that analyze lexical features, active and passive DNS. However, there are deficiencies in the classification of domains, especially in newly registered domains. Therefore, this work proposes a system for the classification of newly registered domains through passive DNS, which applies five models to monitor the first month of life of domains after the first query, with each model referring to a specific interval of time. Data is collected from the authoritative server of a Top-Level Domain, and 20 features are extracted from passive DNS with geolocation information. In addition, Cluster Centroids and K-Means SMOTE methods were combined for class balancing, given that there are more legitimate than malicious domains. Five tree-based machine learning algorithms were compared for training the models, of which LightGBM obtained the best results. Finally, two models of the system were validated with new data, obtaining an average True Positive Rate of 0.8669.

Keywords: Passive DNS, Newly Registered Domains, Malicious Domains, Machine Learning, Data Imbalance.

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

AA – *Authoritative Answer*

ANCOUNT – *Answer Count*

ARCOUNT – *Additional Information Count*

AS – *Autonomous System*

ASN – *Autonomous System Number*

AUC – *Area Under the ROC Curve*

BIND – *Berkeley Internet Name Domain*

C&C – *Command and Control*

CatBoost – *Categorical Boosting*

CC – *Cluster Centroids*

ccTLD – *Country Code Top-Level Domain*

CD – *Checking Disabled*

CDN – Rede de Distribuição de Conteúdo

CNAME – *Canonical Name*

DDoS – *Distributed Denial of Service*

DGA – *Domain Generation Algorithm*

DNS – *Domain Name System*

DNSSEC – *Domain Name System Security Extensions*

DT – *Decision Tree*

E – Especificidade

EFB – *Exclusive Feature Bundling*

FN – Falso Negativo

FP – Falso Positivo

FQDN – *Fully Qualified Domain*

GB – Gigabyte

GBDT – *Gradient Boosting Decision Tree*

GBM – *Gradient Boosting Machine*

GHz – Gigahertz

GOSS – *Gradient-based One-Side Sampling*

gTLD – *Generic Top-Level Domain*

HTTP – *Hypertext Transfer Protocol*

IDE – *Integrated Development Environment*

IP – *Internet Protocol*

ISP – *Internet Service Provider*

KNN – K-*Nearest Neighbors*

LightGBM – *Light Gradient Boosting Machine*

M3D – Modelo de Três Dias

ML – *Machine Learning*

MS1 – Modelo da Semana 1

MS2 – Modelo da Semana 2

MS3 – Modelo da Semana 3

MS4 – Modelo da Semana 4

MX – *Mail Exchange*

NIC.br – Núcleo de Informação e Coordenação do Ponto BR

NS – *Name Server*

NSCOUNT – *Authority Count*

NXDomain – *Non-Existent Domain*

PCA – *Principal Component Analysis*

pDNS – *Passive DNS*

PR – Precisão-*Recall*

PTR – *Pointer*

QNAME – *Query Name*

QTYPE – *Query Type*

RAM – *Random Access Memory*

RCODE – *Response Code*

RF – *Random Forest*

ROC – *Receiver Operating Characteristic*

ROS – *Random Oversampling*

RRs – *Resource Records*

RUS – *Random Undersampling*

SMOTE – *Synthetic Minority Oversampling Technique*

SOA – *Start of Authority*

SSD – *Solid State Drive*

SVM – *Support Vector Machine*

TCP – *Transmission Control Protocol*

TFN – Taxa de Falso Negativo

TFP – Taxa de Falso Positivo

TLD – *Top-Level Domain*

TTL – *Time to Live*

TVN – Taxa de Verdadeiro Negativo

TVP – Taxa de Verdadeiro Positivo

UDP – *User Datagram Protocol*

UNESP – Universidade Estadual Paulista “Júlio de Mesquita Filho”

URL – *Uniform Resource Locator*

VN – Verdadeiro Negativo

VP – Verdadeiro Positivo

XGBoost – *Extreme Gradient Boosting*

Sumário

Capítulo 1 – Introdução

1.1. Considerações Iniciais

No início da Internet, não havia um serviço que traduzia os nomes de *hosts* em endereços *Internet Protocol* (IP). A tradução era feita por meio de um arquivo local em texto plano, armazenado na máquina do usuário, o qual se denominava *hosts.txt* e continha uma lista com os nomes de *hosts* existentes e seus endereços IP (LIU; ALBITZ, 2006). Com a evolução da Internet, a solução tornou-se inviável, e, com isso, originou-se o *Domain Name System* (DNS). O DNS é um banco de dados distribuído que fornece o mapeamento de nomes de *hosts* em endereços IP e vice-versa (FALL; STEVENS, 2011). Atualmente, existem 566.500.663 domínios registrados no mundo divididos entre os 1.599 *Top-Level Domains* (TLDs) (DOMAIN NAME STAT, 2022), dentre os quais o TLD ".br" corresponde a 4.842.785 dos domínios (REGISTRO.BR, 2022). Diante disso, o DNS é um protocolo vital para a operação da Internet, pois permite que nomes sejam memorizados em vez de longos números, facilitando a navegação dos usuários pela Web.

Dado o crescimento da Internet e a importância substancial do DNS, pessoas mal intencionadas se aproveitam da estrutura do DNS para o abuso de domínios e registram nomes de domínios para fins maliciosos, como a prática de *phishing*, disseminação de *malwares*, *botnets*, *fast-flux domains*, *malicious Domain Generation Algorithms* (DGAs), entre outros tipos de ataques (TORABI *et al.*, 2018), os quais continuam sendo realizados atualmente. Segundo a pesquisa da Kaspersky referente ao terceiro trimestre de 2021, o

Brasil foi classificado em primeiro lugar na lista dos países com maior participação na distribuição de ataques de *phishing*[1], com 6,63% (KULIKOVA *et al.*; 2021). Por isso, procura-se reconhecer e reportar tais domínios às autoridades para impedir os diferentes tipos de ataques, os quais resultam desde o roubo de informações pessoais, como nomes de usuários e senhas, até o uso das máquinas contaminadas para ataques em larga escala.

Uma forma tradicional de atenuar os domínios maliciosos é o uso de *blocklists*, em que domínios são listados como maliciosos para serem bloqueados depois, entretanto há um tempo considerável entre a detecção e inserção nas listas, e, nesse intervalo de tempo, mais usuários seriam vítimas desses domínios. Por esse motivo, sistemas para detecção de domínios maliciosos, aplicando técnicas de *Machine Learning* (ML), foram propostos como forma de solucionar o problema, além da utilização de diversas fontes de dados para o treinamento dos modelos. A detecção automatizada de domínios maliciosos costuma ser dividida em duas abordagens: a primeira é na etapa de pré-registro, que é quando um usuário se dirige até um *registry*, insere suas informações pessoais e efetua a compra de um nome de domínio (SPOOREN *et al.*, 2019); e a segunda é no pós-registro, que utiliza o DNS ativo (KOUNTOURAS *et al.*, 2016) e passivo (SILVEIRA *et al.*, 2020) para detectar esses domínios.

A partir do que foi apresentado, observa-se a necessidade da detecção de domínios maliciosos, em especial na etapa de pré-registro. Porém, quando determinados domínios contornam os métodos de detecção dessa etapa, é preciso ter um sistema de classificação após o registro, em especial um que monitore e classifique os domínios recém-registrados em legítimos ou maliciosos. Além do uso de dados do DNS passivo, visto que são dados reais das consultas e respostas DNS e contêm informações relevantes para a detecção de domínios maliciosos (WEIMER, 2005), e a aplicação de algoritmos recentes de ML.

1.2. Justificativa

A fim de que a classificação de domínios por meio de DNS passivo seja eficiente em um ambiente de produção, é preciso a utilização de algoritmos de ML robustos e o tratamento de classes desbalanceadas, além de uma fonte de dados que seja confiável. Em geral, nos trabalhos que utilizam dados originados do DNS passivo, a coleta é feita em servidores recursivos, e apenas em um trabalho (ANTONAKAKIS *et al.*, 2011) tem-se a coleta em um servidor autoritativo, o qual dispõe de autoridade para fornecer dados de

[1] Tentativa de obter dados pessoais por meio de cópias de websites que parecem reais (XIANG *et al.*, 2011).

um domínio. Sobre o tratamento do desbalanceamento de classes, no trabalho de Wang *et al.* (2020) é explorado o método K-*Means* SMOTE, mas não há um comparativo entre os métodos de balanceamento de classes. Sobre os algoritmos de ML, no sistema desenvolvido em Silveira *et al*. (2020), é usado o XGBoost, que consiste em um algoritmo com uma alta escalabilidade, mas nenhum outro algoritmo com técnicas mais recentes é aplicado na maioria dos trabalhos. Por fim, os autores abordam a detecção de domínios maliciosos na etapa pré e pós-registro, mas não classificam domínios recém-registrados por meio de DNS passivo, isto é, classificar logo após a primeira *query*, além de monitorar esses domínios.

Neste contexto discutido, surge-se a necessidade de um sistema que classifique domínios recém-registrados por meio de DNS passivo aplicando algoritmos de ML mais recentes, além do tratamento do desbalanceamento de classes e uso de uma fonte de dados confiável. Diante disso, a proposta desse trabalho é utilizar o tráfego DNS coletado do servidor autoritativo de um determinado TLD por meio de DNS passivo, o que garante uma maior visibilidade global por estar nos níveis superiores da hierarquia DNS. Além da utilização de algoritmos de ML com método *boosting* para a geração dos modelos, que são a nova geração de algoritmos baseados em árvores, como o LightGBM. Finalmente, o emprego dos métodos de balanceamento de classes recentes, como o K-*Means* SMOTE.

1.3. Objetivos

Este trabalho tem como objetivo geral o desenvolvimento de um sistema para classificar domínios recém-registrados por meio de DNS passivo, empregando os algoritmos de ML. Por fim, os objetivos específicos obtidos durante o desenvolvimento deste trabalho são:

- Análise por completo do domínio em todo seu primeiro mês de vida após a etapa de registro, e, ao final, classificar se é legítimo ou malicioso;
- Coleta do tráfego DNS do servidor autoritativo de um determinado TLD por meio de DNS passivo;
- Uso de características do DNS passivo junto com os dados de geolocalização que são enriquecidos;
- Comparação entre os algoritmos de ML baseados em árvores, além dos métodos de balanceamento de classes existentes;
- Aplicação da otimização Bayesiana nos algoritmos XGBoost e LightGBM.

1.4. Metodologia

O desenvolvimento deste trabalho foi dividido em cinco etapas, conforme segue:

- **Levantamento bibliográfico:** buscar e analisar trabalhos recentes que retratam o tópico da classificação de domínios por meio de DNS passivo aplicando técnicas de ML. A partir disso, é efetuada uma ordem cronológica dos trabalhos e indicadas as principais contribuições de cada um;
- **Estudo das limitações existentes:** listar as limitações encontradas nos trabalhos anteriores, e, com isso, procurar em quais pontos é possível melhorar, além de avaliar a forma mais viável de resolver determinadas limitações;
- **Definição do sistema:** definir um sistema para efetuar a classificação de domínios recém-registrados a partir do estudo teórico e listagem das limitações existentes na literatura. Assim, define-se cada função do sistema e seu funcionamento em detalhes, o que auxilia na etapa de desenvolvimento;
- **Desenvolvimento do sistema:** desenvolver o sistema definido anteriormente, em que se implementa cada função específica, e, ao final, os modelos são construídos para classificar certo domínio em seu primeiro mês de vida após o registro, isto é, quando o domínio é consultado e o tráfego DNS começa a ser coletado. Logo, em três dias de coleta, tem-se a primeira classificação parcial do domínio, seguindo para sete dias e assim por diante até quatro semanas;
- **Avaliação dos resultados obtidos:** avaliar os resultados adquiridos do sistema no treinamento e teste dos modelos da etapa de desenvolvimento. A partir disso, é testado e validado o desempenho de cada modelo em novos dados coletados em outro período. Após isso, nota-se os pontos fortes e fracos a partir dos resultados obtidos, e, na sequência, propõe-se possíveis melhorias aos modelos construídos.

1.5. Organização da Monografia

Os capítulos subsequentes que formam a presente monografia para o cumprimento da disciplina Trabalho Conclusão de Curso são:

Capítulo 2 – Revisão Bibliográfica: Este capítulo contempla a fundamentação teórica necessária para a realização deste trabalho. Assim, neste capítulo são apresentados o protocolo DNS, contemplando a hierarquia dos servidores DNS e como é efetuada uma consulta DNS, além dos *Resource Records* (RRs), DNS passivo e ataques que envolvem o DNS. Depois, são apresentados os conceitos do ML e de alguns algoritmos empregados

no trabalho, e, em seguida, tem-se as técnicas de validação do modelo e métricas de avaliação, e, ainda, os métodos para balanceamento de classes. Por fim, são abordados os trabalhos correlatos, trazendo a contribuição de cada um e suas limitações.

Capítulo 3 – Desenvolvimento: Neste capítulo é apresentada a metodologia, com uma descrição geral do sistema, além do detalhamento de cada etapa, sendo a aquisição e preparação dos dados, extração de características, rotulação dos dados, balanceamento de classes, treinamento e teste dos modelos, além dos modelos em produção.

Capítulo 4 – Resultados: Este capítulo apresenta e discute todos os resultados alcançados neste trabalho, principalmente em relação aos métodos de balanceamento de classes e algoritmos de ML.

Capítulo 5 – Conclusões: Neste capítulo são apresentadas as conclusões obtidas com o desenvolvimento deste trabalho, além das dificuldades encontradas, propostas para trabalhos futuros e as produções acadêmicas conquistadas com este trabalho.

Capítulo 2 – Revisão Bibliográfica

2.1. Protocolo DNS

Os endereços IP são difíceis de serem utilizados e lembrados pelos humanos, em especial os endereços IPv6, que é a versão mais atual com endereços de 128 bits. Assim, surge o DNS, o qual se trata de um banco de dados distribuído que faz o mapeamento de nome de *host* em endereço IP e vice-versa. Para ofertar a escalabilidade, o DNS tem uma estrutura hierárquica, contando com servidores hierárquicos (FALL; STEVENS, 2011).

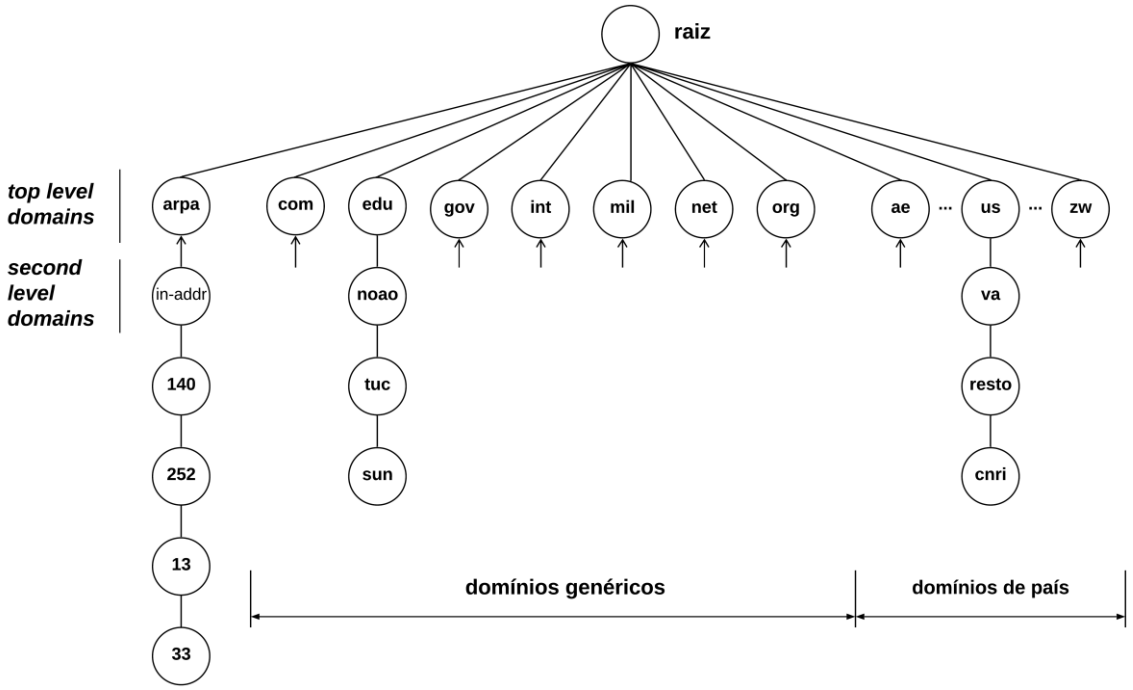
Além disso, o DNS é definido como um protocolo da camada de aplicação que utiliza UDP na porta 53, que permite a consulta de *host* em um banco de dados distribuído. Na maioria das vezes, os servidores DNS são máquinas Unix que contam com o software *Berkeley Internet Name Domain* (BIND). Um exemplo do uso do DNS é o acesso de uma página a partir de um navegador, que ocorre da seguinte modo (KUROSE; ROSS, 2013):

1. O computador do usuário executa o lado cliente (*resolver*) da aplicação DNS;
2. O navegador retira o nome de *host* do *Uniform Resource Locator* (URL) e informa o nome para o lado cliente da aplicação DNS;
3. O cliente envia uma consulta com o nome do *host* para o servidor DNS;
4. O cliente recebe uma resposta que tem o endereço IP do nome do *host*;
5. Com o endereço IP, o navegador pode abrir uma conexão *Transmission Control Protocol* (TCP) com o processo servidor *Hypertext Transfer Protocol* (HTTP), que se localiza na porta 80 do endereço IP.

2.1.1. Hierarquia DNS

Por conta de ser uma aplicação escalável diante o crescimento da Internet, o DNS utiliza vários servidores distribuídos que são organizados hierarquicamente no mundo todo. De maneira mais simples, existem três classes de servidores DNS: raiz, TLD, e servidores DNS autoritativos (KUROSE; ROSS, 2013). A estrutura hierárquica do DNS pode ser comparada com a estrutura de arquivos do Unix, porque os dois empregam a estrutura de árvore para o armazenamento de informações e ligações (STEVENS, 1994). Na Figura 1 é ilustrada a estrutura hierárquica do DNS.

Figura 1: Estrutura hierárquica do DNS



Fonte: Adaptado de Stevens, 1994

No topo da estrutura hierárquica do DNS encontram-se os servidores raiz, que consistem no ponto inicial da consulta DNS. Logo abaixo do raiz, tem-se os servidores da classe TLD, os quais são divididos em (STEVENS, 1994):

- **Arpa:** domínio especial aplicado para consulta DNS reversa, isto é, endereço IP para nome de *host*;
- ***Country Code Top-Level Domain* (ccTLD):** TLD que identifica algum país em específico, como o ".br" para o Brasil.
- ***Generic Top-Level Domain* (gTLD):** TLD genérico, o qual não necessita da posição geográfica.

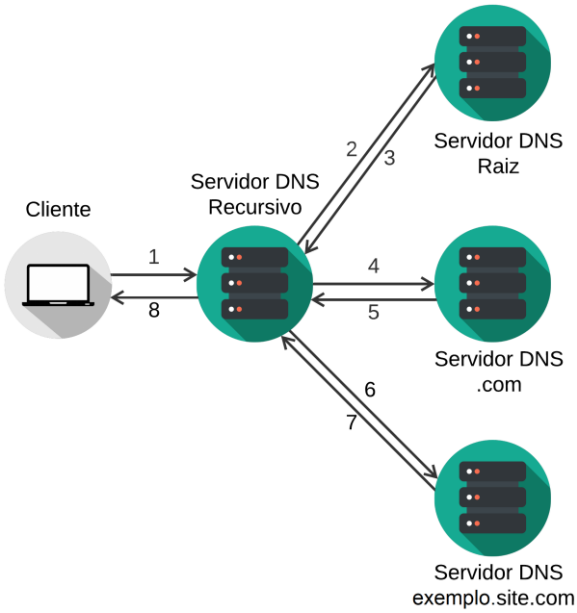
Depois do ccTLD ou gTLD, encontram-se os *second-level domain* e *third-level domain*, dentre outros, até que a formação completa do nome do domínio seja composta, o que se denomina *Fully Qualified Domain Name* (FQDN). De acordo com a Figura 1, para a formação do FQDN basta percorrer os nós da folha até a raiz, e assim junta-se os rótulos, dividindo-os com o ponto. Ainda, todos os nós têm que incluir ponteiros para os nós interiores, por exemplo, um servidor raiz tem os endereços IP de todos os servidores TLDs. Com isso, na Figura 1 forma-se três FQDNs, que são 33.13.252.140.in-addr.arp, sum.tuc.noao.edu e cnri.resto.va.us, com destaque ao primeiro FQDN, o qual se trata da classe arpa. É interessante ressaltar que os servidores raiz não possuem rótulo.

Além das classes de servidores DNS, há um tipo importante de DNS denominado servidor DNS recursivo, que não pertence a hierarquia, mas é considerado relevante para a arquitetura DNS. Cada *Internet Service Provider* (ISP) dispõe de um ou mais servidores DNS recursivos e, quando um *host* conecta em um ISP, são fornecidos os endereços IP de um ou mais servidores DNS recursivos. Ainda, o DNS recursivo contém um arquivo com o endereço IP dos 13 servidores raiz existentes (KUROSE; ROSS, 2013), em que cada servidor raiz tem instâncias espalhadas pelo mundo.

A partir da explicação da hierarquia DNS, torna-se mais simples compreender o funcionamento da consulta DNS, que envolve a interação com os servidores DNS, como ilustrado na Figura 2, e opera do seguinte modo para o domínio exemplo.site.com:

1. O cliente envia uma mensagem de consulta DNS ao seu servidor DNS recursivo com o nome que será traduzido, por exemplo, exemplo.site.com;
2. O servidor DNS recursivo encaminha a mensagem de consulta DNS para um dos servidores raiz;
3. O servidor raiz retorna ao servidor DNS recursivo os endereços IP dos servidores TLDs que respondem pelo “.com”;
4. O servidor DNS recursivo reenvia a mensagem de consulta DNS para um dos servidores TLDs;
5. O servidor TLD nota o sufixo site.com e retorna ao servidor DNS recursivo o endereço IP do servidor autoritativo do domínio;
6. O servidor DNS recursivo retransmite a mensagem de consulta DNS diretamente ao servidor autoritativo;
7. O servidor autoritativo responde com o endereço IP de exemplo.site.com; e
8. O cliente recebe, como resposta do servidor DNS recursivo, o IP do domínio.

Figura 2: Interação dos diversos servidores DNS



Fonte: Adaptado de Kurose; Ross, 2013

2.1.2. *Resource Records*

Os *Resource Records* (RRs) estão armazenados nos servidores DNS e fornecem os mapeamentos de nomes de *hosts* em endereços IP e vice-versa. Cada mensagem de resposta DNS contém um ou mais RRs, e, além disso, é possível enviar RRs específicos ao efetivar uma consulta DNS. Um RR constitui-se em uma tupla de quatro elementos, que tem os campos nome, valor, tipo e *Time to Live* (TTL). Os campos nome e valor têm dependência do tipo do RR, e, por sua vez, o TTL determina quando um RR precisa ser removido de um *cache* do servidor DNS (KUROSE; ROSS, 2013). Por fim, alguns tipos de RRs serão explicados a seguir (FALL; STEVENS, 2011):

- ***Address Record for IPv4* (A):** aponta um nome de *host* para um endereço IPv4;
- ***Address Record for IPv6* (AAAA):** aponta um *hostname* para um endereço IPv6;
- ***Canonical Name* (CNAME):** define um alias, isto é, sinônimo ou nome canônico, para um nome de *host*;
- ***Mail Exchange* (MX):** aponta onde entregar mensagens para um determinado domínio ou nome de *host*;
- ***Name Server* (NS):** aponta um nome de *host* para um domínio;
- ***Pointer* (PTR):** aponta um endereço IP para um nome de host (resolução reversa);
- ***Start of Authority* (SOA):** sinaliza o início de dados de uma certa zona e define parâmetros que afetam toda a zona.

2.1.3. DNS Passivo

O DNS passivo, do inglês *passive DNS* (pDNS), foi proposto por Weimer (2005), com o intuito de interromper ataques de *malwares*. Com isso, usou servidores de nomes de domínios recursivos para o registro de respostas recebidas de distintos servidores de nomes. Assim, o pDNS é a coleta da comunicação entre os servidores DNS, que é feita por meio de sensores instalados com acesso a servidores DNS para obter as consultas e respostas reais do DNS. Os dados de pDNS podem não retratar a estrutura DNS completa, porque contêm somente os nomes de domínios que foram consultados durante o período de coleta. No entanto, a agregação dos dados de pDNS constituem uma grande parte do tráfego, que propende a ser uma fonte confiável de dados (ANTONAKAKIS *et al.*, 2010).

O tráfego DNS possui uma quantidade interessante de recursos para identificar nomes de domínios ligados com atividades maliciosas, além de ser possível enriquecer as informações com dados associados ao *Autonomous System Number* (ASN), WHOIS, geolocalização, entre outros. Por conta disso, há vários locais da hierarquia DNS para fazer a coleta do pDNS, como instalar sensores em servidores DNS recursivos de ISP ou em servidores autoritativos de TLDs.

2.1.4. Ataques Envolvendo DNS

Apesar do avanço nos sistemas de detecção de ataques em Segurança Cibernética, os usuários da Internet são alvos de ataques envolvendo o ecossistema DNS. Diante disso, o DNS é mal utilizado para implementar, desenvolver, ofertar suporte e esconder várias atividades maliciosas. Sendo assim, determinados abusos que acontecem no DNS e o seu uso indevido serão discutidos a seguir (TORABI *et al.*, 2018).

Um dos problemas é o abuso no registro de nome de domínio, uma vez que os usuários são capazes de registrar qualquer nome de domínio disponível em qualquer hora na etapa de pré-registro, e, com isso, alguns usuários registram determinados nomes de domínios que geralmente representam entidades comerciais ou pessoas conhecidas, com o objetivo de lucrar sobre os nomes de domínios registrados. Essa ação descrita titula-se *cybersquatting* ou *domain squatting*, a qual é proibida em alguns países, como nos EUA. Além disso, existe o *typosquatting*, que se trata da prática de registrar nomes de domínios semelhantes a nomes de domínios conhecidos com a finalidade de gerar confusão entre os usuários da Internet, como exemplo, variações com erros de ortografia ou domínios conhecidos escritos de forma incorreta, e assim uma parte do tráfego DNS é direcionado

para esses domínios falsos. Um exemplo de *typosquatting* seria o amaz0n.com.br, em que altera a letra "o" pelo número "0" (zero), e muitas pessoas são vítimas desses domínios, principalmente ao inserirem informações pessoais e até cartões de crédito, e essa prática é caracterizada como um *phishing*, que é uma tentativa de conseguir dados confidenciais por meio de réplicas de websites que aparentam ser reais (XIANG *et al.*, 2011). Embora esses domínios nem sempre estejam relacionados em atividades maliciosas, os estudos mostraram que grande parte desses domínios têm envolvimento com diferentes ataques de *phishing* e campanhas fraudulentas (ALRWAIS *et al.*, 2014).

Além do problema descrito anteriormente, a estrutura aberta do DNS possibilita que atacantes operem e mantenham domínios e *hosts* maliciosos ou comprometidos na Internet. Assim, o atacante utiliza de modo indevido os serviços DNS para implementar atividades maliciosas na Internet, além de usar domínios e *hosts* comprometidos para a ocultação das suas atividades maliciosas e para impossibilitar a detecção por autoridades, e ainda os utilizar em ataques *Distributed Denial of Service* (DDoS). Em Torabi *et al.* (2018), são exploradas três formas de abuso DNS por meio de domínios e endereços IP maliciosos ou comprometidos, sendo elas:

- ***Botnets*:** os atacantes instalam um software malicioso, conhecido como *bots*, em distintas máquinas, e os *bots* são programados com a capacidade de interação entre si para a formação de uma rede de *bots* ou "*botnets*". O *botmaster* é o que controla e opera a *botnet* por meio de servidores *Command and Control* (C&C) para a execução de ataques, sendo ataques DDoS, campanhas de spam, coleta de dados, cyber-fraudes, disseminação de *malwares*, entre outros. É importante entender que as máquinas afetadas não são controladas de modo total pelo atacante, ou seja, a operação das *botnets* limita-se à operação das máquinas que estão estabelecidas;
- ***Fast-Flux Domains* e *Flux Networks*:** o *domain fluxing* trata-se de quando um certo nome de domínio resolve para vários *hosts* alternativos, isto é, endereços IP. O *domain name fluxing* foi desenvolvido com um propósito legítimo, como o balanceamento de carga e Rede de Distribuição de Conteúdo (CDN). Contudo, os atacantes utilizam do *domain name fluxing* para esconder rastros de atividades maliciosas por meio de atualizações rápidas de RRs, o que possibilita a resolução de um nome para *hosts* alternados. Em geral, o *fast fluxing* utiliza valores baixos de TTL, o que permite um IP diferente para o domínio em quase toda solicitação. A hospedagem do *fast-flux* é sobre redes distribuídas de sistemas maliciosos, que

são bem estabelecidas ou comprometidas, por exemplo, *botnets*. As *flux networks* são empregadas por atacantes para a realização de ataques distribuídos por toda a Internet (PERDISCI *et al.*, 2012). Em situação mais avançada, tem-se o *double flux*, que é quando os atacantes complementam a rede de serviço que hospeda websites com uma outra rede de serviço para hospedagem dos servidores DNS, e, com isso, é possível ocultar a localização do provedor de conteúdo malicioso;

- ***Malicious Domain Generation Algorithms* (DGAs) :** o DGA é usado para criar de modo automático vários nomes de domínios, e, por isso, atacantes aproveitam para gerar domínios maliciosos. Os *botnets* aproveitam-se dos DGAs para deixar a estrutura resiliente de *blocklists*, em que registram inumeráveis domínios para intercalarem. Além disso, há o registro de domínios maliciosos em distintos TLDs aplicando DGA, o que possibilita a distribuição global pela Internet.

Por fim, a estrutura aberta do DNS permite que os atacantes realizem a utilização inadequada desses serviços e aplicações legítimas para exercerem atividades maliciosas, como o número de *resolvers* abertos de DNS mostraram determinadas discrepâncias nas resoluções DNS, os quais foram desenvolvidos para a censura de canais de comunicação, transferência de arquivos maliciosos, *phishing* e injeção de anúncios (KÜHRER *et al.*, 2015). Além disso, entidades maliciosas, como *botnets*, foram localizadas empregando ferramentas de varredura para obterem informações sobre os servidores DNS, as quais identificam vulnerabilidades para realizarem ataques direcionados. Os atacantes também utilizam serviços e aplicações legítimas para evitar a detecção na Internet, de maneira a esconder vestígios dos ataques realizados (DURUMERIC *et al.*, 2014).

2.2. Aprendizado de Máquina

De acordo com Arthur Lee Samuel, um dos precursores da área de Inteligência Artificial, o aprendizado de máquina, ou do inglês *Machine Learning* (ML), é o campo de estudo que dá aos computadores a capacidade de aprender sem ser explicitamente programados, ou seja, é a ciência da programação que propicia o aprendizado por meio dos dados disponíveis, com o intuito de achar padrões (WEISS, 1992). O ML é aplicado desde a Economia até a Medicina, englobando várias áreas, e é relevante na atualidade, em especial devido ao *Big Data* e à evolução no desempenho das máquinas.

O ML divide-se em quatro tipos: aprendizado supervisionado, aprendizado não supervisionado, aprendizado semi-supervisionado e aprendizado por reforço. Diante do

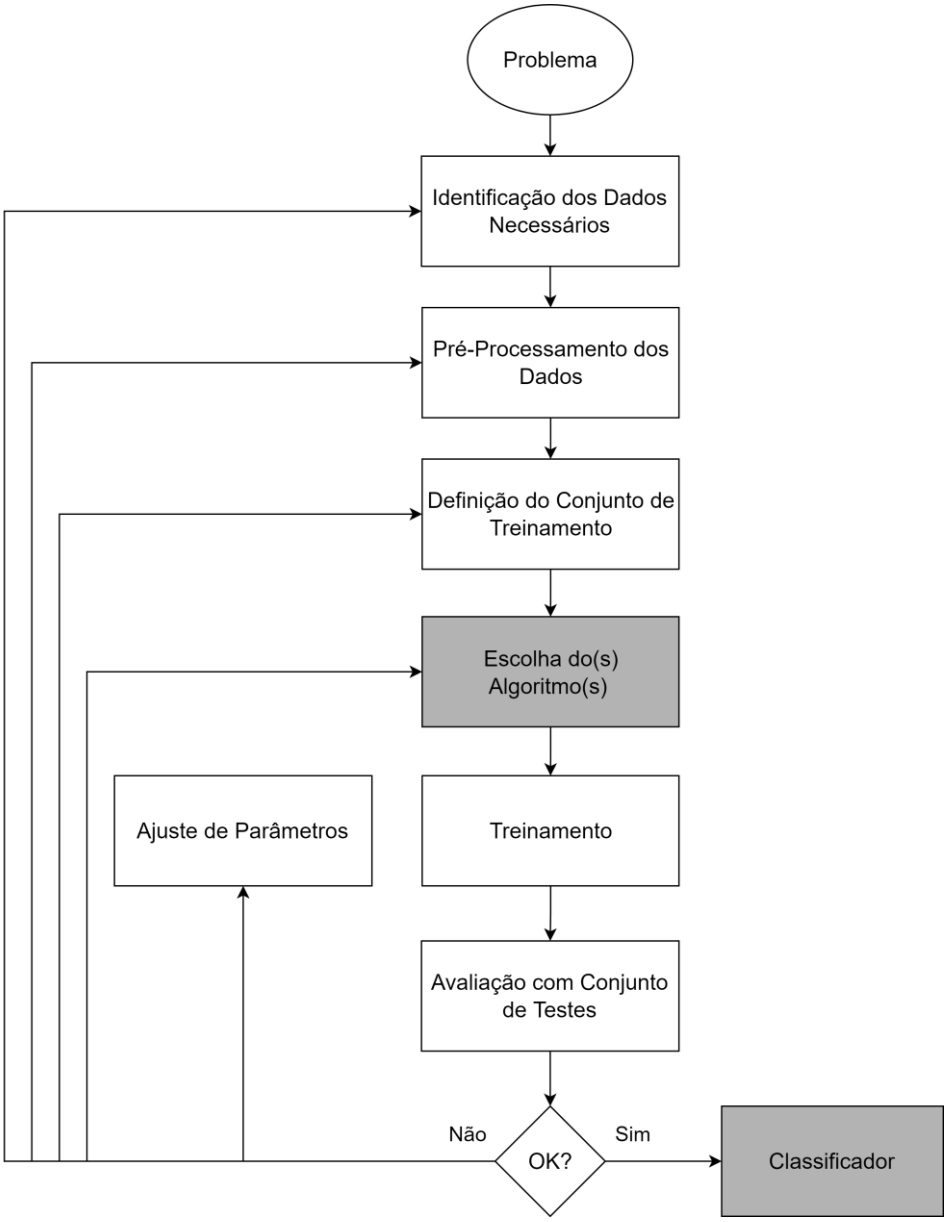
exposto, no aprendizado supervisionado, os dados providos para o algoritmo fazer a etapa de treino são rotulados e, com base nisso, o algoritmo é capaz de fazer previsões em dados não vistos e, além disso, geralmente é aplicado para tarefas de classificação e regressão. Por sua vez, no aprendizado não supervisionado, os dados de treinamento oferecidos ao algoritmo não têm rotulação, ou seja, o intuito é tentar aprender sozinho e ainda costuma ser aplicado para agrupamento e redução de dimensionalidade. O semi-supervisionado são algoritmos que dispõem a capacidade de aprender com uma pequena parte dos dados rotulados e uma grande parte não rotulados, combinando as técnicas supervisionadas e não supervisionadas. Por fim, no algoritmo por reforço, a máquina tenta aprender qual a melhor ação a ser decidida baseada em certa situação que tem punição ou recompensa ao fim de cada decisão (GÉRON, 2019).

Como citado anteriormente, a classificação é uma tarefa típica do aprendizado supervisionado, a qual é aplicada para classificar nomes de domínios, por exemplo, que se tem o treino do algoritmo com dados do tráfego de domínios legítimos e maliciosos, ou seja, cada domínio está rotulado como “legítimo” ou “malicioso”. A partir do treino, o modelo é capaz de diferenciar entre as duas classes em domínios não vistos e assim possibilita a classificação de domínios desconhecidos. É importante frisar a necessidade de haver a mesma quantidade de dados em cada classe, ou seja, que as classes estejam balanceadas, para que o classificador não opere de modo imparcial.

De acordo com Kotsiantis (2007), o aprendizado de máquina é a aprendizagem de um conjunto de regras baseado em instâncias e, na situação, os dados de treinamento fariam esse papel. De modo mais geral, é a criação de um classificador que possa generalizar a partir das instâncias. Na Figura 3, é exibido um fluxograma que se refere ao processo do aprendizado supervisionado para um problema real.

Uma etapa fundamental em ML, principalmente no aprendizado supervisionado, é a extração de características, que consiste na seleção e agrupamento dos atributos mais relevantes e assim os algoritmos são capazes de achar padrões. Ainda, destaca-se que as características extraídas nos dados de entrada devem ser as mesmas dos dados de treino. Após definir as características, é preciso escolher os dados que serão usados no treino, os quais estão rotulados em uma das classes, e a representação desses dados é uma matriz, em que as colunas são as características, com exceção da última coluna que é o rótulo, e as linhas são os dados de entrada. Depois de analisar os dados, um algoritmo é escolhido para treinar o modelo e, em sequência, tem-se a classificação em dados não rotulados.

Figura 3: Processo do aprendizado supervisionado



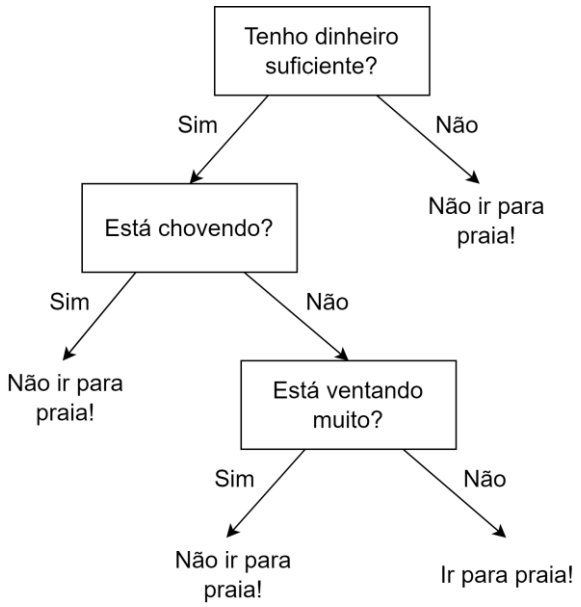
Fonte: Adaptado de Kotsiantis, 2007

A respeito dos algoritmos usados no aprendizado supervisionado, especialmente para a tarefa de classificação, tem-se algoritmos como a regressão logística, redes neurais, árvore de decisão, floresta aleatória, redes neurais, *K-Nearest Neighbors* (KNN), *Support Vector Machine* (SVM), entre outros (GÉRON, 2019). Os algoritmos utilizam diferentes abordagens para aprendizagem e, por conta disso, os resultados variam de um algoritmo para outro. Portanto, é importante selecionar o algoritmo certo para o problema que está solucionando para que tenha bons resultados. Nas subseções a seguir, serão apresentados determinados algoritmos de ML baseados em árvores, sendo: árvore de decisão, floresta aleatória, *gradient boosting machine*, XGBoost e LightGBM.

2.2.1. Árvore de Decisão

A árvore de decisão, ou do inglês *Decision Tree* (DT), classifica os dados com uma série de perguntas a respeito das características associadas aos dados (KINGSFORD; SALZBERG, 2008). Cada pergunta está dentro de um nó e, para cada resposta possível à pergunta, o nó interno aponta para um nó filho. Desse modo, as perguntas compõem uma estrutura hierárquica, codificada como uma árvore. Assim sendo, em essência, a DT é uma sequência de regras "se-então" e, por conta de aprender bem o conjunto de dados, são propensas a *overfitting*, que é quando o modelo se apresenta adequado somente para os dados de treinamento. Na Figura 4, tem-se um exemplo de DT para decidir se deve ou não ir para a praia, em que se repara perguntas do tipo sim ou não, iniciando da raiz, e, ao final, a resposta é obtida em um dos nós folhas.

Figura 4: Exemplo de árvore de decisão – Devo ir para praia?



Fonte: Elaborado pelo autor

Em Safavian; Landgrebe (1991), é discutido o classificador baseado no algoritmo DT e um ponto crítico no estudo é a construção da árvore, porque o algoritmo faz a análise das amostras de treino disponíveis e cria relações entre as suas características e rótulos. Por conta disso, é preciso exercer um estudo cauteloso sobre as características que serão adotadas no classificador, para assim se obter um classificador eficiente.

2.2.2. Floresta Aleatória

A floresta aleatória, ou do inglês *Random Forest* (RF), é um método *ensemble*, ou seja, combina-se vários modelos simples para ter um maior desempenho na classificação

em vez de usar somente um modelo complexo (GÉRON, 2019). Portanto, no RF, tem-se a combinação das árvores de decisão (floresta) em paralelo, em que cada árvore possui uma divisão aleatória, em linhas e colunas, do conjunto de dados e, com isso, contém-se árvores de decisão simples sobre os pedaços do conjunto de dados (CUI *et al.*, 2018).

Sobre o resultado da classificação obtido em dados não rotulados, o algoritmo RF disponibiliza uma entrada para todas as árvores que constituem a floresta e, a partir das saídas das árvores, tem-se um método de votação para decidir o resultado. Caso for um problema de regressão linear, calcula-se uma média dos resultados das árvore e obtêm-se o resultado final e, se for um problema de classificação, a classe eleita é o que a maioria das árvores decidiram.

Ainda, o RF é propenso apenas a *overfitting* local, diferentemente do DT, o qual não influencia no cenário total retratado pelos dados, em que cada árvore é responsável por um pedaço do conjunto de dados, sendo impossível que determinada árvore distribua perfeitamente a totalidade dos mesmos. Portanto, a abordagem adotada no RF adiciona uma diversidade mais ampla no processo de classificação, o que pode implicar em um melhor resultado tanto por parte do modelo, quanto por parte das predições.

2.2.3. *Gradient Boosting Machine*

O algoritmo *Gradient Boosting Machine* (GBM), também chamado de *Gradient Boosting Decision Tree* (GBDT), é similar ao RF, contudo ele aplica o método *boosting*, que se refere a qualquer método *ensemble* que combina vários modelos fracos em um forte (GÉRON, 2019). Dessa forma, há o treino sequencial dos classificadores, onde cada classificador tenta corrigir o seu antecessor. Logo, no GBM tem-se árvores sequenciais, em que cada árvore corrige sua antecessora, fazendo ajustes por meio dos erros residuais, o qual é calculado pelo valor observado menos o previsto.

Além disso, com o avanço da DT, o GBM é capaz de evitar melhor a possibilidade de *overfitting* no modelo e, ainda, quanto mais estimadores utilizados, menor a chance de *overfitting*. Em conclusão, o GBM tem uma robustez superior, além do fato de ser menos provável que seja influenciado pela escala dos dados de treino, além das características irrelevantes e os *outliers* dificilmente alterarem seu desempenho (CUI *et al.*, 2018).

2.2.4. XGBoost

O *Extreme Gradient Boosting* (XGBoost) foi proposto por Chen; Guestrin (2016), e constitui-se de um algoritmo baseado no método *boosting* com uma alta escalabilidade,

o qual está entre os algoritmos ganhadores das competições na plataforma Kaggle. Porém, em casos que os dados tabulares forem de pequeno a médio porte, os algoritmos baseados no DT são considerados melhores.

O XGBoost ganhou destaque por conta da sua alta escalabilidade em todos os cenários, a qual possibilita sua execução desde um computador comum até configurações distribuídas para bilhões de exemplos. Além disso, a computação paralela e distribuída torna a aprendizagem mais veloz, permitindo uma exploração mais rápida do modelo. Em complemento, o mais interessante desse algoritmo é a exploração da computação fora do núcleo, a qual proporciona que cientistas de dados processem centenas de milhões de exemplos em um desktop.

É importante destacar que o XGBoost apresenta métodos para evitar *overfitting*, sendo o objetivo de aprendizagem regularizado, o que contribui na suavização dos pesos finais aprendidos, e tende a selecionar um modelo que aplica funções preditivas e simples; o encolhimento, o qual diminui a influência das árvores individuais e abre espaço para as árvores mais novas deixarem o modelo melhor; e a subamostragem da coluna, que, além de prevenir *overfitting*, permite um aumento na velocidade de processamento ao efetuar os cálculos no algoritmo em paralelo. Por fim, para obter o máximo de desempenho no XGBoost, é preciso ajustar os hiperparâmetros, que são variáveis definidas antes do treino ocorrer e fazem o controle da própria etapa de aprendizado, especialmente os seguintes:

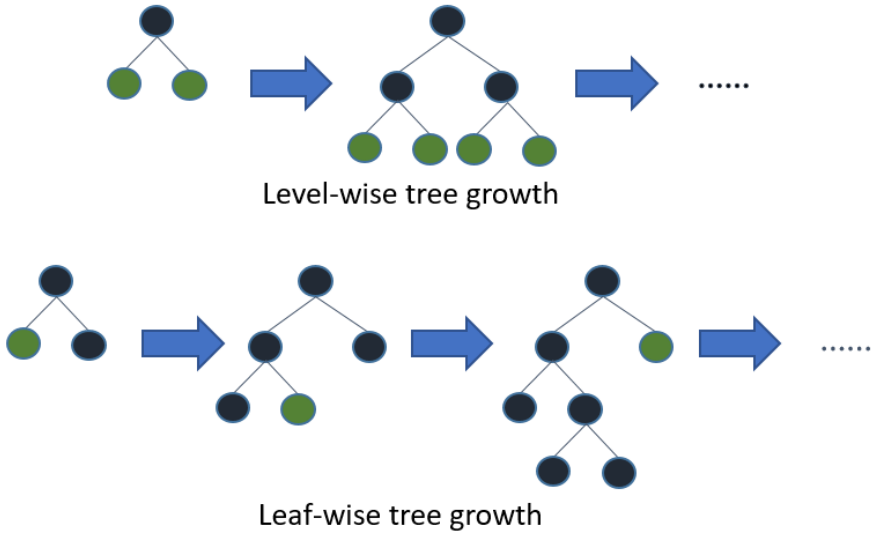
- ***learning_rate:*** também conhecido como *eta*, refere-se a taxa de aprendizagem, a qual é a definição do tamanho do passo utilizado na atualização para impedir o *overfitting*. Depois de cada etapa do *boosting*, obtém-se de modo direto os pesos das características novas;
- ***n_estimators:*** número de árvores com *gradient boosting*. Equivale ao número de rodadas de reforço;
- ***max_depth:*** profundidade máxima da árvore. Quanto mais profunda a árvore, o modelo torna-se mais complexo e aumenta a chance de *overfitting*;
- ***subsample:*** proporção de subamostra das instâncias de treinamento;
- ***gamma:*** redução de perda mínima necessária para fazer uma partição em um nó folha da árvore. Quanto maior for o *gamma*, o algoritmo será mais conservador;
- ***reg_lambda:*** termo de regularização L2 sobre pesos. Quanto mais aumentar esse valor, mais conservador torna-se o modelo.

2.2.5. LightGBM

O *Light Gradient Boosting Machine* (LightGBM), apresentado por Meng *et al.* (2016) e desenvolvido pela Microsoft, constitui-se de um algoritmo que aplica o *gradient boosting*, que foi projetado para ter uma alta eficiência e velocidade, baixa utilização da memória, melhor acurácia, suporte para aprendizagem em GPU e paralelo, além de poder lidar com grandes conjuntos de dados. O objetivo fundamental do LightGBM é acelerar o processamento do treinamento, oferecendo o mesmo desempenho do XGBoost e, para isso, conta-se com otimizações durante a construção das árvores (MICROSOFT, 2019).

O crescimento da árvore no LightGBM é diferente dos outros algoritmos, pois cresce verticalmente, ou seja, cresce em termos de folha, denominando-se *leaf-wise tree growth*, enquanto em outros algoritmos cresce horizontalmente, isto é, cresce em termos de nível, nomeando-se *level-wise tree growth*. O LightGBM seleciona a folha com uma perda maior para crescer e assim pode-se diminuir mais perdas do que um algoritmo com base no crescimento horizontal de árvores. Na Figura 5, é ilustrada uma comparação entre as duas formas de crescimento em árvore.

Figura 5: Crescimento horizontal e vertical da árvore



Fonte: Extraído de Microsoft Corporation, 2021

E, ainda, o LightGBM utiliza duas técnicas para obter todas as vantagens citadas antes, sendo elas a *Gradient-based One-Side Sampling* (GOSS), que é responsável pela amostragem de dados; e a *Exclusive Feature Bundling* (EFB), a qual reduz o número de características em conjuntos de dados esparsos durante o treino. Finalmente, o LightGBM conta com mais de 100 hiperparâmetros e, para ter o máximo de desempenho e evitar o

overfitting, assim como foi mencionado no algoritmo XGBoost, é necessário o ajuste de determinados hiperparâmetros, principalmente os seguintes:

- ***lambda_l1:*** termo de regularização L1, o qual penaliza a soma dos pesos;
- ***lambda_l2:*** termo de regularização L2, sendo responsável por penalizar soma do quadrado dos pesos;
- ***num_leaves:*** número máximo de folhas em uma árvore;
- ***feature_fraction:*** seleciona de modo aleatório um subconjunto de características em cada iteração (árvore). Caso o parâmetro for definido em 0,8, é escolhido 80% das características antes de treinar cada árvore. Vale ressaltar que pode ser usado para acelerar o treinamento e impedir *overfitting*;
- ***bagging_fraction:*** refere-se à fração de dados que será utilizada em cada iteração e geralmente é utilizado para evitar *overfitting* e acelerar a velocidade de treino;
- ***bagging_freq:*** frequência para *bagging*;
- ***min_child_samples:*** número mínimo de dados em uma folha e pode ser aplicado para lidar com o *overfitting*.

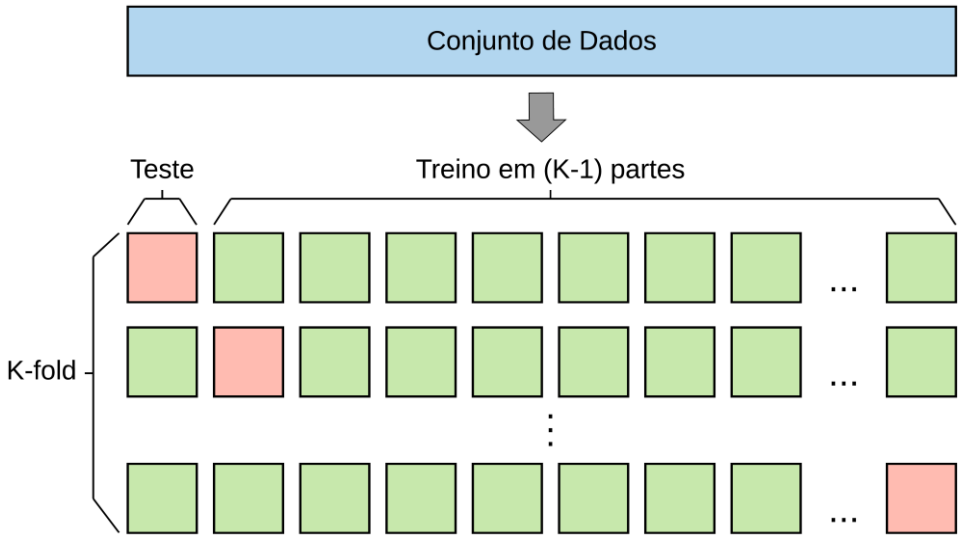
2.3. Técnicas de Validação do Modelo

Durante o desenvolvimento de um modelo de ML, uma parte essencial é a validação do modelo nas etapas de treinamento e teste e, para isso, tem-se duas técnicas: *hold-out* e validação cruzada. A *hold-out* consiste basicamente na divisão do conjunto de dados em duas partes: 80% dos dados para a etapa de treino e 20% para a etapa de teste. Outro valor aplicado para as etapas de treinamento e teste é 70% e 30%, respectivamente. Portanto, na *hold-out*, há o ajuste do modelo nos dados de treino e, depois, o modelo tenta predizer determinada classe nos dados de teste (RASCHKA, 2018).

Por sua vez, a técnica da validação cruzada, também conhecida como K-*Fold cross-validation*, é, com certeza, a técnica mais utilizada, e seu propósito primordial é que cada amostra do conjunto de dados possa ser testada. Dessa forma, efetua-se a validação cruzada, a qual é iterada *K* vezes sobre um conjunto. Em cada iteração, tem-se a divisão do conjunto de dados em *K* partes, em que o treino é executado em $K - 1$ partes, e o teste na parte não usada pelo treinamento. Um dos problemas dessa técnica é que pode resultar em algumas partes terem mais de um tipo de classe do que outra, porque o conjunto de dados é dividido de forma aleatória (RASCHKA, 2018). Desse modo, a validação cruzada estratificada surge como solução, em que a proporção presente no conjunto de dados é

mantida na divisão de cada uma das partes, ou seja, o balanceamento de classes presente no conjunto de dados é mantido nas partes geradas (BREIMAN *et al.*, 1984). Na Figura 6, é mostrado um exemplo genérico do funcionamento da validação cruzada.

Figura 6: Validação cruzada



Fonte: Elaborado pelo autor

2.4. Métricas de Avaliação

A avaliação do desempenho de um modelo é essencial em ML, assim como a etapa de construção do mesmo, pois, com a avaliação, é possível compreender se o modelo está efetuando predições adequadamente. Para isso, existem diversas métricas, como a matriz de confusão, a partir da qual a acurácia, precisão, *recall* e F1-score são extraídas. Ainda, tem-se a curva de Característica de *Receiver Operating Characteristic* (ROC) e a *Area Under the ROC Curve* (AUC) (GÉRON, 2019).

A matriz de confusão consiste de uma tabela aplicada para verificar o desempenho do modelo. Ao utilizar a matriz, é possível observar a quantidade de vezes que o modelo realizou a predição corretamente e erroneamente e, além disso, qual a real classe que o modelo deveria ter feito a predição. Assim, os seguintes termos são definidos: Verdadeiro Positivo (VP), que é prever positivo e ser verdade; Verdadeiro Negativo (VN), o qual é prever negativo e ser verdade; Falso Positivo (FP), que é prever positivo e ser falso; Falso Negativo (FN), o qual é prever negativo e ser falso. Para o trabalho em questão, VP e VN são os domínios maliciosos e legítimos identificados de modo correto, respectivamente; FP são os domínios legítimos classificados como maliciosos de forma incorreta; e FN são os domínios maliciosos classificados como legítimos de modo incorreto.

Sobre as métricas extraídas a partir da matriz de confusão, tem-se: a acurácia, que é a fração das previsões que o modelo acertou, definida pela equação 1; a precisão, a qual é a métrica que responde qual a proporção de identificações positivas está de fato correta, calculada pela equação 2; a *recall*, também denominada de sensibilidade ou revocação, que visa responder qual a proporção de positivos reais foi identificada de forma certa, definida na equação 3; e, por fim, a F1-score, a qual combina precisão e *recall*, em que se calcula a média harmônica entre as duas, descrita na equação 4.

$$Acurácia = \frac{VP+VN}{VP+VN+FP+FN} \tag{1}$$

$$Precisão = \frac{VP}{VP+FP} \tag{2}$$

$$Recall = \frac{VP}{VP+FN} \tag{3}$$

$$F1\text{-}Score = 2 \cdot \frac{Recall \cdot Precisão}{Recall+Precisão} \tag{4}$$

A respeito da curva ROC, trata-se de uma métrica usada para verificar o quanto o modelo é capaz de diferenciar entre as classes e, além disso, é composta de dois eixos, em que, no eixo x, tem-se a Taxa de Falso Positivo (TFP) e, no eixo y, a Taxa de Verdadeiro Positivo (TVP). A TFP é calculada com base na equação 5 ou por $1 - TVN$ e demonstra a taxa de amostras positivas que foram classificadas erradas. Por sua vez, a TVP, também denominada de *recall*, é definida na equação 3 e, como explicado antes, indica a taxa de amostras positivas que foram classificadas corretamente. Além dessas taxas apresentadas, existem: a Taxa de Falso Negativo (TFN), a qual se calcula com base na equação 6 ou $1 - TFP$ e indica a taxa de amostras negativas que foram classificadas erroneamente; e, ainda, a Taxa de Verdadeiro Negativo (TVN), também nomeada de Especificidade (E), a qual é calculada de acordo com a equação 7 e corresponde a taxa de amostras negativas que foram classificadas corretas.

$$TFP = \frac{FP}{VN+FP} \tag{5}$$

$$TFN = \frac{FN}{VP+FN} \tag{6}$$

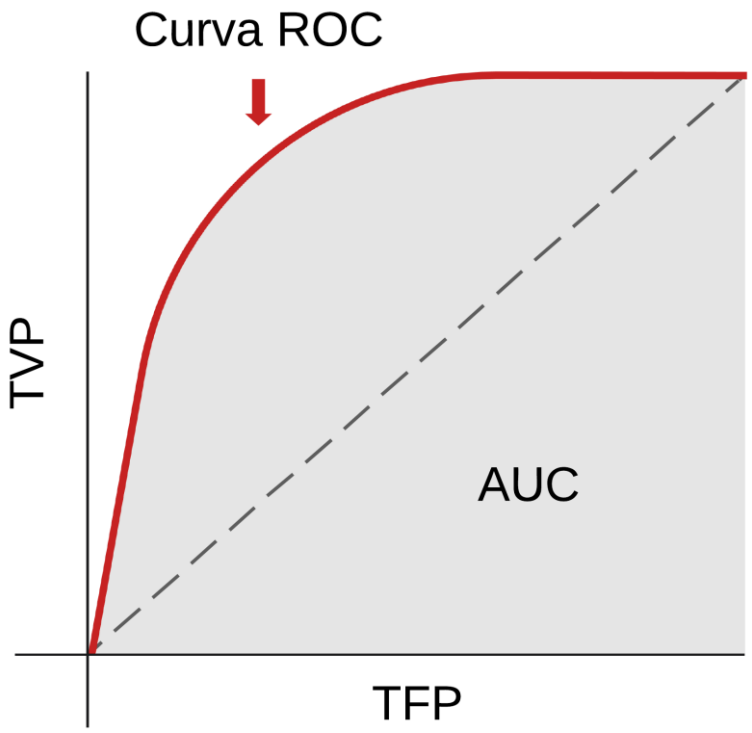
$$TVN = \frac{VN}{VN+FP} \tag{7}$$

A cada *threshold*, tem-se o cálculo da TVP e TFP, e o gráfico é gerado. Ainda, quanto menor o TFP e maior o TVP para cada *threshold*, ou seja, quanto mais próximo a curva estiver do canto superior esquerdo, melhor é o modelo. Com o objetivo de facilitar

o compreendimento da curva ROC, tem-se AUC, a qual consiste no cálculo da área sob a curva e é uma técnica útil para a visualização do desempenho dos classificadores, além de estar sendo cada vez mais utilizada nas comunidades de ML (FAWCETT, 2004).

A respeito do valor da AUC, varia no intervalo [0,0; 1,0], e o limiar é 0,5 e assim, quanto mais a curva ROC estiver no canto superior esquerdo, maior será a AUC, e, ainda, uma AUC de 0,5 quer dizer que o modelo classifica as classes aleatoriamente. Por fim, a AUC é indicada quando há uma preocupação igual entre as classes, ou seja, preocupação com os VPs tanto quanto com os VNs e, por conta disso, é muito utilizada para avaliar modelos de classificação binária e foi escolhida como métrica para comparar os modelos neste trabalho. Na Figura 7, é ilustrada a curva ROC e AUC, além dos eixos TVP e TFP e da linha pontilhada que indica um modelo com taxa de acerto de 50%. (GÉRON, 2019).

Figura 7: Curva ROC e AUC



Fonte: Elaborado pelo autor

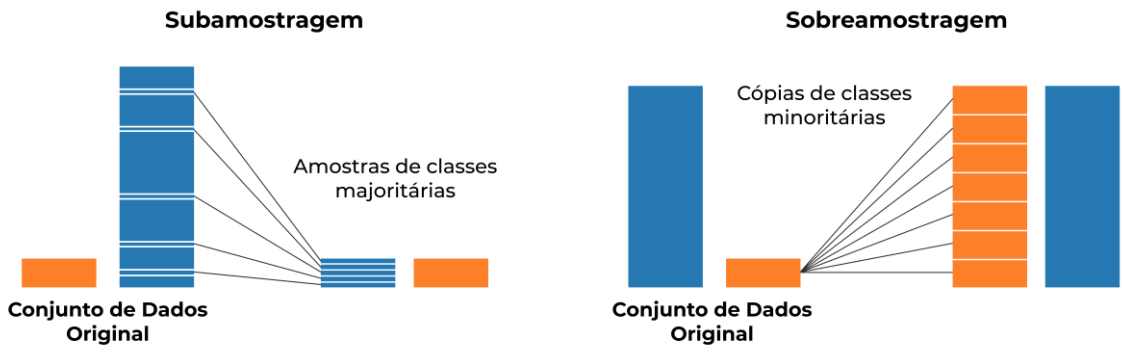
É interessante ressaltar que, se as classes do modelo estiverem desbalanceadas, não se pode utilizar métricas de avaliação como acurácia e AUC. Por exemplo, caso o modelo seja capaz de classificar somente a classe majoritária, o resultado seria uma alta acurácia. Além disso, a TFP é reduzida devido ao grande número de VN na AUC. Quando há a presença de classes desbalanceadas, é indicado uso da curva de Precisão-*Recall* (PR) e PR AUC (DAVIS; GOADRICH, 2006).

2.5. Balanceamento de Classes

O problema do desbalanceamento de classes acontece quando há um conjunto de dados que possui uma quantidade muito superior de determinadas classes em comparação com as demais. O desbalanceamento de classes é comum em situações que envolvem a detecção de fraudes, como a detecção de domínios maliciosos em que é normal a presença de mais classes legítimas do que classes maliciosas. Esse problema conduz os algoritmos de classificação padrão em tenderem para as classes majoritárias e, portanto, ocorre uma taxa de classificação errônea dos dados que são de classes minoritárias. Como solução, surgem os métodos de reamostragem de classes, nos quais se aplica uma etapa de pré-processamento para equilibrar as classes, ou seja, os dados de treinamento são alterados de maneira a produzir uma distribuição de classes mais igualitária, e, com isso, tem-se o balanceamento de classes. Além disso, a vantagem de usar os métodos de reamostragem de classes é que eles são independentes dos algoritmos de classificação em utilização. Por fim, esses métodos são divididos em três famílias: sobreamostragem, subamostragem, e os métodos híbridos (BATISTA *et al.*, 2004).

Os métodos de subamostragem basicamente criam um subconjunto do conjunto de dados original, removendo dados que geralmente são dados da classe majoritária. Por sua vez, os métodos de sobreamostragem formam um superconjunto do conjunto de dados original, replicando alguns dados da classe minoritária ou, ainda, criando novos dados da classe minoritária a partir dos existentes, ou seja, geração de dados sintéticos. Por último, nos métodos híbridos, tem-se a combinação de ambos os métodos que foram apresentados anteriormente (FERNÁNDEZ *et al.*, 2018). Como forma de exemplificar, na Figura 8, estão ilustrados os métodos de subamostragem e sobreamostragem, respectivamente.

Figura 8: Subamostragem e sobreamostragem



Fonte: Adaptado de Liu *et al.*, 2019

É importante destacar que, ao utilizar os métodos de sobreamostragem, é possível trazer *overfitting* no modelo, uma vez que os dados são replicados ou gerados a partir dos dados originais. Além disso, os métodos de subamostragem podem resultar na perda de dados importantes para o treinamento do modelo, visto que dados da classe majoritária são eliminados na reamostragem. Assim, surge o conceito de classificador simbólico, o qual tem regras que são visivelmente precisas, mas cobrem somente os dados replicados. Como soluções, é preciso um controle no valor da proporção de balanceamento, utilizar métodos de sobreamostragem e subamostragem mais robustos, isto é, em que se aplicam abordagens heurísticas, além de métodos adequados para cada situação.

Neste trabalho, foram aplicados métodos de subamostragem e sobreamostragem, que englobam desde os mais simples até os mais robustos, com a finalidade de combinar os métodos, devido à desproporção entre as classes dos conjuntos de dados, e fornecer uma comparação. Portanto, cada método usado é discutido nas seções a seguir.

2.5.1. Métodos de Subamostragem

No trabalho, utilizou-se dois métodos de subamostragem, um mais simplista e outro que usa abordagem de aprendizado não supervisionado. O *Random Undersampling* (RUS) consiste em um método não heurístico que visa o equilíbrio das classes por meio da eliminação aleatória de dados da classe majoritária e, com isso, o balanceamento de classes é obtido. Apesar de simples, o RUS apresenta ótimos resultados com um tempo de resposta baixo (BATISTA *et al.*, 2004).

A respeito do *Cluster Centroids* (CC), detalhado por Yen; Lee (2009), aplica-se o algoritmo de agrupamento K-*Means*, que consiste em uma abordagem de aprendizado não supervisionado. Assim, o CC encontra os centroides dos *clusters*, em que cada *cluster* é gerado abrangendo os dados pertencentes à classe majoritária, visto que as classes estão rotuladas. Após obter os centroides de cada *cluster* da classe majoritária, os dados que se encontram mais longes do centroide são eliminados do *cluster*, ou seja, considera-se esses dados menos importantes. Caso o conjunto de dados esteja extremamente desbalanceado, por exemplo, 1% dos dados são da classe minoritária, é comum usar apenas os centroides como dados da classe majoritária.

2.5.2. Métodos de Sobreamostragem

A respeito dos métodos de sobreamostragem, usou-se cinco métodos no trabalho, um simples e quatro que utilizam algoritmos de ML. O *Random Oversampling* (ROS) é

um outro método não heurístico que equilibra as classes por meio da replicação aleatória de dados da classe minoritária. Assim como o RUS, apesar de ser simples, alcançou bons resultados com um tempo de resposta baixo (BATISTA *et al.*, 2004).

O método *Synthetic Minority Oversampling Technique* (SMOTE), proposto por Chawla *et al.* (2002), utiliza o algoritmo KNN para gerar os dados sintéticos. O SMOTE inicia selecionando dados aleatórios da classe minoritária e, depois, define os K vizinhos mais próximos e seleciona um dos vizinhos de forma aleatória, e um dado sintético é gerado entre os vizinhos. O processo descrito antes é repetido até que a classe minoritária possua a mesma proporção da classe majoritária.

Por sua vez, o método *Borderline*-SMOTE, apresentado por Han *et al.* (2005), é uma extensão do SMOTE, o qual cria dados sintéticos ao longo da fronteira de decisão entre as classes, em vez de criar de forma aleatória entre os vizinhos. Ainda, existem dois tipos desse método: o *Borderline*-SMOTE 1, em que é efetuada uma sobreamostragem nos dados da classe majoritária que estão ocasionando erros de classificação na fronteira de decisão; e o *Borderline*-SMOTE 2, que somente sobreamostra a classe minoritária.

Uma variação do *Borderline*-SMOTE é o *Borderline*-SMOTE SVM, ou apenas SVM-SMOTE, apresentado por Nguyen *et al.* (2011), que utiliza o algoritmo SVM, em vez do KNN, para a identificação da classificação errônea. Assim, a área da fronteira de decisão é aproximada pelos vetores de suporte após o treinamento do classificador SVM no conjunto de dados original. Os dados sintéticos são gerados aleatoriamente no decorrer das linhas que conectam cada vetor de suporte da classe minoritária com os seus vizinhos mais próximos. A vantagem do SVM-SMOTE, em comparação ao *Borderline*-SMOTE, é que mais dados são sintetizados distantes do local de sobreposição das classes, isto é, o foco maior é onde os dados são separados.

Por fim, o método K-*Means* SMOTE, proposto por Douzas *et al.* (2018), consiste da aplicação do algoritmo de agrupamento K-*Means* antes de aplicar SMOTE e, além disso, o método divide-se em três etapas: agrupamento, filtragem e sobreamostragem. Na etapa de agrupamento, os dados da classe minoritária agrupam em K *clusters* por meio do K-*Means*. Na filtragem, é escolhido os *clusters* para a sobreamostragem, mantendo os *clusters* que têm uma alta proporção de dados da classe minoritária e, depois, distribui-se o número de dados sintéticos que serão gerados e, caso o *cluster* possuir dados da classe minoritária distribuídos esparsamente, mais dados sintéticos são gerados. Finalmente, na

sobreamostragem, aplica-se o método SMOTE em cada *cluster*, atingindo a proporção pretendida de dados das classes minoritária e majoritária.

Além dos métodos de sobreamostragem, aplicados no trabalho, têm-se outros métodos de sobreamostragem baseados no SMOTE, como SMOTE-NC, o qual é possível ter características numéricas e categóricas; SMOTEN, que espera que os dados tenham somente características categóricas; e ADASYN, em que dados sintéticos são criados com base na densidade dos dados e, com isso, a composição do dado sintético é inversamente proporcional à densidade da classe minoritária, isto é, mais dados sintéticos são criados em locais que a densidade da classe minoritária é baixa ou não alta.

2.6. Trabalhos Correlatos

Existem vários trabalhos que aplicam características presentes na coleta de DNS passivo para o treinamento dos modelos de ML para a detecção de domínios que estão vinculados em atividades maliciosas. Como explicado antes, a coleta de DNS passivo constitui-se em armazenar dados anonimamente relativos a um certo domínio relacionado a um registro DNS em particular (WEIMER, 2005). Os trabalhos, em geral, distinguem o algoritmo de ML aplicado, a base de dados pDNS, o uso dos métodos de balanceamento de classes e o local de coleta do tráfego DNS, sendo possível a coleta em servidores DNS autoritativos ou recursivos. Dessa forma, na sequência, são apresentados brevemente os trabalhos correlatos ao sistema proposto neste presente documento. Os três primeiros trabalhos marcam o início da detecção de domínios maliciosos por meio de DNS passivo, sendo a base para o restante dos trabalhos. Para a procura desses trabalhos, utilizou-se as seguintes fontes bibliográficas para a pesquisa: ACM Digital Library; Google Scholar; IEEE Xplore; ScienceDirect; e Scopus.

Antonakakis *et al.* (2010) apresentam o Notos, o primeiro sistema de reputação dinâmico para DNS, em que o objetivo principal é atribuir de modo automático uma baixa pontuação a um domínio malicioso e, de modo contrário, alta pontuação a um domínio legítimo. Notos foi avaliado em uma rede de um ISP com tráfego DNS de 1,4 milhões de usuários, em que detectou domínios maliciosos com alta precisão com uma TVP de 96,8% e uma baixa TFP de 0,38%. Portanto, conclui-se que o sistema pode identificar domínios maliciosos semanas ou até meses antes de surgirem em *blocklists* públicas.

Em Bilge *et al.* (2011), é proposto o Exposure, um sistema que emprega técnicas de análise de DNS passivo em grande escala em um servidor DNS recursivo para detectar

domínios envolvidos em atividades maliciosa e, de modo distinto do Notos, a abordagem requer menor tempo de treinamento e menos dados de treino e não tem algumas de suas limitações. No trabalho, utilizou-se 15 características, das quais nove eram novas e, com isso, foi possível a caracterização das distintas propriedades do DNS, além das formas de consultas. Os experimentos com 100 bilhões de solicitações de DNS, além da instauração em um ISP por duas semanas, apontaram que o Exposure é escalável e capaz de identificar domínios maliciosos utilizados em ataques, como C&C de *botnet*, spam e *phishing*.

Em Antonakakis *et al.* (2011), é apresentado o Kopis, um sistema para detectar nomes de domínio relacionados a *malware*. O sistema realiza o monitoramento do tráfego DNS passivo nos níveis superiores da hierarquia DNS, o que garante uma visibilidade global maior. Em comparação com o Notos e Exposure, que coletam os dados de pDNS de servidores DNS recursivos, o Kopis adiciona a vantagem em relação às características, que são capazes de ofertar uma visibilidade global por conta do monitoramento do tráfego de rede na hierarquia superior do DNS. Ainda, o Kopis identifica domínios envolvidos em *malware* não possuindo informação sobre a reputação de IP. Por fim, o sistema foi utilizado para detectar a criação de um *botnet* de DDoS na China.

Lison; Mavroeidis (2017) apresentam um modelo de ML com a capacidade de detectar de forma automática se os nomes de domínio e endereços IP são legítimos ou maliciosos, em que o modelo necessita de uma arquitetura de aprendizagem profunda e é treinado em um conjunto de dados de DNS passivo. De acordo com os autores, o sistema é pioneiro no emprego de redes neurais profundas para desenvolver modelos dinâmicos de reputação. Finalmente, o modelo tem a capacidade de detectar domínios maliciosos com F1-score de 0,96.

Bao *et al.* (2019) utilizam XGBoost para a detecção de domínios referentes à DGA multidimensional por meio de DNS passivo, que considera várias características usadas na detecção de DGA e melhora a precisão na identificação desses domínios e, além disso, apresentam um método para detectar nome de domínio pornográfico baseado no vetor de palavras combinado com um ambiente de rede chinês. Por fim, os autores implementaram um protótipo do sistema para detectar nomes de domínios maliciosos e obtiveram bons resultados nos testes.

No trabalho de Wang *et al.* (2020), é apresentado o KSDom, que consiste em um sistema de detecção de domínio, o qual utiliza a técnica K-*Means* SMOTE para lidar com o desbalanceamento de classes e *Categorical Boosting* (CatBoost) como algoritmo de

classificação, que é excelente ao trabalhar com características categóricas. Os resultados indicaram que o KSDom é robusto em conjuntos de dados desequilibrados com distintas proporções, principalmente quando a proporção de desequilíbrio das classes do conjunto de dados atinge 10:1.

Em Silveira *et al.* (2020), os autores utilizam apenas o tráfego DNS como fonte de dados para detecção de domínios maliciosos, extraindo 15 características exclusivas de dados de pDNS e, ainda, aplicam o algoritmo XGBoost para treinamento do modelo e o método de subamostragem para balanceamento de classes. Para obter os melhores hiperparâmetros no XGBoost, a otimização Bayesiana foi utilizada com 21 pontos iniciais e 500 iterações. Ainda, os autores executaram testes em dez subconjuntos diferentes, em que se notou uma baixa variância, e o modelo não possui *overfitting*. Por fim, o modelo provou realmente ser efetivo com uma AUC média de 0,9763.

Finalmente, no trabalho de Zhang *et al.* (2021), é proposto o GAMD, o qual é uma rede neural de grafos heterogêneos semi-supervisionada para a identificação de domínios maliciosos. Os experimentos realizados na rede da universidade mostraram que o GAMD é apropriado para conjunto de dados em larga-escala com classes limitadas. Os autores trabalharam apenas com os tipos de nós *host*, domínio e IP resolvido, contudo pretendem estender para a adição de mais dados como parte dos atributos, como CNAME, segmento de rede, dados WHOIS e registro.

2.7. Considerações Finais

Neste capítulo, foi apresentada toda a fundamentação teórica para o entendimento do trabalho, iniciando-se com a explicação do protocolo DNS e sua hierarquia, além dos RRs, como funciona a coleta de DNS passivo e os principais ataques envolvendo DNS. Depois, foram apresentados conceitos de ML e algoritmos e, na sequência, tratou-se sobre as técnicas de validação do modelo e métricas de avaliação, além dos métodos aplicados para o balanceamento de classes.

Dos trabalhos correlatos apresentados, somente dois trabalhos (BAO *et al.*, 2019; SILVEIRA *et al.*, 2020) usam XGBoost para detectar domínios maliciosos, e, em Bao *et al.* (2019), é exclusivamente para a detecção de DGA. O restante dos trabalhos apoia-se em outros algoritmos baseados em árvores, como o DT (ANTONAKAKIS *et al.*, 2010; BILGE *et al.*, 2011), RF (ANTONAKAKIS *et al.*, 2011) e CatBoost (WANG *et al.*, 2020), além das redes neurais (LISON; MAVROEIDIS, 2017; ZHANG *et al.*, 2021). A

respeito dos métodos de balanceamento de classes, somente três trabalhos mencionam o aproveitamento desses métodos, sendo eles *downsampling* (BAO *et al.*, 2019), K-*Means* SMOTE (WANG *et al.*, 2020) e RUS (SILVEIRA *et al.*, 2020). Por último, apenas, no Kopis (ANTONAKAKIS *et al.*, 2011), é usado dados de um servidor DNS autoritativo, em que se incluem informações do servidor TLD ".ca" e, além disso, em três trabalhos (ANTONAKAKIS *et al*., 2010; ANTONAKAKIS *et al*., 2011; SILVEIRA *et al*., 2020), são usadas somente características numéricas para a obtenção dos resultados.

Desse modo, este documento propõe um sistema para a classificação de domínios recém-registrados por meio de DNS passivo, em que os dados são coletados do servidor autoritativo de um TLD, além de utilizar o algoritmo LightGBM, que não foi explorado nos trabalhos e possui alta eficiência e velocidade (MENG *et al.*, 2016), e de métodos de balanceamento mais recentes, como K-*Means* SMOTE (DOUZAS *et al.*, 2018).

Capítulo 3 – Desenvolvimento

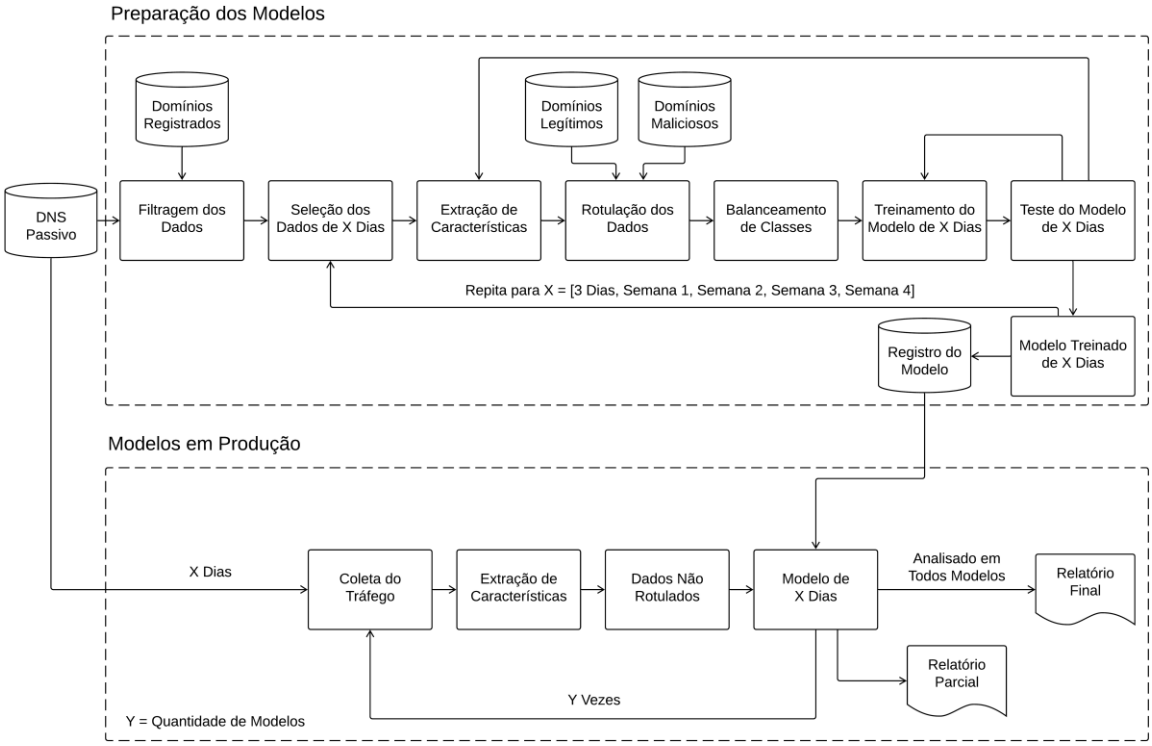
3.1. Descrição Geral

Uma vez que o objetivo deste trabalho é a elaboração de um sistema eficaz para classificar domínios recém-registrados no primeiro mês de vida a partir da primeira *query*, é preciso aplicar técnicas de ML para a construção dos classificadores, utilizando dados de pDNS. Assim, nesta seção, bem como nas posteriores, é realizado o detalhamento da metodologia do sistema, a qual se divide em duas partes:

- **Preparação dos Modelos:** consiste na filtragem e seleção dos dados, extração de características, rotulação dos dados, balanceamento de classes, treinamento e teste de Y modelos de ML, em que cada modelo corresponde a um período de dias em que foi coletado o tráfego DNS dos domínios recém-registrados. Sobre o valor de Y, definiu-se como cinco, ou seja, cinco modelos e, com isso, há os Modelos de Três Dias (M3D) e Semana 1 (MS1), em que o MS1 corresponde aos sete dias iniciais. Além disso, o sistema dispõe dos Modelos da Semana 2 (MS2), Semana 3 (MS3) e Semana 4 (MS4);
- **Modelos em Produção:** corresponde ao monitoramento de determinado domínio recém-registrado em cada um dos Y modelos construídos anteriormente, em que, no final de cada classificação, há um relatório parcial obtido em cada modelo e, no fim, há o relatório final. Diante disso, têm-se a análise por completo do domínio em todo seu primeiro mês de vida após o registro.

Com isso, foi elaborado o diagrama geral do funcionamento do sistema mostrado na Figura 9, em que, nas seções seguintes, são detalhadas as etapas apresentadas.

Figura 9: Diagrama do funcionamento do sistema



Fonte: Elaborado pelo autor

3.2. DNS Passivo

Conforme apresentado na Figura 9, o conjunto de dados é obtido da coleta de 12 meses do tráfego DNS do servidor autoritativo de um TLD por meio de DNS passivo, que começou no início de março de 2020. O sistema de coleta utiliza o ENTRADA, que constitui em um *data streaming* de alto desempenho, proporcionando a análise de grandes volumes de tráfego de rede em segundos até poucos minutos (WULLINK *et al.*, 2016).

Assim, inicialmente, é efetuada a coleta do tráfego de rede em arquivos de formato pcap, além da inserção das instâncias no ENTRADA, o qual limpa, enriquece e compacta os dados e, em seguida, insere no *Cluster* Hadoop. As solicitações DNS são combinadas com as suas respectivas respostas, e os dados dos cabeçalhos IP, TCP, UDP ou DNS são armazenados. Após isso, o processo de enriquecimento é efetuado, no qual são gerados os campos do código do país e ASN por meio do banco de dados de GeoIP (Maxmind)[2].

[2] Fornecimento de dados geográficos e outras informações associadas a endereços de protocolos específicos da Internet (MAXMIND, 2021).

Por isso, o conjunto de dados, a princípio, tem todos os campos presentes no sistema de coleta do ENTRADA. Após o processamento dos dados, os arquivos são armazenados no formato Apache Parquet, o que proporciona um maior desempenho e baixa utilização de armazenamento. Em Wullink *et al.* (2016), é constatado que, ao trabalhar com os arquivos em Parquet, há uma redução de até 93% quando comparados com os pcaps brutos.

3.3. Filtragem e Seleção dos Dados

O sistema inicia com a filtragem dos dados, em que se consulta uma lista de domínios registrados, fornecida pelo *registry* responsável pelo TLD, para filtrar somente as *queries* dos domínios recém-registrados. Após isso, há a seleção dos dados de X dias, em que X corresponde ao período de três ou sete dias iniciais de vida dos domínios, ou ainda, das próximas três semanas de vida dos domínios, em que cada X corresponde à preparação de um modelo em específico.

Desse modo, ao fazer a coleta de tráfego DNS e filtrar com uma lista, a qual continha 1.348.938 domínios recém-registrados no intervalo de 01 de março de 2020 até 31 de dezembro de 2020, resultou-se em 1.304.893 *queries*. Ressalta-se que o período de coleta dispõe o tempo da lista mais dois meses extras, porque os domínios geralmente são registrados e demoram até serem utilizados, baseado nas análises dos dados.

3.4. Extração de Características

Na extração de características, é feita a agregação das *queries* de cada domínio e a extração de suas características, que estão na Tabela 1. A maioria das características foram baseadas em trabalhos anteriores (LISON; MAVROEIDIS, 2017; KHALIL *et al.*, 2016; WATKINS *et al.*, 2017). Porém, cinco características são exclusivas deste trabalho, as quais estão marcadas com asterisco (*), visto que o sistema é voltado para os domínios recém-registrados, além do ENTRADA ser aplicado para a coleta dos dados.

Após a análise em algumas características, destaca-se que os domínios maliciosos tendem a ser usados logo após o registro, e o número de dias coletados é menor. Ainda, o campo contagem de *Query Names* (QNAMEs) distintos refere-se ao número de *aliases* no domínio. A respeito de alguns campos existentes no ENTRADA, o campo protocolo indica se foi utilizado TCP (6) ou UDP (17), e faz-se uma média para ver qual foi mais usado na solicitação. O *Authoritative Answer* (AA) indica se uma resposta é autoritativa ou não, e o *Checking Disabled* (CD) indica se o *Domain Name System Security Extensions*

(DNSSEC) está desabilitado ou não. Os campos *Answer Count* (ANCOUNT), *Additional Information Count* (ARCOUNT) e *Authority Count* (NSCOUNT) referem a contagem de respostas, dados adicionais e de autoridade, respectivamente. O campo *Response Code* (RCODE) é 0 (zero) – *No Error* ou 3 (três) – *Non-Existent Domain* (NXDomain), e, se não dispor nenhuma resposta, é −1 (menos um). Por fim, o *Query Type* (QTYPE) expõe o RR, e o QNAME *labels* é o número de rótulos no QNAME, por exemplo, site.com.br. contém 3 rótulos.

Tabela 1: Características extraídas do pDNS e suas descrições

Característica	Descrição
nb_days_until_collect*	Número de dias do registro até a primeira consulta.
nb_days*	Número de dias coletados.
nb_domain_queries	Número de consultas para o domínio.
nb_qnames	Contagem de QNAMEs distintos.
min_ttl	TTL mínimo.
ttl_changes	Contagem de mudanças no TTL.
avg_prot*	Média do campo protocolo.
nb_ips	Contagem de endereços IP distintos.
frequent_aa	AA mais frequente.
frequent_cd*	CD mais frequente.
avg_ancount	Média do campo ANCOUNT.
avg_arcount	Média do campo ARCOUNT.
avg_nscount	Média do campo NSCOUNT.
frequent_rcode	RCODE mais frequente.
avg_qtype	Média do campo QTYPE.
nb_countries	Contagem de países distintos.
frequent_country	País mais frequente.
nb_asns	Contagem de ASNs distintos.
avg_labels*	Média do campo QNAME *labels*.
avg_res_len	Média do tamanho da mensagem de resposta DNS.

Fonte: Elaborado pelo autor

Portanto, extraiu-se 20 características do DNS passivo junto com as características de geolocalização vindas das colunas enriquecidas no ENTRADA. Além disso, nota-se o não uso de características textuais, em que algumas são empregadas apenas para auxiliar a etapa de rotulação dos dados, como nome de domínio, primeiro e último dia de coleta, entretanto, na etapa de treinamento, são utilizadas as características da Tabela 1. Após a extração de características, resultou-se em 89.988 domínios aos dados de três dias e da primeira semana, 2.409 da segunda semana, 2.108 da terceira semana e 3.044 da quarta semana. Nota-se que houve uma diminuição brusca na utilização dos domínios após a primeira semana e, por isso, a maior atenção deste trabalho é na detecção em até sete dias.

3.5. Rotulação dos Dados

A respeito da rotulação dos dados, a *blocklist* é disponibilizada pela equipe do *registry* responsável pelo TLD, sendo filtrada em apenas domínios maliciosos registrados no período de coleta, o que resultou em 4.815 domínios. A *allowlist* foi gerada com base na lista de domínios registrados, da qual se eliminou os domínios maliciosos. Ressalta-se que os domínios registrados são analisados cautelosamente pelo *registry* do TLD e, caso não detectado indicadores de maliciosidade nas etapas de pré e pós-registro, os domínios são considerados legítimos. Com isso, os domínios maliciosos são rotulados como 1 (um), e os legítimos como 0 (zero). Por fim, como resultado do processo de rotular os dados, há as seguintes proporções das classes de domínios de cada período:

- **Três dias e primeira semana:** 88.926 legítimos e 1.062 maliciosos;
- **Segunda semana:** 2.368 legítimos e 41 maliciosos;
- **Terceira semana:** 2.078 legítimos e 30 maliciosos;
- **Quarta semana:** 2.991 legítimos e 53 maliciosos.

Vale reforçar que, dos 4.815 domínios maliciosos registrados no período de coleta que estão na *blocklist*, constam apenas 1.062 nos dados de três dias, por exemplo. Tal fato acontece em função do TLD empenhar-se vigorosamente para bloquear e remover esses domínios, antes mesmo que tenham *queries*.

3.6. Balanceamento de Classes

Como a quantia de domínios legítimos é extremamente maior que a de maliciosos, e somente 1,18% dos domínios são maliciosos nos conjuntos de dados de três dias e da primeira semana, é necessário o balanceamento das classes. Além disso, é notado também

o desbalanceamento de classes nos conjuntos de dados da segunda semana em diante ao observar a quantidade de cada classe obtida na rotulação.

Dessa forma, são aplicados dois métodos de subamostragem, em que são RUS e CC, e cinco métodos de sobreamostragem, que são ROS, SMOTE, *Borderline*-SMOTE, SVM-SMOTE e K-*Means* SMOTE. Com isso, é combinado o uso de subamostragem e sobreamostragem, do qual se diminui os dados da classe majoritária para o dobro da classe minoritária e, após isso, iguala-se as duas classes, resultando em 2.124 domínios legítimos e 2.124 maliciosos. Assim, a combinação dos métodos aplicados que conseguir, em geral, uma maior AUC é empregada aos modelos em produção. Os métodos de subamostragem são essenciais para que os modelos não disponham um alto enviesamento dos dados, pois se utiliza algoritmos tradicionais de ML, os quais tendem para classe majoritária se não for aplicado o balanceamento das classes (FERNÁNDEZ *et al.*, 2018).

3.7. Treinamento e Teste dos Modelos

Após balancear as classes, o treinamento dos modelos é feito com os algoritmos DT, RF, GBM, XGBoost e LightGBM, com o objetivo de comparar e ver qual algoritmo obtém o melhor desempenho nos conjuntos de dados. Neste trabalho, optou-se por utilizar esses algoritmos por conta dos trabalhos na Seção 2. alcançarem os melhores resultados com algoritmos baseados em árvores, além do algoritmo XGBoost mostrar sua eficiência em detectar ataques DDoS (CHEN *et al.*, 2018), por exemplo. É interessante frisar que, como esse grupo de algoritmos é utilizado, não é preciso normalizar os dados, pois não sofrem com os dados desnormalizados (BORKIN *et al.*, 2019).

Destaca-se o emprego dos hiperparâmetros padrões da biblioteca scikit-learn[3] para os algoritmos DT, RF e GBM e, para XGBoost e LightGBM, trazem-se os das bibliotecas XGBoost[4] e LightGBM[5], respectivamente. Ainda, no XGBoost e LightGBM, é aplicada a otimização Bayesiana, cujo propósito é encontrar o mínimo global da função no menor número possível de iterações e, com isso, há a seleção dos melhores hiperparâmetros com base na AUC. A partir disso, o desempenho dos modelos cresce, além do *overfitting* ser evitado (SILVEIRA *et al.*, 2020). Para definir o número de pontos iniciais e iterações na otimização Bayesiana, é elaborado testes com os seguintes números de pontos iniciais e

[3] https://scikit-learn.org/stable/
[4] https://xgboost.readthedocs.io/en/stable/
[5] https://lightgbm.readthedocs.io/en/latest/

iterações: 10 e 100; 25 e 250; 50 e 500; 100 e 1000; 250 e 2500; e 500 e 500; e, após isso, avalia-se qual conjunto obteve uma excelente AUC em um tempo de execução viável. Os hiperparâmetros otimizados, no XGBoost, são: *learning rate*, *num. estimators*, *max. depth*, *subsample*, *gamma* e *reg. lambda*; e, no LightGBM, são: *lambda L1*, *lambda L2*, *num. leaves*, *feature fraction*, *bagging fraction*, *bagging freq.* e *min. child samples*.

Além do discutido, com o propósito de identificar a ocorrência de *overfitting* e *underfitting*, é utilizada a técnica de validação cruzada estratificada com $K = 5$, na qual o *K* foi definido com base no tamanho do conjunto de dados. Além disso, ele consiste em um valor interessante para analisar como os modelos se comportam sem que exista uma alta elevação do custo computacional (KOHAVI, 1995). Como métrica de avaliação, há o uso da AUC, em que cada *K* no treino e teste gera uma curva, e, ao final, tem-se a curva e AUC média. Além disso, a F1-score e o tempo de treinamento são métricas também utilizadas. Após validar o modelo, o registro é feito na base de dados dos modelos, e o processo de preparação dos modelos repete-se até que todos os modelos sejam treinados.

3.8. Modelos em Produção

Após a construção de todos os modelos, tem-se o início da coleta de tráfego DNS de determinado domínio a ser analisado em X dias, em que se extrai as características e obtém-se os dados não rotulados e, logo na sequência, é classificado em um dos modelos desenvolvidos e é gerado um relatório parcial de classificação. O processo repete-se até o domínio ser classificado em todos os modelos e, ao final, gera-se o relatório final da classificação com a porcentagem referente ao domínio ser malicioso ou legítimo, e sua respectiva classe. Com base nisso, o sistema auxilia a detectar novos domínios maliciosos recém-registrados, em especial aqueles que contornaram a detecção feita pelo *registry*.

3.9. Considerações Finais

Neste capítulo, foi apresentada a metodologia do sistema para classificar domínios recém-registrados, destacando as etapas fundamentais para desenvolver o sistema, como a obtenção, filtragem e seleção dos dados de pDNS, além da extração de características, treino e teste dos modelos e, enfim, a implantação dos modelos em produção. A execução dessas etapas é essencial para obter os resultados, e assim validar o sistema proposto.

O próximo capítulo apresenta os resultados obtidos após a construção do sistema, bem como as discussões acerca dos resultados.

Capítulo 4 – Resultados

4.1. Hardware e Ferramentas Utilizadas

Com a conclusão do desenvolvimento da metodologia do sistema para classificar domínios recém-registrados, começaram-se os testes para obter os resultados. Para isso, os testes foram empregados em uma máquina com um processador Intel Xeon E52650, com 2.30 GHz (10 núcleos e 20 *threads*), 32 GB de memória RAM e armazenamento de 300 GB (SSD), e o sistema operacional Ubuntu Server 18.04 LTS.

Além disso, usou-se a linguagem de programação Python na sua versão 3.6 para o desenvolvimento do sistema junto às bibliotecas: Jupyter Notebook, como *Integrated Development Environment* (IDE); Apache Spark, para a extração e filtragem dos dados no ENTRADA; Pandas, para o tratamento e transformação dos dados, além da extração de características; imbalanced-learn, para o balanceamento das classes; scikit-learn, para o treinamento e teste dos modelos em geral, além das métricas de avaliação; XGBoost, para o treino do algoritmo XGBoost; LightGBM, para treinamento do LightGBM; e, por fim, bayesian-optimization, para a execução da otimização Bayesiana.

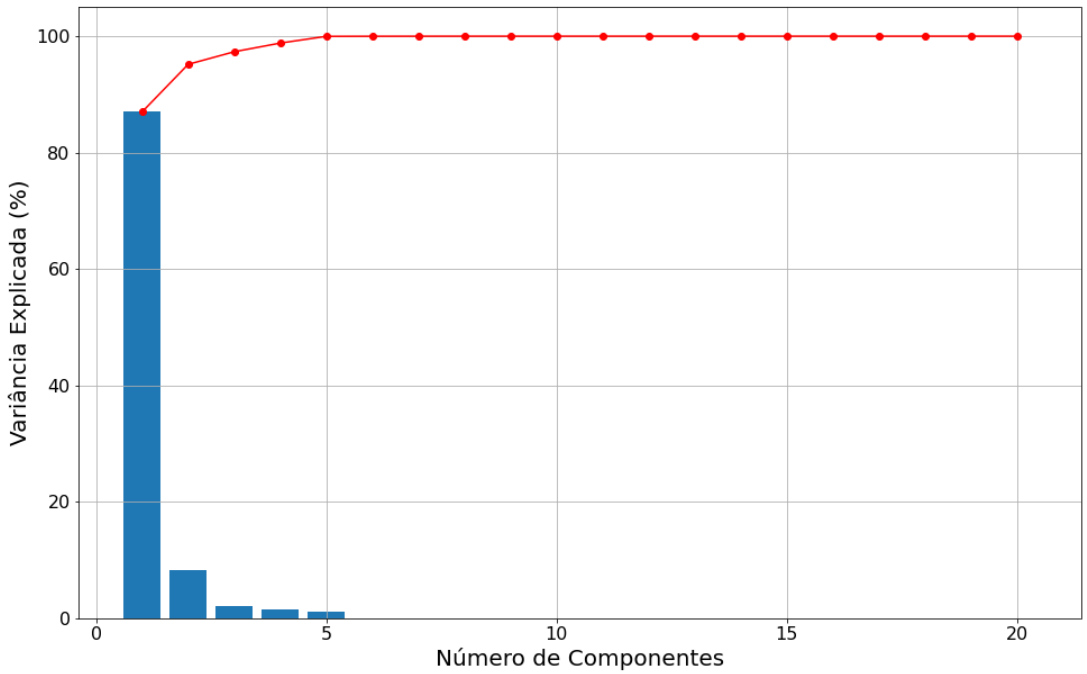
É importante ressaltar a possibilidade de utilizar o Google Colaboratory também, que é um ambiente em nuvem de notebooks Jupyter, executado por meio do navegador, o qual conta com grande parte das bibliotecas citadas anteriormente instaladas, com ponto negativo da execução ser mais lenta devido às limitações de hardware da versão gratuita, incluindo a questão de armazenamento e memória RAM.

4.2. Avaliação dos Métodos de Reamostragem

Como exposto na Seção 3.6, houve a redução dos dados dos domínios legítimos para o dobro dos domínios maliciosos. Outros valores para a taxa de balanceamento foram testados também, como a triplicação dos dados, mas se observou a presença de *overfitting* a partir do momento que há mais dados sintéticos do que reais, com resultados contendo uma AUC próxima de 1. Por conta disso, o máximo foi a duplicação dos dados da classe minoritária, que contribuiu para o desempenho dos modelos, dado a quantia de domínios maliciosos à disposição. A fim de compreender melhor o funcionamento dos métodos de subamostragem utilizados neste trabalho, o algoritmo de redução de dimensão *Principal Component Analysis* (PCA) foi aplicado nos dados de três dias, o qual reduziu-se de 20 para somente três características, ou seja, três componentes principais.

O número de componentes são três devido a possibilidade de visualizar os dados em um gráfico tridimensional, para ver os métodos, além da variância total explicada ser 97,326% com três componentes nos dados de três dias sem balancear as classes, como pode ser observado na Figura 10. A variância explicada indica o quanto da informação é mantida em cada componente após usar a redução de dimensionalidade (GÉRON, 2019). Portanto, quase toda a informação é preservada do conjunto de dados quando se reduz de 20 componentes para somente três.

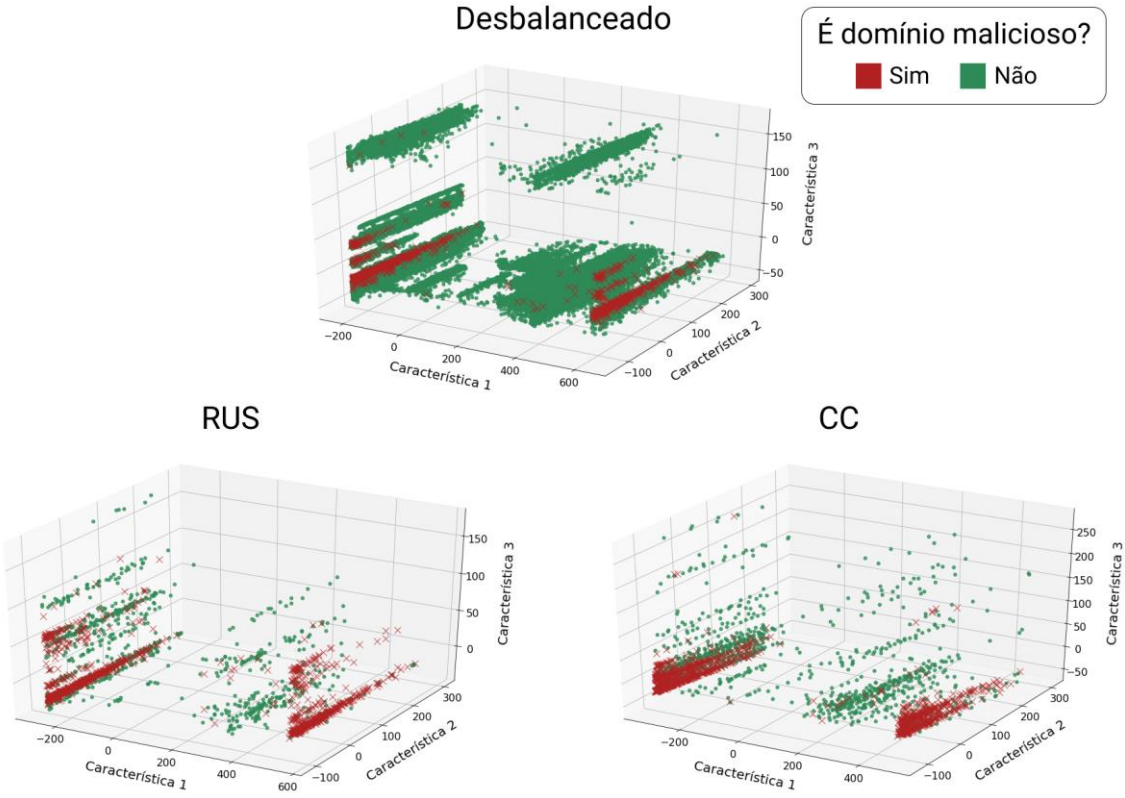
Figura 10: Variância explicada no PCA



Fonte: Elaborado pelo autor

Então, na Figura 11, é ilustrado o gráfico do PCA em três dimensões com (a) dados desbalanceados de três dias, aplicando os métodos de subamostragem (b) RUS e (c) CC. Percebe-se como há uma discrepância entre as classes e, a partir da aplicação dos métodos de subamostragem, há uma maior igualdade entre ambas as classes.

Figura 11: Comparação entre os métodos de subamostragem



Fonte: Elaborado pelo autor

Na Figura 11, nota-se que o método CC preenche mais o espaço em comparação com a RUS, mesmo com o uso da subamostragem, devido a sua função de subamostrar por meio da geração de agrupamentos. Por isso, espera-se que o método CC disponha um desempenho superior ao do RUS.

Combinando os métodos de subamostragem e sobreamostragem, alcançou-se os resultados mostrados na Tabela 2. Os métodos são avaliados em relação à AUC em cada um dos modelos, além da aplicação do LightGBM com os hiperparâmetros padrões. Na comparação apenas dos métodos de subamostragem, o CC obteve uma vantagem devido ao seu funcionamento, obtendo amostras de domínios legítimos em diferentes regiões do espaço. Dessa forma, o CC mostrou um melhor desempenho do que RUS em quase todos os cenários, o qual obteve um acréscimo máximo na AUC de 0,0856, se comparado com RUS no M3D, por exemplo.

Tabela 2: Avaliação dos métodos de subamostragem e sobreamostragem

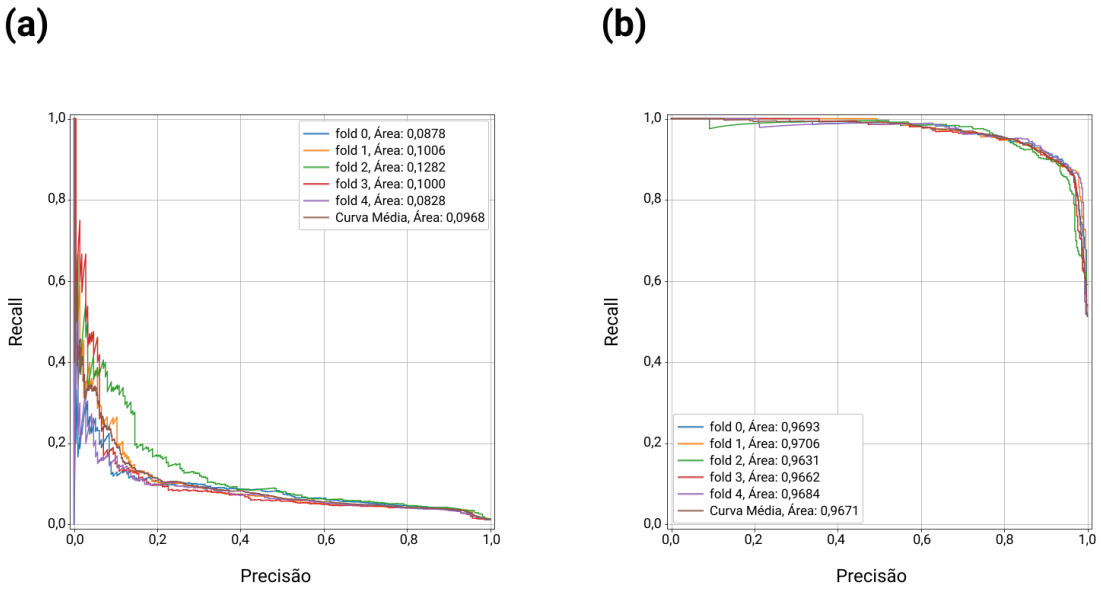
Subamostragem	Sobreamostragem	AUC				
		M3D	**MS1**	**MS2**	**MS3**	**MS4**
RUS	Sem sobreamostragem	0,8431	0,8740	0,8458	0,7788	0,7957
	ROS	0,9021	0,9142	0,8982	0,8994	0,8933
	SMOTE	0,9178	0,9289	0,8980	0,8951	0,8675
	Borderline-SMOTE	0,9134	0,9223	0,8518	0,8466	0,8614
	SVM-SMOTE	0,9178	0,9265	0,8644	0,8663	0,8877
	K-*Means* SMOTE	0,9315	0,9360	0,9075	0,9130	0,9291
CC	Sem sobreamostragem	0,9287	0,9301	0,8427	0,8503	0,8711
	ROS	0,9616	0,9630	**0,9312**	0,8881	**0,9604**
	SMOTE	0,9650	**0,9662**	0,9118	0,8956	0,9173
	Borderline-SMOTE	0,9655	0,9625	0,8949	0,8992	0,9323
	SVM-SMOTE	0,9648	0,9628	0,8782	0,9066	0,9265
	K-*Means* SMOTE	**0,9669**	0,9661	0,9161	**0,9212**	0,9397

Fonte: Elaborado pelo autor

Quando a combinação dos métodos de subamostragem e sobreamostragem é feita, há um acréscimo notável na AUC. Os métodos de sobreamostragem que conseguiram os maiores destaques nos resultados são o K-*Means* SMOTE e ROS. Contudo, o K-*Means* SMOTE é o método escolhido por ser mais recente e possuir um funcionamento diferente dos outros métodos. É importante ressaltar que os outros métodos também conseguiram excelentes AUC, com variações mínimas se comparadas com a maior AUC em cada teste. Logo, nos conjuntos de dados para treinamento, combina-se os métodos CC e K-*Means* SMOTE para tratar o desbalanceamento de classes.

Como forma de validar o balanceamento de classes, na Figura 12, é apresentada as curvas de PR com as classes (a) desbalanceadas e (b) balanceadas nos dados de três dias. Observa-se a sensibilidade da curva com as classes desbalanceadas, diferentemente da curva ROC, que sofre baixa variação. Desse modo, a utilização da curva ROC e AUC apenas é possível neste trabalho por conta de haver uma igualdade entre as classes depois da utilização dos métodos de reamostragem. Por fim, é possível ver que, com as classes balanceadas, há uma alta PR AUC, afirmando a credibilidade ao balancear as classes.

Figura 12: Curva de PR com as classes (a) desbalanceadas e (b) balanceadas



Fonte: Elaborado pelo autor

4.3. Pontos Iniciais e Iterações na Otimização Bayesiana

Ao usar a otimização Bayesiana para a seleção dos hiperparâmetros no XGBoost e LightGBM, com intuito de melhorar ainda mais o desempenho, é fundamental a escolha do número de pontos iniciais e iterações. Para isso, como se observa na Tabela 3, foram executados seis testes alterando o número de pontos iniciais e iterações, em que cada teste é associado a um número. O intuito dos testes é comparar a AUC e o tempo de execução ao alterar o número de pontos iniciais e iterações. O algoritmo LightGBM, com os dados de três dias, foi empregado para a realização dos testes. Ainda, destaca-se que o número total de iterações trata-se da soma do número de pontos iniciais e iterações.

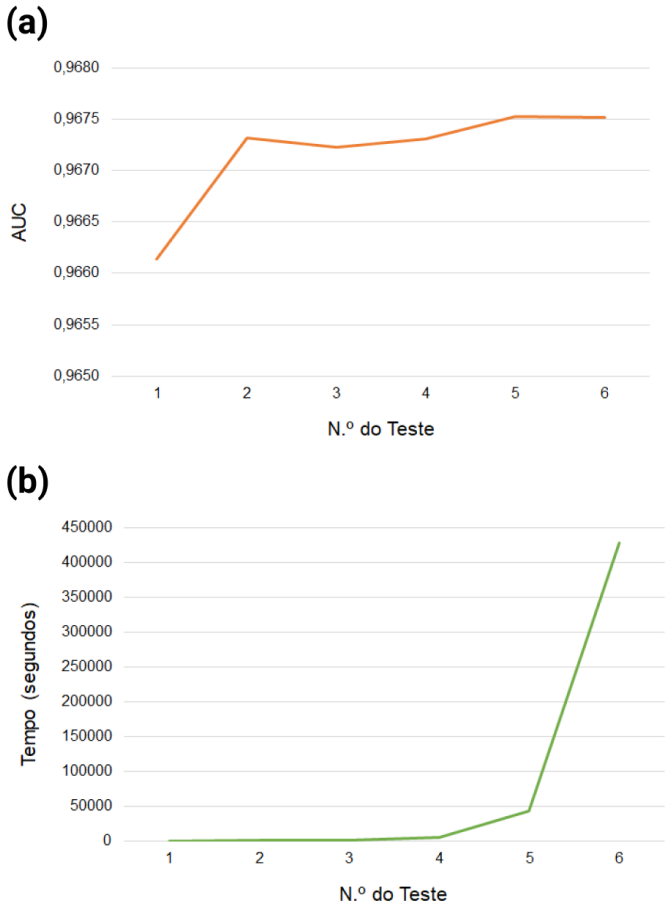
Tabela 3: Testes alterando o número de pontos iniciais e iterações

N.º do Teste	N.º de Pontos Iniciais	N.º de Iterações
1	10	100
2	25	250
3	50	500
4	100	1000
5	250	2500
6	500	5000

Fonte: Elaborado pelo autor

A partir do descrito antes, na Figura 13, gerou-se os gráficos comparando (a) AUC e (b) tempo de execução nos testes. Sobre a AUC, quanto maior é a quantidade de pontos iniciais e iterações, melhor é a AUC. Entretanto, há uma convergência da AUC do teste 2 em diante, dado que o acréscimo da métrica começa a ser mínimo. Por sua vez, o tempo gasto na execução cresce à medida que aumenta o número de pontos iniciais e iterações, o qual observa um crescimento exponencial do tempo.

Figura 13: (a) AUC e (b) tempo de execução nos testes



Fonte: Elaborado pelo autor

Logo, o baixo acréscimo na AUC e o longo tempo de execução, o qual levou cerca de cinco dias para ser executado, torna o teste 6 inviável, isto é, o ganho da AUC é mínimo quando comparado com o tempo. Desse modo, o teste 3 é escolhido por dispor do tempo de execução de aproximadamente 30 minutos, além da AUC ser suficiente. Portanto, a otimização Bayesiana é aplicada com 50 pontos iniciais e 500 iterações, totalizando um número total de iterações de 550. Com fundamento nisso, o equilíbrio da AUC e do tempo de execução é primordial, tendo em conta as configurações da máquina usada e o que é mais factível para cada situação em questão.

4.4. Comparação dos Algoritmos

Com a seleção dos métodos de subamostragem e sobreamostragem nos conjuntos de dados, iniciou-se o comparativo com diferentes algoritmos baseados em árvores, sendo eles o DT, o RF, o GBM, o XGBoost e o LightGBM. Os algoritmos foram comparados nos dados de três dias, além da primeira, segunda, terceira e quarta semana, resultando nos respectivos modelos para a classificação de domínios.

As métricas utilizadas para a avaliação são F1-score e principalmente AUC, como citado antes na Seção 3.7, e o tempo de treino em segundos. A AUC nos comparativos indica a AUC média, por causa da geração de cinco curvas ROC e suas respectivas áreas por conta do valor de $K = 5$. Por fim, (P) quer dizer que os hiperparâmetros padrões das bibliotecas scikit-learn, XGBoost e LightGBM foram aplicados; e (OB) significa que a otimização Bayesiana foi empregada para selecionar os hiperparâmetros.

4.4.1. Modelos de Três Dias e da Semana 1

Na Tabela 4, são exibidos os resultados obtidos no treinamento e teste do M3D. Com base nos resultados em relação à AUC, os algoritmos baseados no método *boosting* obtiveram os valores mais elevados, com a AUC maior ou igual a 0,9600. O algoritmo LightGBM (OB) alcançou a maior AUC média de 0,9673, melhorando 0,04% depois de usar a otimização Bayesiana para a seleção dos hiperparâmetros. Apesar da melhora ser baixa por lidar com um conjunto de dados bem reduzido, o desempenho tende a aumentar à proporção que há o aumento do conjunto de dados.

Sobre o tempo de treinamento, o DT obteve o tempo mais baixo, mas atingiu os piores valores nas outras métricas. Quando se compara somente os algoritmos baseados em *boosting*, o GBM é o que demorou mais tempo e o XGBoost (P) é o mais veloz. Tendo em mente a seleção de hiperparâmetros por meio da otimização Bayesiana, o LightGBM apresentou o menor tempo de treinamento. Ressalta-se que, por conta de dispor de poucos dados de treino e das configurações do hardware, o XGBoost e LightGBM não mostraram tanto diferencial no tempo, porém a computação paralela é extremamente útil para escalar em cenários com enormes conjuntos de dados.

Portanto, o LightGBM (OB) obteve os melhores resultados no geral, com a maior AUC média e uma boa F1-score, além do tempo menor do que o XGBoost (OB). Vale notar que a F1-score do algoritmo LightGBM (OB) é menor que a do LightGBM (P), pois a otimização Bayesiana é feita em termos da AUC, como explicado antes na Seção 3.7.

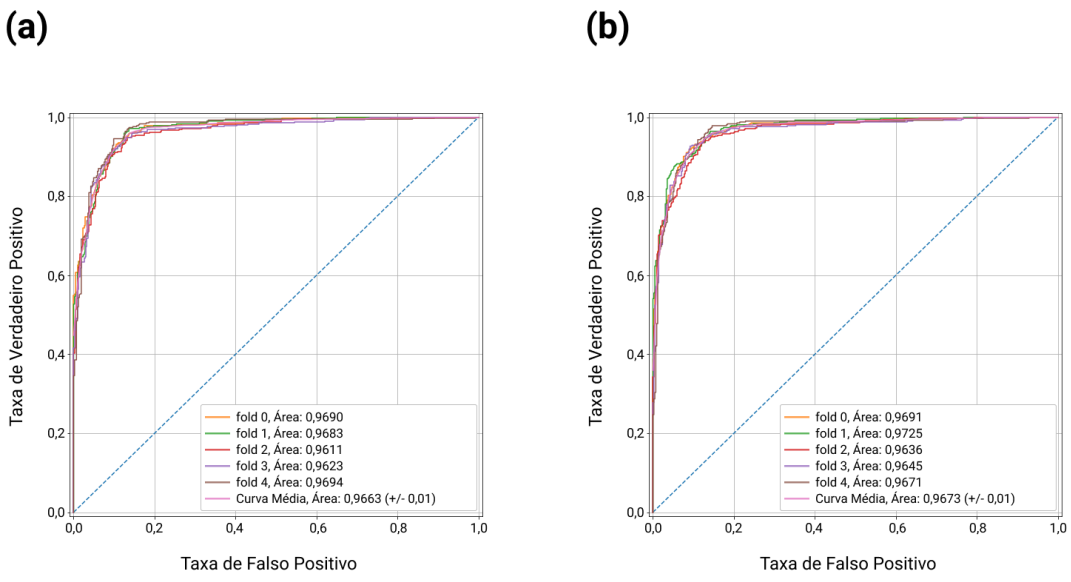
Tabela 4: Desempenho dos algoritmos – M3D

Algoritmo	F1-Score	AUC	Tempo de Treino (em segundos)
DT (P)	0,880	0,8765	**0,129712**
RF (P)	0,900	0,9582	2,091682
GBM (P)	0,905	0,9600	1,916872
XGBoost (P)	0,905	0,9642	0,904666
LightGBM (P)	**0,910**	0,9669	1,583613
XGBoost (OB)	**0,910**	0,9663	4,865108
LightGBM (OB)	0,905	**0,9673**	1,898552

Fonte: Elaborado pelo autor

Com o objetivo de comparar os dois melhores resultados mostrados na Tabela 4, na Figura 14, é apresentada a curva ROC do (a) XGBoost (OB) e (b) LightGBM (OB), no M3D. Na legenda de cada curva ROC, há a AUC em cada um dos cinco *folds*, e cada *fold* equivale a uma curva distinta. Após o treinamento e teste nos *folds*, uma sexta curva ROC é gerada junto com sua respectiva AUC, a qual denota a média das curvas anteriores. Além disso, a variação presente entre as curvas é indicada entre os parênteses após a área média, notando uma baixa variação de 0,01 em ambas as situações. Por conta disso, os valores da AUC nos *folds* são bem próximos.

Figura 14: Curva ROC do (a) XGBoost (OB) e (b) LightGBM (OB) – M3D



Fonte: Elaborado pelo autor

Das curvas ROC anteriores, os algoritmos apresentaram AUC próximas, com uma pequena vantagem de 0,0010 ao LightGBM (OB). Os resultados podem melhorar mais com o aumento do número de pontos iniciais e iterações na otimização Bayesiana, mas, como mencionado antes na Seção 4.3, é necessário analisar as configurações da máquina, além do quanto é viável o aumento no tempo de execução.

Na Tabela 5, são expostos os resultados alcançados no treino e teste do MS1. Dos resultados, observa-se um padrão semelhante ao M3D, em que a diferença está em um valor maior na F1-score e AUC no algoritmo LightGBM (OB), além da baixa variação no tempo de treinamento. Portanto, o LightGBM (OB) obteve uma AUC média de 0,9674 e tempo de 2,152275 segundos.

Novamente, verifica-se o excelente desempenho do algoritmo LightGBM e assim a sua aplicabilidade para a construção dos modelos. Por fim, como há poucos domínios com mais de três dias consecutivos de *queries*, é esperado que os M3D e MS1 apresentem resultados semelhantes. Por esse motivo, esses modelos possuem uma maior relevância no sistema, constatando nos resultados dos MS2, MS3 e MS4 posteriormente, visto que estes outros modelos apresentaram uma queda no desempenho.

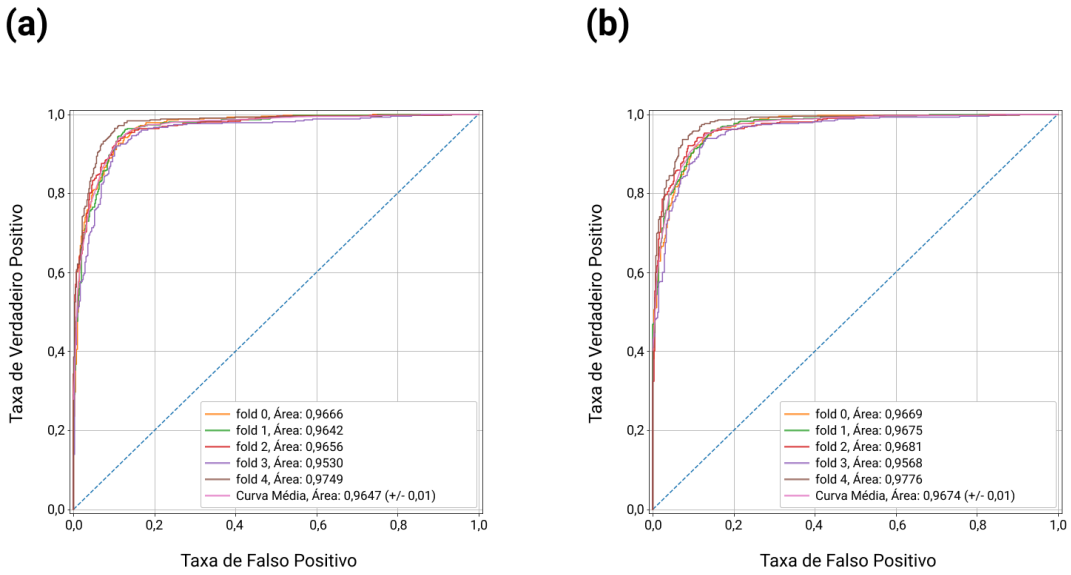
Tabela 5: Desempenho dos algoritmos – MS1

Algoritmo	F1-Score	AUC	Tempo de Treino (em segundos)
DT (P)	0,875	0,8748	**0,137400**
RF (P)	0,905	0,9548	1,896597
GBM (P)	0,905	0,9569	2,080640
XGBoost (P)	0,905	0,9633	0,921984
LightGBM (P)	**0,910**	0,9661	1,700336
XGBoost (OB)	**0,910**	0,9647	4,073478
LightGBM (OB)	**0,910**	**0,9674**	2,152275

Fonte: Elaborado pelo autor

Na Figura 15, é ilustrada a curva ROC do (a) XGBoost (OB) e (b) LightGBM (OB) no MS1. Novamente, observa-se que o algoritmo LightGBM (OB) obteve vantagem em relação ao XGBoost (OB), sendo maior que no M3D, em que a AUC aumentou 0,0027. A variação entre as curvas se manteve em 0,01 também, notando a proximidade entre as curvas ROC, como nos resultados anteriores do M3D.

Figura 15: Curva ROC do (a) XGBoost (OB) e (b) LightGBM (OB) – MS1



Fonte: Elaborado pelo autor

4.4.2. Modelos da Semana 2, Semana 3 e Semana 4

Nas Tabelas 6, 7 e 8, são mostrados os resultados do comparativo do treinamento e teste nos algoritmos nos MS2, MS3 e MS4, respectivamente. Nota-se que aconteceu uma queda no desempenho dos modelos quando comparados com os M3D e MS1, e isso deve-se ao fato de dispor de menos dados de treinamento. O LightGBM (OB) obteve as melhores AUC nos MS3 e MS4, exceto do MS2, no qual o XGBoost (P) foi superior. No entanto, como o LightGBM (OB) contou com as métricas mais altas em quatro dos cinco modelos, ele é escolhido para o treinamento de todos os modelos.

Tabela 6: Desempenho dos algoritmos – MS2

Algoritmo	F1-Score	AUC	Tempo de Treino (em segundos)
DT (P)	0,840	0,8418	**0,067413**
RF (P)	0,860	0,9296	1,096429
GBM (P)	**0,865**	0,9285	0,419237
XGBoost (P)	0,845	**0,9301**	0,328955
LightGBM (P)	0,840	0,9161	0,257108
XGBoost (OB)	0,820	0,9031	0,925798
LightGBM (OB)	0,855	0,9297	0,165021

Fonte: Elaborado pelo autor

Tabela 7: Desempenho dos algoritmos – MS3

Algoritmo	F1-Score	AUC	Tempo de Treino (em segundos)
DT (P)	0,755	0,7519	**0,064504**
RF (P)	**0,870**	0,9314	0,957978
GBM (P)	0,850	0,9073	0,349468
XGBoost (P)	0,785	0,8818	0,302123
LightGBM (P)	0,810	0,9212	0,403082
XGBoost (OB)	0,810	0,9217	1,180987
LightGBM (OB)	0,845	**0,9434**	0,197354

Fonte: Elaborado pelo autor

Tabela 8: Desempenho dos algoritmos – MS4

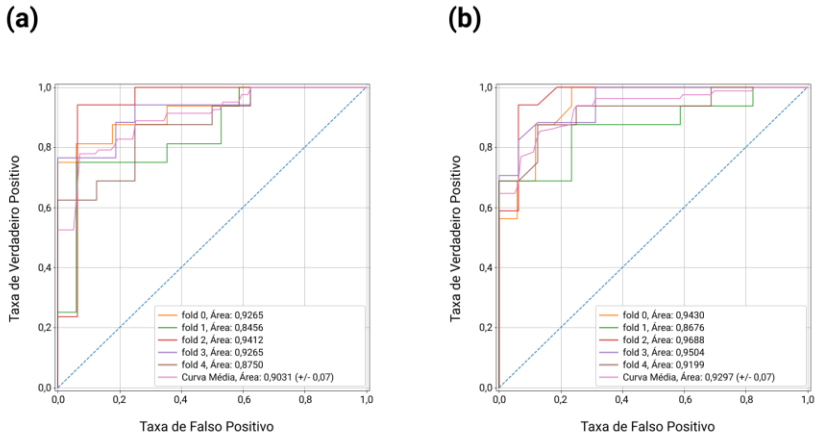
Algoritmo	F1-Score	AUC	Tempo de Treino (em segundos)
DT (P)	0,805	0,8026	**0,066146**
RF (P)	**0,860**	0,9443	1,083737
GBM (P)	0,850	0,9446	0,426847
XGBoost (P)	0,855	0,9345	0,362028
LightGBM (P)	0,850	0,9397	0,339182
XGBoost (OB)	0,850	0,9387	0,348786
LightGBM (OB)	0,855	**0,9504**	0,481022

Fonte: Elaborado pelo autor

Nas Figuras 16, 17 e 18, são apresentadas as curvas ROC do (a) XGBoost (OB) e (b) LightGBM (OB) nos MS2, MS3 e MS4, respectivamente. Nessas situações, as curvas ROC têm aparências distintas das exibidas anteriormente, além de uma variação máxima um pouco maior de 0,11. Esses fatores devem-se à presença de poucos dados para o treino desses modelos, como visto na Seção 3.5, mesmo após duplicar os domínios maliciosos.

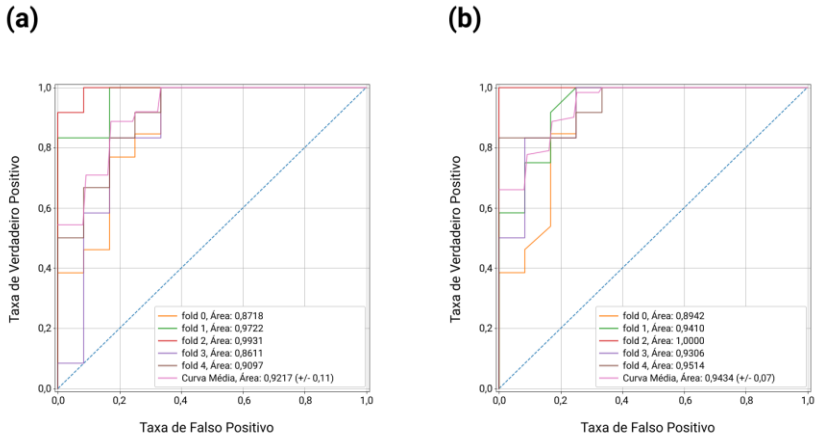
Consequentemente, não é possível garantir uma alta confiabilidade nos MS2, MS3 e MS4, necessitando-se o treino com mais dados para equiparar em desempenho com os M3D e MS1. Logo, em uma implementação dos modelos em um ambiente de produção, a atenção maior é nos modelos treinados com mais dados, por desfrutarem de um maior conhecimento agregado.

Figura 16: Curva ROC do (a) XGBoost (OB) e (b) LightGBM (OB) – MS2



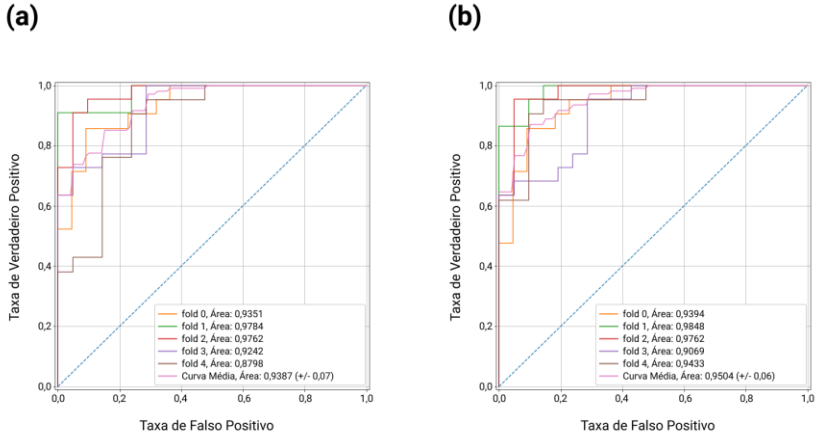
Fonte: Elaborado pelo autor

Figura 17: Curva ROC do (a) XGBoost (OB) e (b) LightGBM (OB) – MS3



Fonte: Elaborado pelo autor

Figura 18: Curva ROC do (a) XGBoost (OB) e (b) LightGBM (OB) – MS4

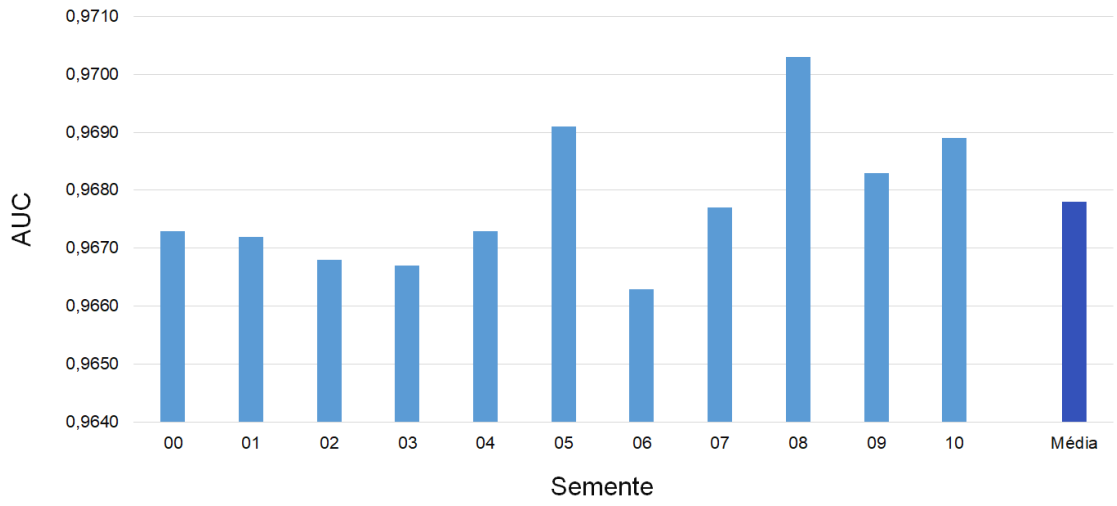


Fonte: Elaborado pelo autor

4.5. Alteração da Semente no CC

Como o método CC foi selecionado para subamostrar os conjuntos de dados, em concordância com os resultados apresentados na Seção 4.2, é necessário avaliar a variação da AUC ao alterar a semente do método. É importante ressaltar que a semente é vital para controlar a randomização do algoritmo. Assim, definiu-se 11 sementes distintas, das quais a primeira semente equivale a utilizada para subamostrar os dados de três dias e, com isso, gerou-se um conjunto de dados diferente para cada semente. A partir disso, avaliou-se os resultados com relação a AUC no LightGBM com a otimização Bayesiana. Dessa forma, na Figura 19, é ilustrada a variação da AUC ao alterar a semente no CC.

Figura 19: Variação da AUC ao alterar a semente no CC



Fonte: Elaborado pelo autor

Com base no gráfico anterior, a média de todos os valores da AUC foi 0,9678, que é um valor próximo do resultado do modelo de três dias. Além disso, a AUC maior foi na semente 08 e menor na semente 03, sendo a variação máxima da AUC igual a 0,0004. Portanto, é possível o uso do método CC para subamostragem dos dados, uma vez que a variação da AUC ao modificar a semente é mínima, além da baixa variação apontar que não houve *overfitting* e perda significante dos dados.

4.6. Teste dos Modelos em Novos Dados

Depois de construir todos os modelos para classificar domínios recém-registrados, é necessário o teste nos modelos com novos dados, os quais são referentes aos domínios registrados entre 01 de janeiro de 2021 e 30 de junho de 2021. Como mencionado antes

na Seção 3.3, o tempo de coleta é mais dois meses extras, então o tráfego de DNS passivo é até final de agosto de 2021. Com a coleta do tráfego e a extração de características realizadas, os dados foram classificados nos M3D e MS1. Em razão de possuir poucos dados coletados da segunda semana em diante, uma vez que o TLD remove os domínios maliciosos de forma rápida, os MS2, MS3 e MS4 não foram possíveis de serem validados com novos dados. Nas Tabelas 9 e 10, são apresentadas as matrizes de confusão com dados de teste nos M3D e MS1, respectivamente.

Tabela 9: Matriz de confusão com dados de teste – M3D

		Valor Predito	
		Malicioso	**Legítimo**
Valor Real	**Malicioso**	335	52
	Legítimo	9.917	18.657

Fonte: Elaborado pelo autor

Tabela 10: Matriz de confusão com dados de teste – MS1

		Valor Predito	
		Malicioso	**Legítimo**
Valor Real	**Malicioso**	336	51
	Legítimo	9.189	19.385

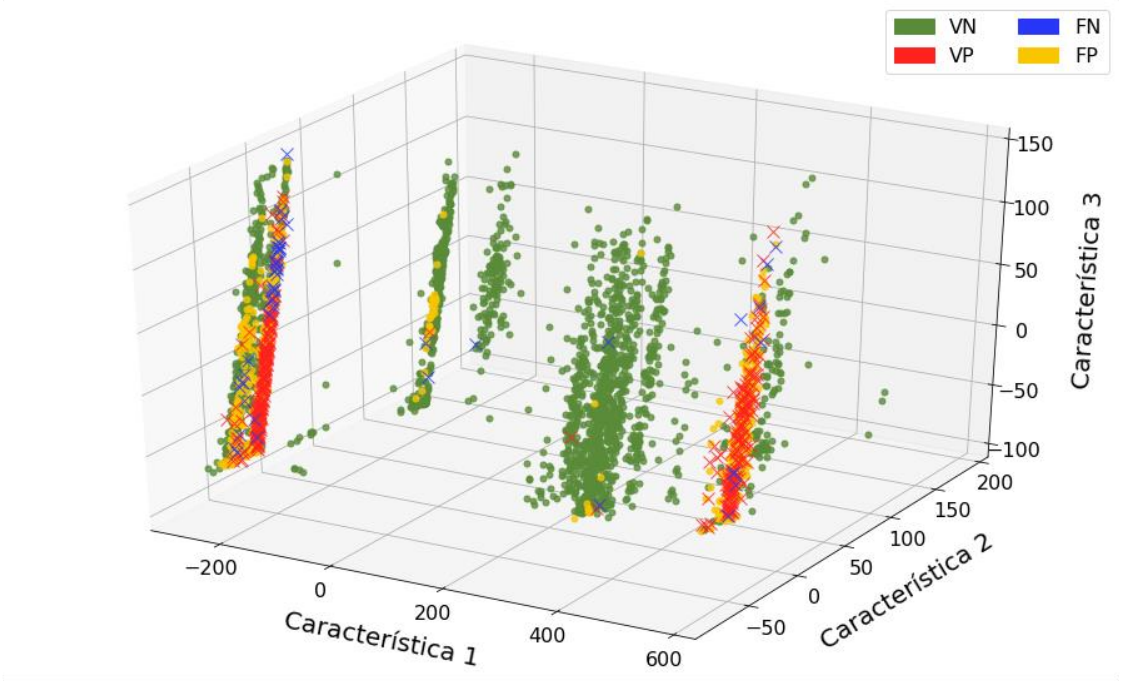
Fonte: Elaborado pelo autor

Com a meta de simplificar as matrizes de confusão, a TVP e a TFP foram calculadas em ambos os modelos. Recordando, a TVP é a taxa de domínios maliciosos que foram classificados corretamente e, por sua vez, a TFP consiste na taxa de domínios maliciosos classificados incorretamente. Logo, os M3D e MS1 tiveram, respectivamente, uma TVP de 0,8656 e 0,8682, e uma TFP de 0,3471 e 0,3216. A partir desses resultados, é possível comprovar um resultado considerável em especial na detecção dos domínios maliciosos. No entanto, o número de FPs é um pouco elevado, o que designa uma possível falha dos modelos em relação aos domínios legítimos.

Para melhor entender a situação dos dados classificados, na Figura 20, é exibido o gráfico PCA tridimensional dos dados de teste do M3D, em que os dados estão rotulados como VN, VP, FN e FP. É importante ressaltar que em razão do *overplotting* gráfico, isto

é, a presença de muitos pontos se sobrepondo, foi necessário usar o método RUS antes de construir o gráfico para auxiliar na visualização.

Figura 20: PCA em três dimensões nos dados de validação



Fonte: Elaborado pelo autor

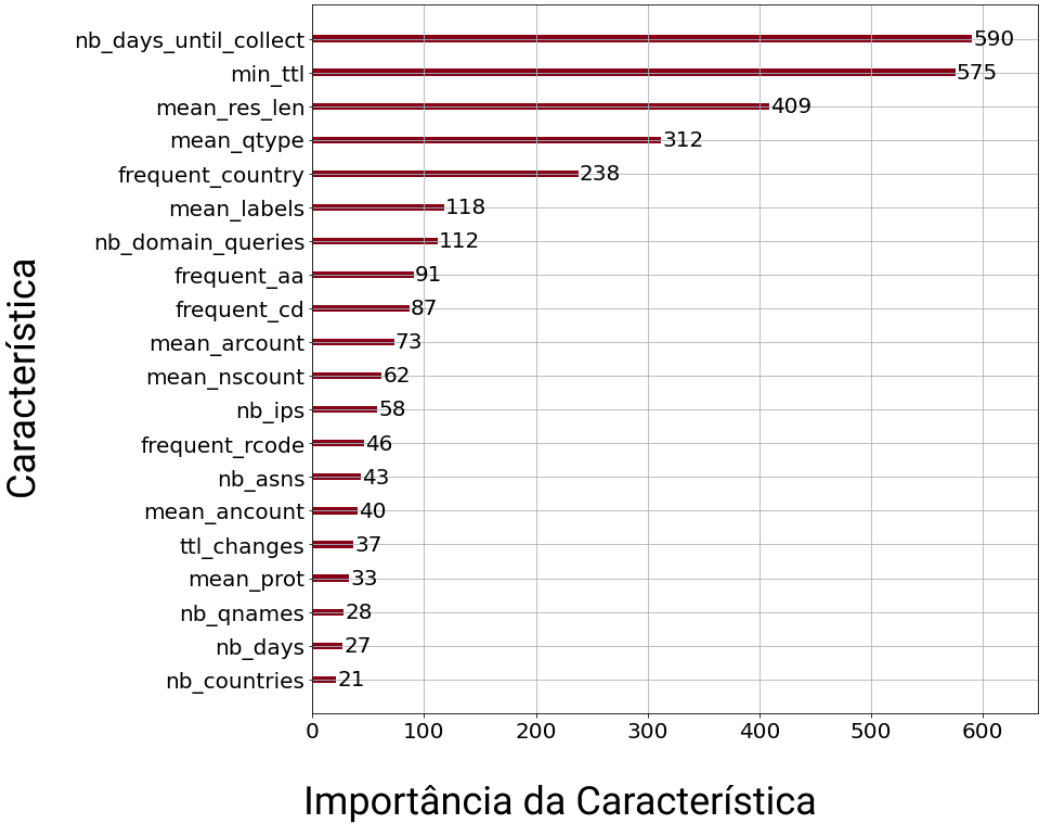
A partir do gráfico, repara-se que existem domínios maliciosos bem similares aos legítimos, além dos domínios classificados erroneamente estarem na fronteira de decisão do algoritmo. Portanto, as classificações erradas aconteceram por causa da aplicação da sobreamostragem nos conjuntos de dados, dado que os domínios maliciosos presentes na fronteira de decisão são replicados. A solução do problema é a obtenção de mais amostras reais da classe minoritária, o que garante mais confiabilidade aos modelos.

4.7. Importância das Características

Na Figura 21, é ilustrada a importância de cada característica após construir o M3D. As três características mais pertinentes ao modelo são: a “nb_days_until_collect”, dado que os domínios maliciosos propendem a ser registrados e, logo na sequência, são usados, por esse motivo a importância dessa característica; a “min_ttl”; em que domínios maliciosos têm preferências a valores de TTL baixos, em especial para *fast fluxing*, proporcionando um IP diferente para quase toda solicitação DNS (BILGE *et al.*, 2011), que interfere direto na característica “ttl_changes” também; e, por fim, a “mean_res_len”, tendo uma variação no tamanho da mensagem de resposta DNS entre as classes.

Outras características extraídas têm relevância, também, na detecção de domínios maliciosos, como a “frequent_country”, a “nb_ips”, a “nb_asns” e a “nb_countries”, que são características relativas à localização, uma vez que esses domínios resolvem para máquinas contaminadas em diferentes locais do mundo. Por isso, os domínios maliciosos definem vários ASNs para haver uma variação nos prefixos IP (ANTONAKAKIS *et al.*, 2011). Por fim, a “nb_domain_queries” é utilizada devido aos domínios maliciosos, em especial para campanhas *phishing*, possuírem um número alto de *queries* em um intervalo curto de tempo (LISON; MAVROEIDIS, 2017).

Figura 21: Importância de cada característica – M3D



Fonte: Elaborado pelo autor

4.8. Considerações Finais

Neste capítulo, foi discutido e apresentado os resultados obtidos ao desenvolver os modelos. Dentre os resultados, incluem-se a avaliação dos métodos de subamostragem e sobreamostragem, a definição do número de pontos iniciais e iterações na otimização Bayesiana, a comparação dos algoritmos baseados em árvores, a variação da AUC quando modifica a semente no CC, o teste dos modelos em novos dados e, por fim, a importância das características.

Baseado nos resultados alcançados, em especial nos M3D e MS1 na comparação dos algoritmos, notou-se que eles apresentaram a maior AUC média de 0,9673 e 0,9674, respectivamente. Comparando esses valores com a AUC de outros trabalhos que abordam a mesma temática, no trabalho de Bilge *et al.* (2011), foi alcançado uma AUC de 0,987 com o DT e validação cruzada com $K = 5$. Em Lison; Mavroeidis (2017), atingiu-se a AUC mais baixa de 0,976 e mais alta de 0,997, com 12 testes nas redes neurais. Por sua vez, no trabalho de Bao *et al.* (2019), utiliza-se o algoritmo XGBoost e obtém-se uma AUC de 0,974, com as características de acesso, e 0,994, com todas as características, incluindo as características textuais. Enfim, em Silveira *et al.* (2020), é obtida uma AUC de 0,976, com a aplicação do XGBoost também. Desse modo, em nenhum dos trabalhos, é empregado o algoritmo LightGBM, sendo, no máximo, usado o XGBoost (BAO *et al.*, 2019; SILVEIRA *et al.*, 2020) e o CatBoost (WANG *et al.*, 2020), os quais são algoritmos baseados no método *boosting*. Sobre as características, os trabalhos usam características textuais e numéricas para conseguirem altos resultados (BILGE *et al.*, 2011; LISON; MAVROEIDIS, 2017; BAO *et al.*, 2019), exceto alguns trabalhos (ANTONAKAKIS *et al.*, 2010, ANTONAKAKIS *et al.*, 2011; SILVEIRA *et al.*, 2020).

Em suma, este trabalho apresentou um resultado eficiente com a combinação dos métodos CC e K-*Means* SMOTE para balancear as classes, além de empregar o algoritmo LightGBM para treinamento dos modelos. Ainda, destaca-se o uso das características de DNS passivo em conjunto com os dados de geolocalização enriquecidos no ENTRADA. Do mesmo modo que no Kopis (ANTONAKAKIS *et al.*, 2011), o tráfego DNS é coletado do servidor autoritativo de um TLD. Levando em conta isso, o sistema desenvolvido neste trabalho atingiu AUC maiores que 0,96 nos dois modelos principais, que se assemelham com os resultados dos trabalhos discutidos. O sistema possui o diferencial na classificação de domínios recém-registrados em um TLD, propiciando o monitoramento dos domínios no primeiro mês de vida depois da primeira *query*. Assim, auxilia na identificação dos domínios maliciosos que contornaram a detecção feita pelo TLD. Por fim, o sistema é um complemento aos trabalhos existentes, uma vez que esses trabalhos classificam domínios registrados sem o início da coleta na primeira *query*, isto é, domínios existentes há algum tempo. No próximo capítulo, serão apresentadas as conclusões do trabalho, englobando a conclusão geral, as dificuldades encontradas, os trabalhos futuros e as produções obtidas.

Capítulo 5 – Conclusões

5.1. Conclusão Geral

Dada a importância do DNS ser essencial para o bom funcionamento da Internet, especialmente por traduzir nomes de *hosts* em endereços IP, atacantes registram domínios por motivos maliciosos. Segundo o *Internet Security Threat Report* (ISTR), um em cada dez URLs são considerados maliciosos (SYMANTEC, 2019). Dessa forma, é preciso um sistema para identificar os domínios maliciosos, principalmente logo após o registro, para assim ser capaz de mitigar esses domínios de imediato.

Portanto, neste trabalho, foi elaborado um sistema para a classificação de domínios recém-registrados em um TLD por meio de DNS passivo. Para desenvolver o sistema, os dados foram coletados do servidor autoritativo de um TLD em um intervalo de tempo de 12 meses e, a partir disso, realizou-se a extração de 20 características do DNS passivo em conjunto com as colunas enriquecidas no *data streaming* ENTRADA.

Por haver poucos dados de domínios maliciosos, os métodos de reamostragem CC e K-*Means* SMOTE foram combinados para o balanceamento das classes, em virtude de serem os métodos com as AUC superiores dentre os dois de subamostragem e cinco de sobreamostragem. Ainda, comparou-se o desempenho de cinco algoritmos baseados em árvores, dos quais os algoritmos com o método *boosting* alcançaram os resultados mais elevados. O LightGBM sobressaiu-se no treinamento dos modelos, aperfeiçoando mais com a otimização Bayesiana de 50 pontos iniciais e 500 iterações. O XGBoost apresentou

bons resultados também, em algumas situações sendo até equivalente ao LightGBM, com uma diferença na AUC de 0,001 no M3D, por exemplo. Contudo, o LightGBM ainda tem a vantagem em relação ao XGBoost na velocidade computacional, além de um consumo de memória menor (MENG *et al.*, 2016).

O sistema desenvolvido monitora os domínios recém-registrados no primeiro mês de vida após a primeira *query*, aplicando cinco modelos para cada intervalo de tempo, os quais obtiveram uma AUC de 0,9673, 0,9674, 0,9297, 0,9434 e 0,9504 aos M3D, MS1, MS2, MS3 e MS4, respectivamente. Além disso, os tempos de treino nos modelos foram baixos, levando no máximo cerca de 2 segundos no LightGBM, e sendo possível escalar em cenários com mais dados de treinamento. Em adição, os M3D e MS1 foram validados com dados de teste coletados em um intervalo de oito meses, principalmente por esses modelos disporem de uma maior confiabilidade devido ao treino com mais amostras de domínios maliciosos, resultando em uma TVP de 0,8656 e 0,8682, respectivamente, as quais são taxas consideradas interessantes para a detecção de domínios maliciosos. Por fim, é fundamental a identificação rápida dos domínios maliciosos recém-registrados, em especial o TLD dispor recursos para tal detecção, mitigando esses domínios, o que impede que mais empresas tenham prejuízos ou usuários sofram golpes.

5.2. Dificuldades Encontradas

Por conta deste trabalho ser desenvolvido com dados provindos de um TLD, que tem um grande cuidado em detectar e remover os domínios maliciosos de modo rápido, em pouco tempo após o registro, muitas vezes não havendo nenhuma *query*, o número de domínios maliciosos se tornou limitado. Assim, a limitação desses dados foi a maior dificuldade no trabalho, sendo preciso aplicar o balanceamento de classes e validar os modelos de modo correto para evitar o *overfitting*. Essa dificuldade foi maior ainda nos modelos da segunda semana em diante, o que afetou o desempenho e a validação em novos dados.

Uma outra dificuldade foi obter os dados no ENTRADA, uma vez que foi preciso configurar corretamente o ambiente com o Apache Spark, o qual funcionou apenas depois de alguns meses de tentativas. A extração de características foi outra dificuldade também, em especial por deparar com características consideradas novas quando comparadas com os trabalhos existentes, sendo preciso validar empiricamente o uso dessas características e o impacto para a classificação de domínios.

5.3. Trabalhos Futuros

Como foi observado durante o trabalho, há poucos dados de domínios recém-registrados maliciosos, o qual corresponde a cerca de 1%, uma vez que o *registry* remove os domínios maliciosos antes mesmo da primeira *query* e, por conta disso, há a baixa volumetria de dados. Logo, um dos trabalhos é o uso do aprendizado não supervisionado para detectar novos domínios maliciosos. Desse modo, o número de amostras da classe minoritária é ampliado, o que reduz a quantia de dados sintéticos e aprimora a confiança dos modelos. Com a presença de mais dados de domínios maliciosos, é possível melhorar o sistema com a criação da classificação multiclasse, apontando o tipo de maliciosidade que o domínio apresenta, caso seja classificado como malicioso (DA SILVA *et al.*, 2020).

Além disso, o desenvolvimento de modelos incrementais é significativo, visto que agrega conhecimento ao decorrer do tempo com o aprendizado contínuo, em que auxilia a combater as classificações incorretas e detectar novos domínios maliciosos. É relevante ressaltar a implementação dos modelos no ambiente de produção, sendo possível validar os modelos na prática, além de aplicar o sistema em outros TLDs, especialmente naqueles que dispõem de uma alta taxa de domínios maliciosos, dado que existem preferência dos atacantes em determinados TLDs (DA SILVA *et al.*, 2020). Neste trabalho não foi usado dados de outros TLDs por conta da dificuldade de contato e acesso, além das questões de parceria. Enfim, um acompanhamento menor dos domínios, gerando classificadores para monitorarem os domínios recém-registrados no primeiro e segundo dia, além da primeira *query*, dado que boa parte dos domínios nos conjuntos de dados têm apenas uma *query*.

5.4. Produções

A respeito das produções obtidas, foram publicados dois artigos extraídos deste trabalho. O primeiro artigo, com o título *Multiclass Classification of Malicious Domains Using Passive DNS with XGBoost: (Work in Progress)*, publicado na conferência *2020 IEEE 19th International Symposium on Network Computing and Applications* (NCA). O segundo artigo, denominado *Detection of Newly Registered Malicious Domains through Passive DNS*, publicado no *Workshop on Big Data for Cybersecurity* (BigCyber) da *2021 IEEE International Congress on Big Data* (BigData). Por fim, um artigo foi aceito para publicação no periódico *International Journal of Communication Networks and Information Security* (IJCNIS), sendo titulado *Early Identification of Abused Domains in TLD through Passive DNS Applying Machine Learning Techniques*.

Referências

ALRWAIS, S.; YUAN, K.; ALOWAISHEQ, E.; LI, Z.; WANG, X. Understanding the Dark Side of Domain Parking. In: **Proceedings of the 23rd USENIX Security Symposium**, p. 207-222, 2014.

ANTONAKAKIS, M.; PERDISCI, R.; DAGON, D.; LEE, W.; FEAMSTER, N. Building a Dynamic Reputation System for DNS. In: **Proceedings of the 19th USENIX Security Symposium**, p. 273-290, 2010.

ANTONAKAKIS, M.; PERDISCI, R.; LEE, W.; VASILOGLOU, N.; DAGON, D. Detecting Malware Domains at the Upper DNS Hierarchy. In: **Proceedings of the 20th USENIX Security Symposium**, p. 1-16, 2011.

BAO, Z.; WANG, W.; LAN, Y. Using Passive DNS to Detect Malicious Domain Name. In: **Proceedings of the 3rd International Conference on Vision, Image and Signal Processing**, p. 1-8, 2019.

BATISTA, G. E. A. P. A.; PRATI, R. C.; MONARD, M. C. A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data. In: **ACM SIGKDD Explorations Newsletter**, v. 6, n. 1, p. 20-29, 2004.

BILGE, L.; KIRDA, E.; KRUEGEL, C.; BALDUZZI, M. EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis. In: **Ndss**, 2011, p. 1-17.

BORKIN, D.; NÉMETHOVÁ, A.; MICHAĽČONOK, G.; MAIOROV, K. Impact of Data Normalization on Classification Model Accuracy. In: **Research Papers Faculty**

of Materials Science and Technology Slovak University of Technology, v. 27, n. 45, p. 79-84, 2019.

BREIMAN, J.; FRIEDMAN, J. H.; OLSHEN, R. A.; STONE, C. J. **Classification and Regression Trees**. CRC Press, 1984.

CHAWLA, N. V.; BOWYER, K. W.; HALL, L. O.; KEGELMEYER, W. P. SMOTE: Synthetic Minority Over-sampling Technique. In: **Journal of Artificial Intelligence Research**, v. 16, p. 321-357, 2002.

CHEN, T.; GUESTRIN, C. XGBoost: A Scalable Tree Boosting System. In: **Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**, p. 785-794, 2016.

CHEN, Z.; JIANG, F.; CHENG, Y.; GU, X.; LIU, W.; PENG, J. XGBoost Classifier for DDoS Attack Detection and Analysis in SDN-Based Cloud. In: **2018 IEEE International Conference on Big Data and Smart Computing (BigComp)**, p. 251-256. IEEE, 2018.

CUI, H.; HUANG, D.; FANG, Y.; LIU, L.; CHENG, H. Webshell Detection Based on Random Forest-Gradient Boosting Decision Tree Algorithm. In: **2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)**, p. 153-160. IEEE, 2018.

DA SILVA, L. M.; SILVEIRA, M. R.; CANSIAN, A. M.; KOBAYASHI, H. K. Multiclass Classification of Malicious Domains Using Passive DNS with XGBoost:(Work in Progress). In: **2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)**, p. 1-3. IEEE, 2020.

DAVIS, J.; GOADRICH, M. The Relationship between Precision-Recall and ROC Curves. In: **Proceedings of the 23rd International Conference on Machine Learning**, p. 233-240, 2006.

DOMAIN NAME STAT. **Domain Name Registration's Statistics**. Domain Name Stat, 2021. Disponível em: <https://domainnamestat.com/statistics/overview>. Acesso em: 26 de jan. de 2022.

DOUZAS, G.; BACAO, F.; LAST, F. Improving Imbalanced Learning through a Heuristic Oversampling Method based on K-Means and SMOTE. In: **Information Sciences**, v. 465, p. 1-20, 2018.

DURUMERIC, Z.; BAILEY, M.; HALDERMAN, J. A. An Internet-wide View of Internet-Wide Scanning. In: **Proceedings of the 23rd USENIX Security Symposium**, p. 65-78, 2014.

FALL, K. R.; STEVENS, W. R. **TCP/IP Illustrated, Volume 1: The Protocols**. Addison-Wesley, 2011.

FAWCETT, T. ROC Graphs: Notes and Practical Considerations for Researchers. In: **Machine Learning**, v. 31, n. 1, p. 1-38, 2004.

FERNÁNDEZ, A.; GARCÍA, S.; GALAR, M.; PRATI, R. C.; KRAWCZYK, B.; HERRERA, F. **Learning from Imbalanced Data Sets**. Berlin: Springer, 2018.

GÉRON, A. **Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems**. O'Reilly Media, 2019.

HAN, H.; WANG, W. Y.; MAO, B. H. Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In: **International Conference on Intelligent Computing**, p. 878-887. Springer, 2005.

KE, G.; MENG, Q.; FINLEY, T.; WANG, T.; CHEN, W.; MA, W.; YE, Q.; LIU, T. Y. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In: **Advances in Neural Information Processing Systems 30 (NIPS 2017)**, v. 30, p. 3146-3154, 2017.

KHALIL, I.; YU, T.; GUAN, B. Discovering Malicious Domains through Passive DNS Data Graph Analysis. In: **Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security**, p. 663-674, 2016.

KINGSFORD, C.; SALZBERG, S. L. What are Decision Trees?. In: **Nature Biotechnology**, v. 26, n. 9, p. 1011-1013, 2008.

KOHAVI, R. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In: **Appears in the International Joint Conference on Artificial Intelligence (IJCAI)**, p. 1137-1145, 1995.

KOTSIANTIS, S. B. Supervised Machine Learning: A Review of Classification Techniques. In: **Emerging Artificial Intelligence Applications in Computer Engineering**, v. 160, n. 1, p. 3-24, 2007.

KOUNTOURAS, A.; KINTIS, P.; LEVER, C.; CHEN, Y., NADJI, Y.; DAGON, D.; ANTONAKAKIS, M.; JOFFE, R. Enabling Network Security through Active DNS Datasets. In: **International Symposium on Research in Attacks, Intrusions, and Defenses**, p. 188-208. Springer, 2016.

KÜHRER, M.; HUPPERICH, T.; BUSHART, J.; ROSSOW, C.; HOLZ, T. Going Wild: Large-Scale Classification of Open DNS Resolvers. In: **Proceedings of the 2015 Internet Measurement Conference**, p. 355-368, 2015.

KULIKOVA, T.; SHCHERBAKOVA, T. **Spam and phishing in Q3 2021**. Securelist, 2021. Disponível em: <https://securelist.com/spam-and-phishing-in-q3-2021/104741/>. Acesso em: 26 de nov. de 2021.

KUROSE, J. F.; ROSS, K. W. **Redes de Computadores e a Internet: Uma Abordagem Top-Down**. Person, 2013.

LISON, P.; MAVROEIDIS, V. Neural Reputation Models Learned from Passive DNS Data. In: **2017 IEEE International Conference on Big Data (Big Data)**, p. 3662-3671. IEEE, 2017.

LIU, C.; ALBITZ, P. **DNS and Bind**. O'Reilly Media, 2006.

LIU, Z.; HARDY, J.; ZHANG, J. A Practical Study on Imbalanced Data Re-Sampling for Conversion Rate of Online Advertising. **10.3906/elk-1806151**, 2019.

MAXMIND. **Licença do Usuário Final da MaxMind**. MaxMind, 2021. Disponível em: <https://www.maxmind.com/en/end-user-license-agreement-pt>. Acesso em: 19 de jan. de 2022.

MICROSOFT CORPORATION. **LightGBM Documentation**. LightGBM – Read the Docs, 2021. Disponível em: <https://lightgbm.readthedocs.io/en/latest/>. Acesso em: 30 de jun. de 2021.

NGUYEN, H. M.; COOPER, E. W.; KAMEI, K. Borderline Over-Sampling for Imbalanced Data Classification. In: **International Journal of Knowledge Engineering and Soft Data Paradigms**, v. 3, n. 1, p. 4-21, 2011.

PERDISCI, R.; CORONA, I.; GIACINTO, G. Early Detection of Malicious Flux Networks via Large-Scale Passive DNS Traffic Analysis. In: **IEEE Transactions on Dependable and Secure Computing**, v. 9, n. 5, p. 714-726, 2012.

RASCHKA, S. Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning. **arXiv**, 2018. arXiv:1811.12808.

REGISTRO.BR. **Domínios .br Registrados Até o Momento**. Registro.br, 2021. Disponível em: <https://registro.br/dominio/estatisticas/>. Acesso em: 26 de jan. de 2022.

SAFAVIAN, S. R.; LANDGREBE, D. A Survey of Decision Tree Classifier Methodology. In: **IEEE Transactions on Systems, Man, and Cybernetics**, IEEE, v. 21, n. 3, p. 660– 674, 1991.

SILVEIRA, M. R.; DA SILVA, L. M.; CANSIAN, A. M.; KOBAYASHI, H. K. XGBoost Applied to Identify Malicious Domains Using Passive DNS. In: **2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)**, p. 1-4. IEEE, 2020.

SPOOREN, J.; VISSERS, T.; JANSSEN, P.; JOOSEN, W.; DESMET, L. Premadoma: An Operational Solution for DNS Registries to Prevent Malicious Domain Registrations. In: **Proceedings of the 35th Annual Computer Security Applications Conference**, p. 557-567, 2019.

STEVENS, W. R. **TCP/IP Illustrated, Volume 1: The Protocols**. Addison-Wesley, 1994.

SYMANTEC. **Internet Security Threat Report – Volume 24 (February 2019)**. Broadcom, 2019. Disponível em: <https://docs.broadcom.com/doc/istr-24-2019-en>. Acesso em: 29 de set. de 2021.

TORABI, S.; BOUKHTOUTA, A.; ASSI, C.; DEBBABI, M. Detecting Internet Abuse by Analyzing Passive DNS Traffic: A Survey of Implemented Systems. In: **IEEE Communications Surveys & Tutorials**, n. 4, p. 3389-3415. IEEE, 2018.

WANG, Q.; LI, L.; JIANG, B.; LU, Z.; LIU, J.; JIAN, S. Malicious Domain Detection Based on K-Means and SMOTE. In: **International Conference on Computational Science, p. 468-481**. Springer, 2020.

WATKINS, L.; BECK, S.; Zook, J.; BUCZAK, A.; CHAVIS, J.; ROBINSON, W. H.; MORALES, J. A.; MISHRA, S. Using Semi-Supervised Machine Learning to Address the Big Data Problem in DNS Networks. In: **2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)**, p. 1-6. IEEE, 2017.

WEIMER, F. Passive DNS Replication. In: **FIRST Conference on Computer Security Incident**. p. 1-14, 2005.

WEISS, E. A. Biographies: Eloge: Arthur Lee Samuel (1901-90). In: **IEEE Annals of the History of Computing**, IEEE, v. 14, n. 3, p. 55-69, 1992.

WULLINK, M.; MOURA, G. C. M.; MÜLLER, M.; HESSELMAN, C. ENTRADA: A High-Performance Network Traffic Data Streaming Warehouse. In: **NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium**, p. 913-918. IEEE, 2016.

XIANG, G.; HONG, J. I.; ROSÉ, C. P.; CRANOR, L. F. CANTINA+: A Feature-Rich Machine Learning Framework for Detecting Phishing Web Sites. In: **ACM Transactions on Information and System Security**, v. 14, n. 2, p. 1-28, 2011.

YEN, S. J.; LEE, Y. S. Cluster-based Under-Sampling Approaches for Imbalanced Data Distributions. In: **Expert Systems with Applications**, v. 36, n. 3, p. 5718-5727, 2009.

ZHANG, S.; ZHOU, Z.; LI, D.; ZHONG, Y.; LIU, Q.; YANG, W.; LI, S. Attributed Heterogeneous Graph Neural Network for Malicious Domain Detection. In: **2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)**, p. 397-403. IEEE, 2021.