UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Avaliação de Modelos de Aprendizado de Máquina para Detecção Reativa e Preventiva de Botnets

Vinicius Oliveira de Souza

Orientador

Sidney Cunha de Lucena

RIO DE JANEIRO, RJ - BRASIL
AGOSTO de 2018

Avaliação de Modelos de Aprendizado de Máquina para Detecção Reativa e Preventiva de Botnets

Vinicius Oliveira de Souza

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO). APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA.

Aprovada por:

Sidney Cunha de Lucena, D.Sc. - UNIRIO

Kate Cerqueira Revoredo, D.Sc. - UNIRIO

Antônio Augusto de Aragão Rocha, D.Sc. - UFF

Ronaldo Moreira Salles, Ph.D. - IME

RIO DE JANEIRO, RJ - BRASIL
AGOSTO de 2018

Souza, Vinicius Oliveira de.
S719 Avaliação de Modelos de Aprendizado de Máquina para Detecção Reativa e Preventiva de Botnets / Vinicius Oliveira de Souza. – Rio de Janeiro, 2018.
110 f.

Orientador: Sidney Cunha de Lucena
Dissertação (Mestrado) - Universidade Federal do Estado do Rio de Janeiro, Programa de Pós-Graduação em Informática, 2018.

1. botnet. 2. aprendizado de máquina. 3. detecção de anomalia. 4. sistemas de detecção de intrusão. I. Lucena, Sidney Cunha de, orient. II. Título.

À Glória do Grande Arquiteto do Universo.

Agradecimentos

Ao meu orientador Sidney Cunha de Lucena pela motivação e total apoio no desenvolvimento deste trabalho, não poupando esforços e trabalhando muitas vezes fora de seu horário.

À minha esposa Gisele Pereira, pelo amor, paciência, compreensão e inspiração.

A todos os meus familiares que me apoiaram desde o início da caminhada, educaram-me e contribuíram para o homem que sou hoje.

A todo o corpo docente do Programa de Pós-Graduação em Informática da Universidade Federal do Estado do Rio de Janeiro (PPGI UNIRIO) pelos ensinamentos transmitidos, em especial à professora Kate Revoredo por ter ministrado com excelência a disciplina Descoberta do Conhecimento em Banco de Dados, fundamental para a definição desta pesquisa.

Aos amigos Daisy e Sérgio Silva do Instituto Militar de Engenharia (IME), pelas reuniões, esclarecimentos e todo suporte concedido.

A todos os amigos do mestrado, pela fraternidade, ensinamentos, experiências trocadas e apoio mútuo.

Ao CPRM - Serviço Geológico do Brasil, pelo apoio e compreensão nos períodos de afastamento.

Souza, Vinicius Oliveira de. **Avaliação de Modelos de Aprendizado de Máquina para Detecção Reativa e Preventiva de Botnets**. UNIRIO, 2018. 110 páginas. Dissertação de Mestrado. Programa de Pós-Graduação em Informática, UNIRIO.

RESUMO

Cada vez mais o mundo se torna dependente da internet e dos sistemas de informação. Com o aumento dessa dependência, também cresce o número de crimes cibernéticos, assim como a relevância de pesquisas na área de segurança da informação e o consequente combate a esses crimes. Para realização dessas atividades maliciosas, geralmente são utilizadas *botnets*, ou seja, redes de máquinas infectadas (os *bots*) controladas remotamente para atuarem como plataformas de lançamento de ataques à segurança de redes e serviços. Detectar essas redes maliciosas é uma tarefa complexa, devido às características dinâmicas deste tráfego. Diante disso, diversos estudos e sistemas baseados em técnicas de aprendizado de máquina foram propostos para a identificação de *malwares* a partir da inspeção do tráfego de rede. Relacionados a isso, podem ser encontrados na literatura diversos estudos propondo técnicas visando selecionar as características do tráfego de rede mais adequadas para serem usadas com diversos tipos diferentes de modelos de aprendizado. No entanto, comparar os resultados entre estes estudos é tarefa muito difícil, algumas vezes inviável, visto que a maioria deles se apoia em diferentes conjuntos de dados e diferentes métodos de avaliação. Por conta disto, há uma escassez de trabalhos que comparem, por intermédio de uma mesma metodologia e apoiado num mesmo conjunto de dados, diversos algoritmos englobando os diversos paradigmas de aprendizado. O objetivo deste trabalho é avaliar diferentes modelos, baseados nos diversos paradigmas do aprendizado de máquina, supervisionado e não supervisionado, para a detecção de diferentes arquiteturas de *botnets* em diferentes cenários de ataques, tanto na forma reativa quanto na preventiva. A metodologia utilizada se vale de selecionar características do tráfego de rede relevantes à detecção de *botnets* através de técnicas de seleção de atributos, comparando o resultado desta seleção com a de trabalhos anteriores. Algumas características foram ratificadas, no entanto, novas características surgem como contribuição. Após isso, essas características alimentaram diversos modelos de aprendizado de máquina, tanto supervisionado quanto não supervisionado, que foram avaliados com métricas recomendadas para cenários desbalanceados. A partir da análise criteriosa dos resultados, os modelos gerados pelo

algoritmo *Random Forest* se destacaram nos diferentes cenários de *botnets*, tanto na detecção reativa, quanto na preventiva.

Palavras-chave: *botnet*, aprendizado de máquina, detecção de anomalia, sistemas de detecção de intrusão.

ABSTRACT

More and more the world becomes dependent on the internet and information systems. As this dependency increases, the number of cybercrimes also increases, as well as the relevance of research in the area of information security and yours consequent fight. In order to perform these malicious activities, botnets are generally used, i.e. networks of infected machines (the bots) controlled remotely to act as launching platforms for attacks on the security of networks and services. Detect these malicious networks is a complex task due to the dynamic characteristics of this traffic. Therefore, several studies and systems based on machine learning techniques were proposed for the identification of malware from network traffic inspection. Related to this, we can find in the literature several studies proposing techniques to select the most suitable network traffic characteristics to be used with several different types of learning models. However, comparing the results between these studies is very difficult, sometimes impractical, since most of them rely on different datasets and different methods of evaluation. Because of this, there is a shortage of works that compare, through a single methodology and supported by the same set of data, several algorithms encompassing the different paradigms of learning. The objective of this work is to evaluate different models, based on the different paradigms of machine learning, supervised and unsupervised, for the detection of different botnet architectures in different attack scenarios, both reactive and preventive. The methodology used is to select network traffic characteristics relevant to the detection of botnets through attribute selection techniques, comparing the results of this selection with those of previous works. Some characteristics have been ratified, however, new features emerge as a contribution. After that, these characteristics fed several models of machine learning, both supervised and unsupervised, that were evaluated with recommended metrics for unbalanced scenarios. From the careful analysis of the results, the models generated by the Random Forest algorithm stand out in the different scenarios of botnets, both reactive and preventive detection.

Keywords: botnet, machine learning, anomaly detection, intrusion detection systems.

Sumário

1 Introdução **1**

1.1 Considerações Iniciais e Escopo do Trabalho 1

1.2 Descrição do Problema e Objetivos 3

1.3 Contribuições do Trabalho 6

1.4 Estrutura da Dissertação 6

2 Fundamentos de Botnets **8**

2.1 *Botnets* 8

2.1.1 Arquitetura 9

2.1.2 Ciclo de Vida 10

2.1.3 Métodos de evasão 11

2.1.4 Atividades Maliciosas 12

2.2 Detecção de *Botnets* 13

2.2.1 Detecção baseada em *Honeynet* 14

2.2.2 Detecção baseada em IDS 14

2.2.2.1 *IDS* Baseados em *Host (HIDS)* 15

2.2.2.2 *IDS* Baseados em *Rede (NIDS)* 15

2.2.2.3 Detecção Baseada em Assinatura 16

2.2.2.4 Detecção Baseada em Anomalia 16
2.2.3 Defesa contra *botnets* 17

3 Fundamentos de KDD e Aprendizado de Máquina 19

3.1 Descoberta de Conhecimento em Banco de Dados 19
3.1.1 Mineração de Dados 20
3.2 Aprendizado de Máquina 21
3.2.1 Hierarquia do Aprendizado 22
3.2.2 Paradigmas do Aprendizado 23
3.2.3 Avaliação de Algoritmos 24
3.2.3.1 Métodos de amostragem 25
3.2.3.2 Classes Desbalanceadas 27
3.2.4 Técnicas de Seleção de Atributos 29
3.2.4.1 Ranqueamento de Atributos 29
3.2.4.2 *Correlation-based Feature Selection* 30
3.2.4.3 *Wrapper* 30
3.2.5 Algoritmos de Aprendizado de Máquina 31
3.2.5.1 Árvore de Decisão 31
3.2.5.2 *Random Forest* 33
3.2.5.3 Rede Bayesiana 34
3.2.5.4 *Support Vector Machines* 36
3.2.5.5 Rede Neural Artificial Perceptron Multicamadas . 37
3.2.5.6 *K-Nearest Neighbor* 40
3.2.5.7 *K-Means* 41

4 Metodologia de Estudo e Avaliação 43

4.1 Trabalhos Relacionados 43
4.2 Metodologia 45
4.2.1 Pré-processamento e Extração de Características 46
4.2.2 Avaliação dos Algoritmos de AM 49
4.3 Ambiente Experimental 50
4.3.1 *Weka* 50
4.3.2 Base de dados 51

5 Resultados **55**

5.1 Seleção de Características 55
5.2 Avaliação dos Modelos de AM 56
5.2.1 Análise das Métricas de Avaliação 59
5.2.2 Otimização dos algoritmos RNA, SVM e *KMeans* 67
5.2.3 Balanceamento de Classes para os algoritmos RNA e SVM . 70
5.2.4 Balanceamento de classes no cenário 7 para os algoritmos Árvore de Decisão J48 e *Random Forest* 72

6 Conclusão e Trabalhos Futuros **75**

Apêndice A *Scripts* utilizados para pré-processamento **85**

Apêndice B Resultados Das Métricas **93**

Lista de Figuras

1.1 Metodologia para detecção de anomalias 4

2.1 Elementos de uma *botnet* genérica 8

2.2 Arquiteturas de botnet 9

2.3 Ciclo de vida de botnet 11

3.1 Processo para descoberta de conhecimento em banco de dados (KDD) 19

3.2 Hierarquia do Aprendizado 22

3.3 Exemplo de árvore de decisão, o atributo umidade é selecionado por possuir maior ganho de informação 33

3.4 Exemplo de classificação com o algoritmo Random Forest 34

3.5 (a)SVM Linear (b)SVM não-linear [1] 36

3.6 Exemplo de RNA 38

3.7 Exemplo de classificação com o algoritmo K-NN 40

4.1 Metodologia 47

5.1 *f-measure* do Random Forest em comparação com K-NN [2, 3] 65

5.2 Precisão dos modelos nos cenários 65

5.3 *Recall* dos modelos nos cenários 66

5.4 *F-measure* dos modelos nos cenários 66

5.5 Tempo de treinamento dos modelos nos cenários 67

5.6 Precisão dos modelos nos cenários após otimização dos algoritmos . . 69

5.7 *Recall* dos modelos nos cenários após otimização dos algoritmos . . . 69

5.8 *F-measure* dos modelos nos cenários após otimização dos algoritmos . 70

5.9 Precisão dos modelos nos cenários após balanceamento de classes . . 71

5.10 *Recall* dos modelos nos cenários após balanceamento de classes 72

5.11 *F-measure* dos modelos nos cenários após balanceamento de classes . 72

5.12 Precisão dos modelos do paradigma simbólico no cenário 7 antes e depois do balanceamento de classes 73

5.13 *Recall* dos modelos do paradigma simbólico no cenário 7 antes e depois do balanceamento de classes 74

5.14 *F-measure* dos modelos do paradigma simbólico no cenário 7 antes e depois do balanceamento de classes 74

Lista de Tabelas

3.1 Matriz de confusão para um problema de classificação binária 28

4.1 Comparação entre este trabalho e [2] [3] 46

4.2 Características dos cenários 53

4.3 Características dos cenários de captura de pacotes: 53

4.4 Distribuição da quantidade de fluxos para os cenários 54

5.1 Resultado da Seleção de Características 57

5.2 Principais parâmetros dos algoritmos de classificação 59

5.3 Parâmetros selecionados por cenário para os algoritmos RNA e SVM 68

B.1 Precisão dos modelos nos cenários 93

B.2 *Recall* dos modelos nos cenários 94

B.3 *F-Measure* dos modelos nos cenários 94

B.4 Tempo de treinamento dos modelos nos cenários 94

B.5 Precisão dos modelos nos cenários após otimização dos algoritmos . . 94

B.6 *Recall* dos modelos nos cenários após otimização dos algoritmos . . . 94

B.7 *F-Measure* dos modelos nos cenários após otimização dos algoritmos . 94

B.8 Precisão dos modelos nos cenários após o balanceamento de classes . 95

B.9 *Recall* dos modelos nos cenários após o balanceamento de classes . . . 95

B.10 *F-Measure* dos modelos nos cenários após o balanceamento de classes 95

B.11 Precisão dos modelos do paradigma simbólico no cenário 7 antes e depois do balanceamento de classes 95

B.12 *Recall* dos modelos do paradigma simbólico no cenário 7 antes e depois do balanceamento de classes 95

B.13 *F-Measure* dos modelos do paradigma simbólico no cenário 7 antes e depois do balanceamento de classes 95

B.14 *F-Measure* 95

Lista de Nomenclaturas

AM	Aprendizado de Máquina
CERT.br	Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança
C&C	Comando e Controle
CF	*Click Fraud*
CFS	*Correlation-based Feature Selection*
CTU	*Czech Technical University*
DDoS	*Distributed Denial of Service*
DNS	*Domain Name System*
FF	*Fast Flux*
FN	*False Negative*
FP	*False Positive*
GB	*Gigabyte*
GA	*Genetic Algorithm*
HTTP	*Hypertext Transfer Protocol*
ICMP	*Internet Control Message Protocol*
IDS	*Intrusion detection System*
IP	*Internet Protocol*
IPv4	*Internet Protocol version 4*
IPv6	*Internet Protocol version 6*
IRC	*Internet Relay Chat*
K-NN	*K Nearest Neighbors*
P2P	*Peer-to-peer*
PS	*Port Scan*
RBF	*Radial Basis Function*
RNA	Redes Neurais Artificiais
ROC	*Receiver Operating Characteristics*
SMTP	*Simple Mail Transfer Protocol*
SVM	*Support Vector Machine*
TCP	*Transmission Control Protocol*
TN	*True Negative*
TP	*True Positive*
TPR	*True Positive Rate*
TTL	*Time to Live*
UDP	*User Datagram Protocol*
VoIP	Voice over Internet Protocol
Weka	*Waikato Environment for Knowledge Analysis*

1. Introdução

1.1 Considerações Iniciais e Escopo do Trabalho

É indiscutível a crescente importância dos serviços que vêm sendo fornecidos pela Internet, aumentando a dependência do uso da rede. Em consequência, o tráfego de dados cresce de modo exponencial [4] e, com esse incremento no uso, também houve crescimento no número de incidentes de segurança. Dentre as atividades maliciosas que geram esses incidentes, estão o envio de *e-mail* não autorizado (*spam*), furto de informações, o *click fraud*, que é a fraude de pesquisas ou de outras atividades onde se é contabilizado o clique de *link*, além dos perigosos ataques distribuídos de negação de serviço (*Distributed Denial of Service*, ou DDoS), que são ataques de negação de serviço (DoS) realizados a partir de origens distribuídas. O objetivo dos ataques DDoS é interromper um serviço através do esgotamento de recursos da vítima, de forma que esta não consiga responder às solicitações legítimas. Nessas atividades maliciosas costumam ser empregadas as chamadas *botnets*, que são redes de máquinas infectadas (*bots*) e controladas por um ou mais atacantes, chamados de *botmasters* [5].

O perfil dos *hackers* mudou, não estando mais interessados apenas em atrair atenção. Agora, muitas vezes são organizações criminosas ou até nações, com objetivo de controlar os dispositivos infectados para causar prejuízo às vítimas ou obter lucro, objetivo esse que está por trás do desenvolvimento das *botnets*. O crescente número de dispositivos conectados à internet e de vulnerabilidades de *software* criam um ambiente propício para a disseminação dessas redes maliciosas. No Brasil o número de notificações de ataques DDoS recebidas pelo CERT.br em 2016 cresceu 138% com relação ao ano anterior [6]. Em 2017 o número total de incidentes reportados foi 29% maior, totalizando 833.775, sendo que 220.188 são referentes a ataques de negação de serviço. Esse número foi quatro vezes maior que as notificações de DoS

recebidas em 2016 [7] e o Brasil é o país da América Latina que mais sofre ataque DDoS, com 54% dos casos [8]. Estimativas mostram que no ano de 2016 as empresas tiveram um prejuízo global de US$ 280 bilhões devido a crimes digitais. Esse número pode ainda aumentar, já que nessa pesquisa 15% das empresas participantes tinham sofrido ataques em 2015 e 21% em 2016 [9].

Boa parte dos ataques de DDoS vem sendo realizada por intermédio de dispositivos de Internet das Coisas (*Internet of Things* ou IoT). O advento da IoT e a ampliação do uso desses dispositivos, que possuem uma inerente fraqueza de segurança, pois muitos são de baixo custo, ampliou também o número e a força dos ataques DDoS. Estes ataques, que adotam como estratégia para dificultar sua mitigação uma ampla distribuição das fontes que geram os fluxos de ataque, passam também a ser alavancados por equipamentos de IoT, transformando-os em armamento para as *botnets*.

Assim sendo, torna-se primordial garantir segurança para que as redes operem normalmente e continuem a propiciar mais serviços ao público. Todavia, diversos problemas são enfrentados na tarefa de detectar *botnets* para a adoção de medidas de segurança. Vem sendo verificada, ao longo do tempo, uma evolução constante dos mecanismos adotados pelas *botnets*, que mudam sua arquitetura e seus protocolos no intuito de evitarem ser detectadas. Também a caracterização do tráfego regular de uma rede nem sempre é um tarefa trivial, o que poderia servir como forma de diferenciá-lo de um tráfego contendo fluxos relacionados com a atividade de uma *botnet*, seja na sua disseminação ou durante o lançamento de ataques. Por exemplo, há semelhanças estatísticas entre certos tráfegos maliciosos e alguns padrões de tráfego legítimo, em especial durante momentos de pico de uso de certos serviços [10]. Além disso, os programas antivírus nem sempre são capazes de identificar e remover um *bot*, devido à sofisticação desses *malwares*. Tais fatores tornam a detecção de uma *botnet* uma tarefa árdua. É por conta dessas dificuldades que técnicas de aprendizado de máquina vêm sendo cada vez mais pesquisadas para fins de detecção de *botnets* [11]. Através desta abordagem, é possível extrair, por meio de exemplos de padrões comportamentais, um modelo de classificação de dados capaz de reconhecer fluxos de pacotes que estão associados ao tráfego malicioso proveniente destes *malwares*.

O presente trabalho busca analisar uma seleção de modelos computacionais de aprendizado de máquina para a detecção de diferentes tipos de *botnets*. Através desta análise, objetiva-se verificar qual subconjunto de características do tráfego de rede que, associado a determinados algoritmos de aprendizado de máquina, possibi-

lita detectar *botnets* de forma preventiva - ou seja, através do tráfego de comando e controle entre os *botmasters* e os *bots* -, mas também de forma reativa, quando as atividades maliciosas dos *bots* (ataques) passam a ser executadas. A melhor combinação de características do tráfego de rede e modelos de Aprendizado de Máquina podem ser implementados em um classificador que irá compor a arquitetura de um IDS real para a detecção de *botnets*.

Para se descobrir este conjunto de características, foram utilizadas três técnicas de seleção de atributos, conhecidas como Ranqueamento, *CFS* e *Wrapper*. Estas técnicas são usadas com o propósito de reduzir o gasto computacional dos algoritmos de aprendizado de máquina, assim como para melhorar a capacidade de detecção do modelo, o que é feito a partir de uma seleção de atributos que garanta um bom desempenho do sistema de detecção. Após os atributos serem selecionados, foram avaliados e comparados os desempenhos de modelos computacionais representando os diversos paradigmas do Aprendizado de Máquina, divididos em aprendizado supervisionado e não supervisionado, com o objetivo de verificar qual apresenta melhor desempenho com relação ao problema. Foram comparados os seguintes paradigmas do aprendizado supervisionado: (i) simbólico, usando Árvore de Decisão J48 e *Random Forest*); (ii) estatístico, usando Rede Bayesiana e *Support Vector Machines*; (iii) conexionista, Redes Neurais Artificiais Multicamadas do tipo Perceptron; e (iv) baseado em exemplos, usando *K-Nearest Neighbor*. Já no aprendizado não supervisionado, foi avaliado o algoritmo de *Clustering KMeans*. Além disso, o algoritmo *Random Forest* também representa a técnica de *Ensemble Learning* do tipo randomização. Estes modelos foram usados para a detecção de fluxos de pacotes decorrentes de atividades de *botnets* registradas experimentalmente em treze cenários de rede da *Czech Technical University* [12].

1.2 Descrição do Problema e Objetivos

Conforme exposto na seção anterior, ficam claras as dificuldades para se detectar *botnets*. Para a detecção de ataques e atividades maliciosas, existem os chamados Sistemas de Detecção de Intrusão (IDS, do inglês *Intrusion Detection System*). Os IDS são divididos naqueles baseados em assinaturas e naqueles baseados em anomalia. Sistemas baseados em assinaturas utilizam padrões conhecidos dos campos de cabeçalho (*header*) e de conteúdo (*payload*) dos pacotes provenientes de fluxos característicos de ataques. Esses sistemas tornam-se ineficazes para o problema a medida que o tempo passa, pois as *botnets* evoluem constantemente e utilizam crip-

tografia na comunicação como método de evasão. Sistemas baseados em detecção de anomalias mostram-se mais promissores, pois a detecção da anomalia se caracteriza pelo processo de encontrar padrões em um conjunto de dados, de maneira a permitir avaliar quando o comportamento do tráfego da rede foge ao que é considerado normal ou esperado [13]. Este tipo de detecção, baseada em anomalias, tem sido o principal alvo de pesquisas voltadas para técnicas de detecção de *botnets* [5]. Neste contexto, diversas técnicas de mineração de dados e aprendizado de máquina vêm sendo usadas, cuja metodologia é mostrada na Figura 1.1 [13].

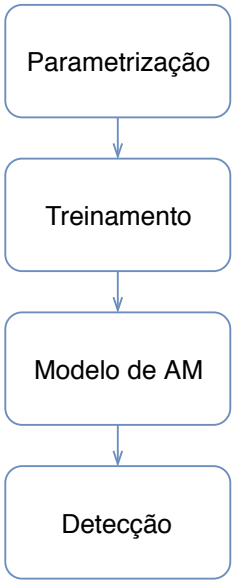

Figura 1.1: Metodologia para detecção de anomalias

Nessa abordagem, primeiramente são extraídas as características do tráfego de rede que serão relevantes para a obtenção do conhecimento necessário à detecção, sendo eliminadas redundâncias e inconsistências. Estas características também podem ser transformadas para o cálculo de novos atributos. Trata-se da fase de Parametrização, considerada de suma importância para a eficácia da detecção, conhecida também como pré-processamento. Após isso, nos casos que fazem uso de técnicas *supervisionadas* de aprendizado de máquina, ocorre um treinamento, onde é utilizado um conjunto de exemplos com o objetivo de se encontrar padrões e regras associadas a uma dada classificação de dados. É nesta etapa que são executados os algoritmos de aprendizado de máquina, com os quais são construídos os modelos que serão usados para a detecção do tráfego malicioso. Através desse processo, um IDS baseado em anomalia pode então aprender a distinguir o funcionamento normal do anormal em uma rede.

Conforme já mencionado, existem ainda duas outras abordagens no contexto da detecção de uma *botnet*: a reativa e a preventiva. A primeira, que objetiva detectar

os fluxos de ataque disparados pelos *bots*, é a mais empregada [14], porém possui desvantagens. Uma delas é o fato de ser necessário grande poder computacional para analisar uma grande quantidade de informações [14] que não para de crescer. Com o advento da Internet das coisas (*Internet of Things – IoT*), estima-se que 80 bilhões de dispositivos estejam interconectados até 2025 [15], dispositivos esses que gerarão grande massa de dados (Big Data). Outra desvantagem é quanto ao tempo para a detecção de uma ameaça. Até o momento da detecção, usuários já foram prejudicados, pois a atividade maliciosa já está em andamento. Sistemas atuais, como o *Security Information and Event Management* (SIEM), detectam 85% das intrusões depois de semanas de ocorrência [16] e a reação às ameaças é lenta, em média 123 horas depois de ocorridas [15]. Já a abordagem preventiva possui diversas vantagens, pois, ao invés de tentar detectar o ataque, investiga-se a causa raiz - ou seja, a instalação de uma *botnet* - para, então, desativá-la [14]. Porém, detectar o tráfego de comando e controle (C&C) é um desafio, pois ele se assemelha ao tráfego normal e possui baixo volume. Podem não haver muitos *bots* na rede monitorada e a comunicação entre eles pode ser criptografada. Vale ressaltar ainda que a detecção preventiva não tem relação com um sistema de prevenção de intrusões, do inglês *Intrusion prevention system* (IPS), que além de monitorar o tráfego de rede e identificar atividades maliciosas - funções de um IDS -, toma decisões como bloquear ou interromper essas atividades, encerrando uma sessão de usuário ou reprogramando o *firewall*, por exemplo. O IPS é considerado uma solução de segurança ativa, enquanto o IDS é considerado um sistema passivo.

Ao longo dos últimos anos, as *botnets* ganharam atenção dos pesquisadores de todo o mundo. Um grande esforço foi dado ao desenvolvimento de sistemas capazes de detectar de forma eficiente e efetiva a presença de uma *botnet*. Para resolver esse problema, os pesquisadores começaram a utilizar Aprendizado de Máquina, que foi considerada a técnica mais efetiva para a detecção de *botnets* [17] [11] [18] [19]. Na literatura existem várias abordagens baseadas em aprendizado de máquina sendo propostas e trabalhos em cima de seleção de características do tráfego de rede para alimentarem esses modelos. O problema é a dificuldade ou quase inviabilidade na comparação desses modelos, visto que os trabalhos utilizam conjunto de dados (*datasets*), algoritmos e métodos de avaliação diferentes. Além disso, não foram identificados trabalhos que comparem os diversos algoritmos do estado da arte de classificação englobando os diversos paradigmas, incluindo aprendizado supervisionado e não supervisionado. Sendo assim, o objetivo deste trabalho, é descobrir se existe um algoritmo que satisfaz todos os cenários de *botnets* ou se uma combinação destas técnicas deve ser usada para uma classificação mais eficaz. Com isso,

espera-se determinar a combinação de características e técnicas de Aprendizado de Máquina ideais para o problema da detecção de *botnets*, onde os modelos gerados poderão ser implementados em uma arquitetura real.

1.3 Contribuições do Trabalho

Como contribuições deste trabalho destacam-se, primeiramente, uma seleção de características do tráfego de rede relevantes para a detecção de *botnets* de arquiteturas centralizada e descentralizada, sendo estas de diversas famílias (Neris, Rbot, Virut, Menti, Shogou, Murlo e NSIS.ay), executando diferentes atividades maliciosas (*Spam*, escaneamento de portas, fraude de clique e *DDoS*) e também atuando na fase de pré-ataque, quando ocorrem sincronizações entre *bots* e *botmasters*. Nesta primeira fase do trabalho, foram utilizadas técnicas de seleção de atributos para encontrar o subconjunto de características mais relevantes para os cenários estudados. No final desta seleção, os resultados foram comparados com trabalhos anteriores, ratificando algumas características e contribuindo com novas.

Numa etapa posterior, os subconjuntos de características encontrados na fase de seleção foram utilizados nos variados modelos de aprendizado de máquina, sendo realizada uma extensa análise comparativa do desempenho e da eficiência dos algoritmos do estado da arte de diversos paradigmas do aprendizado de máquina supervisionado e não supervisionado, algoritmos esses que são os mais representativos e utilizados nos trabalhos de detecção de *botnets* através de AM [2, 3]. Reunindo esta comparação em um único trabalho e utilizando a mesma metodologia, contribui-se em descobrir se existe um algoritmo ideal para os inúmeros cenários de detecção de *botnets* ou se uma combinação dessas técnicas deve ser usada para uma classificação mais eficaz em diferentes casos. Ainda convém lembrar que, tanto as características relevantes encontradas quanto os modelos de aprendizado de máquina que se destacaram, poderão ser implementados em uma arquitetura real.

1.4 Estrutura da Dissertação

A organização da dissertação segue no seguinte formato. No Capítulo 2, são abordados os principais conceitos de *botnets*: arquitetura, ciclo de vida, técnicas de evasão, atividades maliciosas e sistemas de detecção de intrusão. No Capítulo 3, são explanados os conceitos de descoberta de conhecimento em banco de dados, mine-

ração de dados, aprendizado de máquina supervisionado e não supervisionado, paradigmas do aprendizado de máquina, métodos de avaliação de algoritmos, técnicas de seleção de atributos e algoritmos de classificação e agrupamento. No Capítulo 4, são apresentados os trabalhos relacionados, a metodologia usada e o ambiente experimental. No Capítulo 5, são apresentados os resultados, nos quais as características do tráfego de rede que são mais relevantes à detecção de *botnets* são selecionadas, através das técnicas de seleção de atributos, e os modelos de aprendizado de máquina são utilizados para classificação. Esses resultados são analisados a partir de métricas recomendadas para melhor avaliar bases de dados que possuam classes desbalanceadas, ou seja, nas quais as quantidades de elementos de cada classe diferem substancialmente. No Capítulo 7, tem-se a conclusão, as considerações finais e os trabalhos futuros.

2. Fundamentos de Botnets

2.1 *Botnets*

Botnet é uma rede de computadores infectados controlados por um ou mais atacantes para atingir determinados objetivos que são a execução de atividades maliciosas. Os usuários cujos dispositivos estão infectados e fazem parte desta rede geralmente estão ignorantes do fato de estarem participando indiretamente dessas atividades maliciosas [19]. As *botnets* podem ser usadas para uma variedade de atividades ilícitas como Spam, ataques *DDoS*, distribuição de software malicioso (*trojan*, *spyware* e *keylogger*), pirataria de software, roubo de informações, extorsão, roubo de identidade, manipulação de jogos e pesquisas online, dentre outros [14] [20] [21] [5]. Com o advento da *IoT*, o aumento do número de dispositivos conectados e de vulnerabilidades fazem o ambiente propício para a disseminação destas redes. Redes essas que possuem enorme potencial destrutivo, podendo afirmar que são uma das maiores ameaças da Internet [20] [21] [5].

Sendo assim, uma *botnet* é composta pelos seguintes agentes [5] como mostra a Figura 2.1.

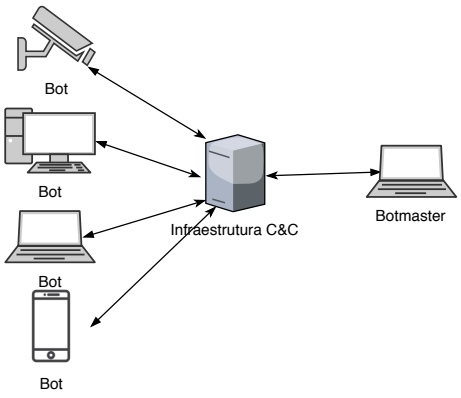

Figura 2.1: Elementos de uma *botnet* genérica

- **Bot** é o *malware* instalado na máquina da vítima, que permite a execução de ações maliciosas, podem ser transmitidos de diversas maneiras como vírus, sites infectados e pela rede.

- **Botmaster** é o indivíduo (*hacker*), organização ou até mesmo uma nação que possui controle da *botnet* e pode ordenar a execução de atividades maliciosas para estes *Bots*.

- **Canal de comando e controle (C&C)** é o meio através do qual os *botmasters* enviam comandos aos *bots*, podendo ser um servidor central, por exemplo utilizando o protocolo IRC, ou distribuído, fazendo uso de redes *peer-to-peer* (P2P);

- e a **Botnet** em si, que é a rede de *bots* conectadas ao centro de comando e controle, aguardando ordens do *botmaster*.

2.1.1 Arquitetura

O canal de Comando e Controle (C&C) é o *backbone* da *botnet* e pode ser classificado de acordo com seu modo de operação e arquitetura : centralizada, descentralizada ou híbridas [17] [5]. Conforme a Figura 2.2.

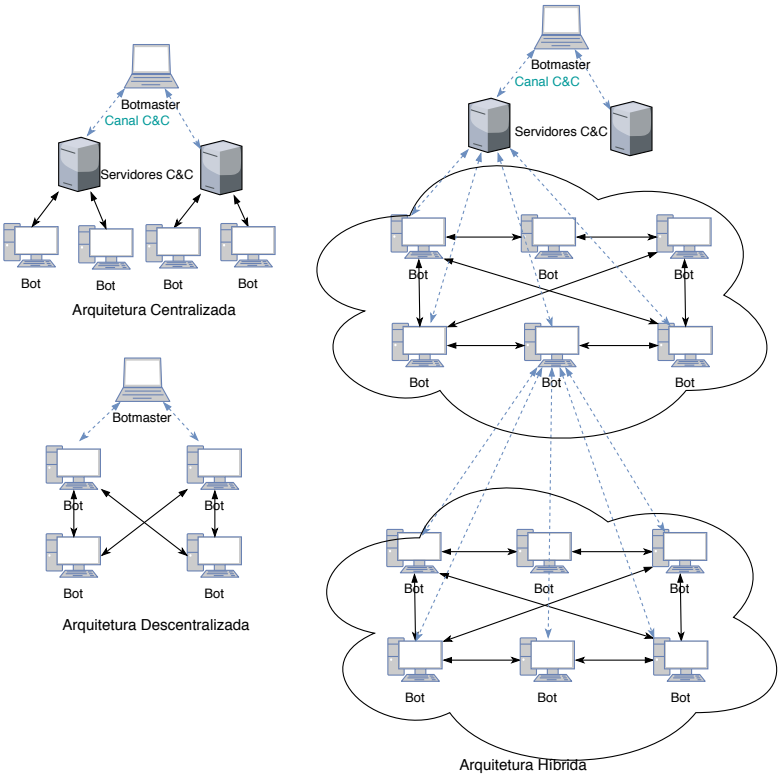

Figura 2.2: Arquiteturas de botnet

.

- **Centralizada**: essa abordagem é semelhante ao modelo clássico de rede cliente-servidor. Nesta arquitetura, os bots estabelecem o canal de comunicação com um ou alguns pontos de conexão que são os servidores C&C. Pode-se utilizar por exemplo o protocolo IRC [22]. Geralmente, servidores C&C enviam comandos para os *bots* e fornecem atualizações de *malware*.

- **Descentralizada**: as *botnets* modernas precisam de flexibilidade e robustez para lidar com o grande número de *bots*. Encontraram nesta arquitetura descentralizada onde a comunicação é realizada de forma distribuída baseada em protocolos de redes *P2P (Peer-to-Peer)*. Essa forma de comunicação dificulta a detecção e também provê resiliência, já que não há mais um ponto central de falha [23] .

- **Híbrida**: essa abordagem combina os princípios das arquiteturas Centralizada e Descentralizada, usando protocolos P2P híbridos junto com o baixo *delay* de rede encontrado em arquiteturas centralizadas.

2.1.2 Ciclo de Vida

As *botnets* possuem um ciclo de vida, com fases definidas desde o momento em que a vítima é infectada até esta fazer parte da *botnet* e executar as atividades maliciosas a que for destinada [5]. Este ciclo é composto das seguintes fases: infecção inicial, infecção secundária, conexão, realização de atividades maliciosas e manutenção e atualização, mostradas na Figura 2.3.

Primeiramente, na Infecção Inicial, o hacker infecta o *host* através da exploração de uma vulnerabilidade ou a vítima executa o *malware*, nesta etapa a máquina torna-se um potencial *bot*. A segunda fase requer que a primeira tenha completado com sucesso, o computador infectado executa um programa que busca por códigos binários do *malware* em um repositório, esse download pode ser feito por *FTP*, *HTTP* ou *P2P*. Quando o download é finalizado e executado, esse *host* passa a se comportar como um *bot* ou zumbi. A próxima fase é a conexão ou *rally*, em que a máquina zumbi entra em contato com o servidor de Comando e Controle (C&C), isso ocorre sempre que é reiniciada, onde ela informa que está ativa na *botnet*. Nesta etapa está estabelecido o canal de Comando e Controle (C&C), o *bot* então aguarda o recebimento de comandos para executar as atividades maliciosas (Fase 4). Na última fase, são realizados procedimentos para preservar a *botnet*. Isso inclui atualização

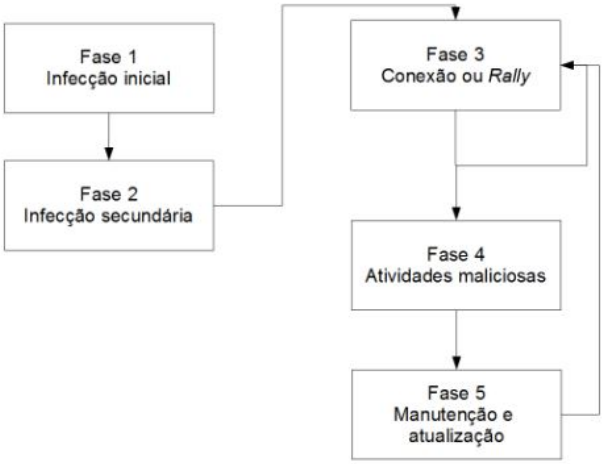

Figura 2.3: Ciclo de vida de botnet

de códigos binários para adicionar novas funcionalidades, evadir-se da detecção ou migrar de servidor C&C [24] [20] [5].

Nas fases de conexão, de realização de atividades maliciosas e de manutenção e atualização, a vítima entra em contato com os servidores de C&C. Estas fases são semelhantes, independente do tipo de *botnet*, e são indicativos de que a atividade maliciosa de uma *botnet*, tanto na fase de controle quanto na fase de ataque em si, possuem padrões comportamentais que podem ser sistematicamente aprendidos para fins de detecção.

2.1.3 Métodos de evasão

Os métodos de evasão são considerados parte do ciclo de vida da *Botnet*, pertencente ao último estágio. Essas técnicas servem para evitar a detecção das *botnets* e seus componentes. Existem diversas técnicas, as seguintes são as mais usadas [25]:

- ***Multihopping***: o hacker se conecta a vários proxies, localizados em países onde o acesso aos logs é dificultado por restrições legais. Com isso, dificulta-se o rastreamento do endereço IP final. Esta técnica geralmente é usada na comunicação C&C com os *bots*.
- **Criptografia**: o Canal de C&C é criptografado para evitar que seja analisado, assim fica mais difícil o desenvolvimento de técnicas para detecção deste tráfego e inviabiliza a detecção por assinatura, já que o conteúdo do pacote não pode ser inspecionado.

- **Ofuscação dos arquivos binários**: os programadores utilizam esta técnica para esconder o código fonte do *bot* [26], assim evitam a execução de uma engenharia reversa e análise do comportamento do *bot*.

- **Polimorfismo**: consiste em criar diferentes versões do código-fonte sem alterar a funcionalidade, esta técnica dificulta a detecção pelos programas antivírus já que a maioria usa detecção por assinatura.

- **Tunelamento**: este método mascara o tráfego real, encapsulando um protocolo dentro de outro. Podem ser através dos protocolos HTTP, ICMP, VoIP [27] ou IPV6 [28].

- ***IP Spoofing***: baseia-se em enviar pacotes IP com endereço de origem falso, é bastante usado em ataques *DoS* para enganar *blacklists* de endereços IP.

- ***E-mail Spoofing***: Semelhante à técnica anterior, dá-se em enviar *email* com um falso remetente ou outro campo do cabeçalho. Geralmente é usado para ataques de *phishing*.

- ***Fast-flux***: Este método se dá em utilizar diversos *proxies* para redirecionamento. Os *proxies* mudam constantemente com o uso de registros DNS com baixo TTL, assim têm-se múltiplos endereços IP associados a um nome de domínio, em seguida, são realizadas trocas rápidas sucessivas. Essa técnica é utilizada para esconder os servidores C&C. O *bot Storm* foi o primeiro a utilizar, ele implementou este mecanismo para ocultar a atualização dos arquivos binários [29].

2.1.4 Atividades Maliciosas

As botnets tem o propósito de executar alguma atividade maliciosa, atividades essas que contam com um enorme número de participantes. Dentre as inúmeras ações maliciosas [30], as principais atividades são [25] [31]:

- **Comprometimento de novos *hosts***: para fortalecer as *botnets*, o *botmaster* pode recrutar novos *hosts* utilizando técnicas de engenharia social, *phishing* e envio de e-*mails* maliciosos.

- ***DDoS***: o ataque *DoS* tenta impedir ou reduzir o uso legítimo de um serviço, sobrecarregando o alvo através de envio massivo de pacotes, impactando na disponibilidade. Já no *DDoS*, múltiplos atacantes agem simultaneamente para

atingir este objetivo [32]. A *botnet* sempre possui um conjunto de mecanismos para execução de *flood* como SYN *Flood*, ICMP *Flood* e HTTP *Flood* ou pode também enviar milhares de solicitações *http* e *ftp* legítimas [31], imitando o comportamento de milhares de clientes legítimos, tornando extremamente difícil criar esquemas de defesa contra um *DDoS* lançado por *botnets*.

- ***Spamming***: spam é *e-mail* não solicitado criado para ser entregue a um grande número de destinatários [33]. O uso de uma botnet aumenta consideravelmente o poder de enviar uma grande quantidade de e-mails de spam em alguns segundos [25]. As *botnets* podem usar servidores *SMTP* para enviar spam, a maioria dos spam de *email* hoje é enviada por *botnet* [31].

- ***Phishing***: é a criação de uma réplica de página web existente ou outro recurso online para enganar usuários, com o objetivo que eles submetam dados pessoais, financeiros ou senhas [34]. *Bots* podem ser usados para hospedar sites de *phishing* [31]. Para esconder estas réplicas os hackers utilizam a técnica fast-flux [25].

- **Furto de dados**: o *botmaster* pode furtar senhas e informações confidenciais dos usuários infectados com os *bots*. Podem ser usadas diversas técnicas como captura de tela, furto de senha, *upload* de arquivos, *keyloggers*, sequestro de cookie e etc.

- **Fraude de clique**: as propagandas online são cobradas por cliques em anúncios ou número de visitas ao site. Esta técnica consiste em enganar usuários para induzi-los a clicar em anúncios ou visitar algum site, aumentando a receita de terceiros [35]. O uso de *botnets* torna possível simular o comportamento de milhões de usuários legítimos [36].

2.2 Detecção de *Botnets*

Tendo em vista os aspectos observados, dado o potencial das *botnets*, as técnicas de detecção são muito importantes para coibir essa ameaça e se tornaram um importante tópico de pesquisa nos últimos anos. Os pesquisadores desenvolveram diversas arquiteturas e propuseram uma série de taxonomias de detecção de *botnet* [20] [37]. As técnicas de detecção são divididas em duas abordagens. Baseadas em *honeynets*, necessárias para entender a tecnologia e as características da *botnet*, mas não necessariamente detectar a infecção. Baseadas em monitoramento e análise do tráfego de rede através de sistemas de detecção de intrusão (*IDS*). Os *IDS* podem

ser classificados por baseados em assinatura ou anomalia, ainda de acordo com a fonte de auditoria podem ser classificados em baseados em *host* ou rede [5] [31]. Todas essas técnicas de detecção serão descritas nesta seção.

2.2.1 Detecção baseada em *Honeynet*

Honeypot é um sistema que atua como chamariz no intuito de atrair atenção de Hackers para que o ataquem com o objetivo de proteger alvos críticos [38]. São sistemas que não possuem valor de produção, como não esperam nenhum dado, qualquer tráfego de entrada e saída é provavelmente atividade suspeita e deve ser investigada [31]. Uma *honeynet* é uma rede de *Honeypots* concebida para ser comprometida e coletar informações dos *bots*, porém possui mecanismos para que não seja utilizada para atacar outras redes [39]. Após essa coleta, é possível compreender a tecnologia e analisar características da *botnet* como: assinatura de *bots* para detecção baseada em conteúdo, informação sobre a arquitetura C&C e vulnerabilidades desconhecidas que permitam *bots* entrar na rede [37]. As *Honeynets* também podem ser usadas para obter binários do *bot* e se infiltrar nas *botnets* [14].

As *Honeynets* possuem as seguintes limitações [5]:

- Podem monitorar um número limitado de atividades.
- Não conseguem capturar *bots* que não usam métodos de propagação ou que usem métodos que necessitem interação com o usuário como download e spam.
- Só podem informar sobre máquinas infectadas que estão sendo usadas como armadilha.
- Com a sua popularidade, os hackers começaram a encontrar maneiras de evitar estas armadilhas [40].

Por todos esses aspectos, essa técnica ainda é bastante eficaz para coletar informações de *botnets* mas não necessariamente detectar a infecção.

2.2.2 Detecção baseada em IDS

IDS (*Intrusion Detection System*) são sistemas que monitoram, identificam e notificam atividades maliciosas e não-autorizadas que estejam ocorrendo em uma rede de computadores. Estes sistemas podem ser baseados em *host* ou rede e quanto à forma de detecção podem ser baseados em assinaturas ou anomalias.

2.2.2.1 *IDS* Baseados em *Host (HIDS)*

Estes *IDS* buscam detectar infecções de *bot* em um *host* individualmente, analisando o comportamento da máquina, já que um *bot* quando está sendo executado realiza sequências de chamadas para bibliotecas do sistema como registro, sistema de arquivos e conexões de rede diferentes das chamadas executadas pelos processos legítimos [41]. IDS Baseados em *Host* usam métodos para correlacionar o tráfego de rede e eventos do sistema com assinaturas de *bot* ou informações de comportamento dependendo do tipo de *IDS* [42]. Estes sistemas têm as seguintes vantagens: fáceis de implantar, beneficiam o usuário final [43] e são mais eficazes contra ataques de download e infecções iniciais [44]. Essa abordagem permite detectar infecções de um único *bot* e é importante para minimizar a propagação, porém analisar e monitorar estações individuais é uma tarefa complexa, custosa e não escalável [5].

2.2.2.2 *IDS* Baseados em *Rede (NIDS)*

Esses IDS podem utilizar técnicas de monitoramento desenvolvidas especificamente para um protocolo ou técnicas genéricas abrangendo diversas arquiteturas e protocolos, eles podem ser classificados em monitoramento ativo e passivo [5]:

- **Monitoramento Ativo**: Neste modo são injetados pacotes na rede para determinar se quem está gerenciando a comunicação é um humano ou *bot*, já que os *bots* possuem um padrão de comunicação não humano, respondendo a comandos predefinidos com respostas constantes cada vez que o comando é repetido. A desvantagem desse método é o aumento do tráfego de rede devido aos pacotes adicionais que são direcionados às máquinas suspeitas, além de facilitar o uso de ferramentas de rastreamento o que está sujeito a problemas legais [5].
- **Monitoramento Passivo**: Esta técnica tem se mostrado bastante útil para identificar *botnets* [45]. Consiste em monitorar o tráfego de rede em busca de comunicações suspeitas, comunicações essas que podem ser oriundas de *bots* ou servidores C&C. Esse tráfego pode ser analisado através de técnicas de detecção baseada em Assinatura ou baseada em Anomalia. O monitoramento passivo utiliza a ideia de que *bots* da mesma *botnet* tendem a apresentar padrões de comunicação semelhantes independente da arquitetura da *botnet* [41]. Como os botmasters precisam se comunicar com os bots para a execução das atividades maliciosas, esse tráfego apresenta um padrão comum referente a cada fase do ciclo de vida da botnet [5].

2.2.2.3 Detecção Baseada em Assinatura

Esta técnica é constituída na extração de informações de pacotes do tráfego monitorado e no registro desses padrões em um banco de dados de *bots* existentes. Aparentemente parece simples realizar as detecções simplesmente comparando cada *byte* no pacote, mas essa técnica possui graves desvantagens [46][5]:

- Não é possível detectar *bots* desconhecidos (*zero-day*), pois não foram mapeados.
- Deve-se sempre atualizar a base de conhecimento com novas assinaturas, o que reduz o desempenho e aumenta o custo de gerenciamento.
- Novos *bots* podem realizar atividades maliciosas antes que a base de conhecimento seja atualizada.
- Em redes com grande tráfego, haverá dificuldade para processar todos os pacotes.
- Essa técnica não permite que seja inspecionado o tráfego criptografado.

2.2.2.4 Detecção Baseada em Anomalia

Detecção baseada em anomalia é o processo de encontrar padrões em um conjunto de dados cujo comportamento foge do normal ou esperado. Essas anomalias nem sempre são ataques, podem ser um comportamento que antes não era conhecido. Esse processo é importante para várias aplicações críticas como detecção de fraudes em cartões de crédito, distúrbios no Ecossistema e Detecção de Intrusão. Existem diversas abordagens para detecção de anomalia, utilizando estatística, aprendizado de máquina e mineração de dados. Quando os dados precisam ser analisados para encontrar relacionamentos ou realizar previsões, as técnicas de mineração de dados são utilizadas [13] [47].

Essa é a principal área de pesquisa em técnicas de detecção de *botnets* [5]. Para realizar essa tarefa são consideradas diferentes anomalias de tráfego de rede como alta latência, alto volume de tráfego, tráfego em portas incomuns ou qualquer comportamento incomum que possa ser um indício de *bots* na rede. Essa técnica tem como desvantagens a alta taxa de falso-positivos e a complexidade para se determinar quais características do tráfego de rede devem ser utilizadas [31].

2.2.3 Defesa contra *botnets*

Após a detecção, é necessário parar a *botnet*. Como há muitas máquinas infectadas, neutralizar um único *bot* não irá resolver o problema. Métodos mais efetivos buscam desativar toda a rede maliciosa [14, 23]. Esses métodos se baseiam no combate à propagação e comunicação da *botnet*. O combate à propagação reduz o número de dispositivos comprometidos, limitando o poder da *botnet*. Ao parar a comunicação entre os dispositivos infectados e os servidores de C&C, os *botmasters* não conseguirão enviar comandos e receber as respostas dos *bots* [5, 14].

Existem três principais formas de se efetivar uma defesa contra *botnets*: prevenção, tratamento e contenção [48]. Essas medidas devem ser tomadas em conjunto por usuários, administradores de rede e provedores de Internet [5]. As técnicas de prevenção à propagação das *botnets* têm o objetivo de reduzir a população vulnerável, limitar a disseminação de *worms* e diminuir o tamanho da *botnet*. Como exemplo dessas técnicas, temos o desenvolvimento seguro de *software*, sistemas com as atualizações de segurança em dia, remoção de vulnerabilidades existentes, uso de programas antivírus e treinamento do usuário [5, 48]. A forma de defesa por tratamento corresponde à desinfecção de dispositivos infectados para reduzir o número de *bots*. O tempo necessário para que sejam desenvolvidos novos mecanismos e atualizações, além do tempo para que sejam realizados testes afim de verificar a efetividade desses métodos em desinfetar máquinas zumbis, é limitado pela escala de tempo humano, que às vezes não reage com rapidez suficiente para combater atividades maliciosas de *botnets* e epidemias de disseminação de *worms* [48]. A contenção possui duas fases: detecção das *botnets* e resposta. Na fase de detecção, monitoram-se os dispositivos ou monitora-se a rede - por exemplo, por intermédio de um IDS. Já na fase de resposta, são usados mecanismos para bloquear o tráfego entre os *bots* e os servidores de C&C, com o objetivo final de desativar esses servidores. O bloqueio pode ser automatizado após a detecção do *bot*, sem depender da ação humana. Para isso, podem ser utilizados sistemas automatizados integrados com *firewalls*, filtros de conteúdo, *blacklists* de endereços e rotas nulas para bloquear comunicações entre os dispositivos infectados e também para bloquear a propagação dos *malwares*. Sendo assim, uma infecção pode ser interrompida ou reduzida, a comunicação da *botnet* pode ser paralisada ou o centro de C&C pode ser desativado [5]. Essas estratégias podem ser implementadas diretamente na rede sem a necessidade de uma solução para cada *host* da Internet.

Neste trabalho, os algoritmos estão localizados dentro da arquitetura de um IDS

que estará monitorando uma rede na fase de detecção. Os testes foram realizados com a intenção de se verificar, a partir do tráfego de rede de uma instituição, atividades oriundas de *botnets* localizadas no ambiente interno ou externo à rede dessa instituição.

3. Fundamentos de KDD e Aprendizado de Máquina

3.1 Descoberta de Conhecimento em Banco de Dados

Também conhecida como KDD, do inglês *Knowledge Discovery in Databases*, é o processo de conversão de dados brutos, presentes em grandes bancos de dados, para informações úteis. Esse processo possui diversas fases de transformação dos dados, iniciando no pré-processamento e finalizando no pós-processamento. Essas fases podem ser vistas na Figura 3.1[47].

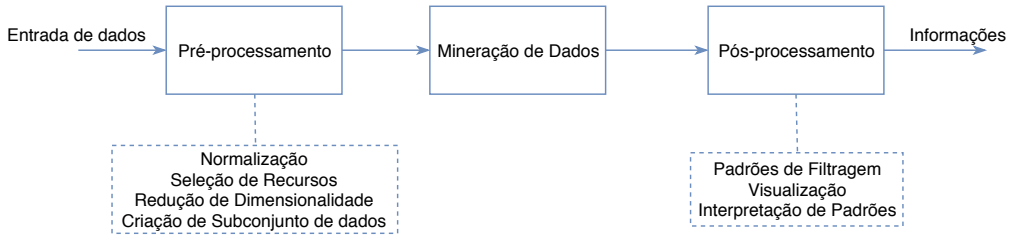

Figura 3.1: Processo para descoberta de conhecimento em banco de dados (KDD)

Na entrada de dados podem ser utilizados os mais diversos formatos de arquivos, como texto simples, planilhas e tabelas relacionais. Esses dados podem estar centralizados ou distribuídos em inúmeros repositórios. O objetivo da fase de pré-processamento é realizar a transformação dos dados brutos para o formato que será utilizado no processo, assim os dados são agrupados e organizados de acordo com o domínio do problema. Nessa primeira fase, podem ser realizadas a fusão de dados de diversos repositórios, cálculos de novos atributos a partir dos dados originais, assim como a limpeza dos dados para remover ruídos, inconsistências e observações redundantes. Nesta fase costuma ser realizada uma seleção de características relevantes à tarefa, onde informações não necessárias para o domínio do problema são removidas e apenas características relevantes são manipuladas na base de dados. Além disso, os dados são formatados para que possam ser utilizados nas técnicas e algoritmos selecionados. Essa é a fase mais trabalhosa e demorada, pois é necessária

muita atenção para selecionar os dados relevantes [47]. Na última fase, denominada pós-processamento, são selecionados apenas os resultados úteis e válidos para o problema, resultados esses que geram conhecimento para interpretação, avaliação e validação do usuário [47].

3.1.1 Mineração de Dados

A mineração de dados é a fase intermediária do processo de descoberta de conhecimento em banco de dados, como visto na Figura 3.1. É nessa fase que as informações úteis são extraídas. As técnicas de mineração de dados são idealizadas para atuar em grandes bases de dados com o objetivo de descobrir, através de análises matemáticas, padrões úteis que poderiam ser ignorados ou realizar previsões sobre uma observação futura. Esses padrões não poderiam ser descobertos com uma análise tradicional devido à complexidade das relações e à grande quantidade de dados, por isso são aplicados métodos que envolvem aprendizado de máquina, estatística e sistemas de banco de dados. Os padrões que forem extraídos devem então ser transformados em uma estrutura compreensível para posterior uso. As quatro tarefas centrais da mineração de dados são [47]:

- **Análise de Agrupamentos**: O objetivo é encontrar e aproximar observações relacionadas, de forma que as pertencentes a um mesmo grupo sejam mais similares entre si do que as pertencentes a outros grupos. O nome desses agrupamentos é *cluster*. São exemplos de uso: reconhecer de padrões, agrupar conjuntos de clientes relacionados, detectar fraudes, descobrir áreas do oceano que possuam impacto sobre o clima e compactar dados, entre outros.
- **Modelagem Previsiva**: A finalidade é construir um modelo para uma variável alvo como sendo uma função de variáveis explicativas. Esse modelo deve minimizar o erro entre a previsão e o valor real da variável alvo. Nessa modelagem há dois tipos de tarefa: classificação, na qual a variável alvo deve ser discreta, e regressão, na qual a variável alvo deve ser contínua. São exemplos de uso: julgar se um paciente possui determinada doença, prever se um usuário fará uma compra e identificar se uma conexão de rede é maliciosa são exemplos de classificação, já que a variável alvo é de valor binário; prever o valor futuro de uma ação e estimar o valor a ser gasto em uma obra são exemplos de regressão, já que a variável alvo é de valor contínuo.
- **Análise de Associação**: O propósito é descobrir padrões que representem características fortemente associadas. Os padrões descobertos geralmente são

representados como regras de implicação ou subconjunto de características. O espaço de busca possui tamanho exponencial, então procura-se obter os padrões mais importantes de modo eficiente. Essas regras podem ser representadas da seguinte forma: se *atributo A* então *atributo B*. São exemplos de uso: produtos que são levados junto em cestas de compras e descoberta de páginas *web* que são acessadas juntas.

- **Detecção de Anomalias**: O objetivo é a identificação de observações que destoem bastante das outras, essas observações são chamadas de anomalias. O algoritmo de detecção de anomalias deve descobrir as anomalias verdadeiras, buscando alta taxa de verdadeiro-positivos e baixa taxa de falso-positivos. São exemplos de uso: pertubações no meio ambiente e detecção de doenças incomuns.

3.2 Aprendizado de Máquina

Na fase de Mineração de Dados são usados algoritmos de Aprendizado de Máquina, também conhecido como *Machine Learning*. É uma área da Inteligência Artificial que desenvolve técnicas computacionais sobre o aprendizado e constrói sistemas com habilidade de aprender de forma automática e progressiva, similar ao comportamento humano. Um sistema de aprendizado de máquina é capaz de tomar decisões apoiado em experiências obtidas na solução de problemas anteriores. A aprendizagem possui três componentes. Um deles é a representação, no qual o sistema de aprendizado precisa achar um conjunto de modelos que tenham bom desempenho em generalizar os dados. Se um modelo não pertence ao conjunto de hipóteses, ele não será utilizado. Outro componente é a métrica, obtida através de uma função de avaliação, que será usada para discernir os modelos que apresentem bom desempenho na resolução do problema. Por último, tem-se a otimização. Nesse componente são usados métodos para encontrar a melhor configuração para os parâmetros de cada modelo. Como mencionado na introdução desta Dissertação, o aprendizado de máquina foi considerada a técnica mais efetiva para a detecção de *botnets*. Embora essa técnica seja bastante sofisticada, não existe um algoritmo específico que apresente o melhor desempenho em todos os problemas. Deve-se, portanto, utilizar uma metodologia para avaliar e comparar os algoritmos. Esses sistemas podem ser divididos quanto a modo, paradigma e forma de aprendizado utilizados [49, 50].

3.2.1 Hierarquia do Aprendizado

A indução é um raciocínio ao qual se estende uma propriedade a todos os termos de um conjunto, ou seja, da parte para o todo. Inicia-se em um conceito específico e parte para conclusões genéricas. A inferência indutiva é um dos principais métodos para extrair conhecimento e prever eventos, porém deve ser utilizado com precaução. Se o número de exemplos não for suficiente ou representativo, as hipóteses podem não ter valor. Aprendizado indutivo é realizado através de exemplos externos ao sistema de aprendizado, que pode ser supervisionado ou não-supervisionado. Essa hierarquia é vista na Figura 3.2 [49].

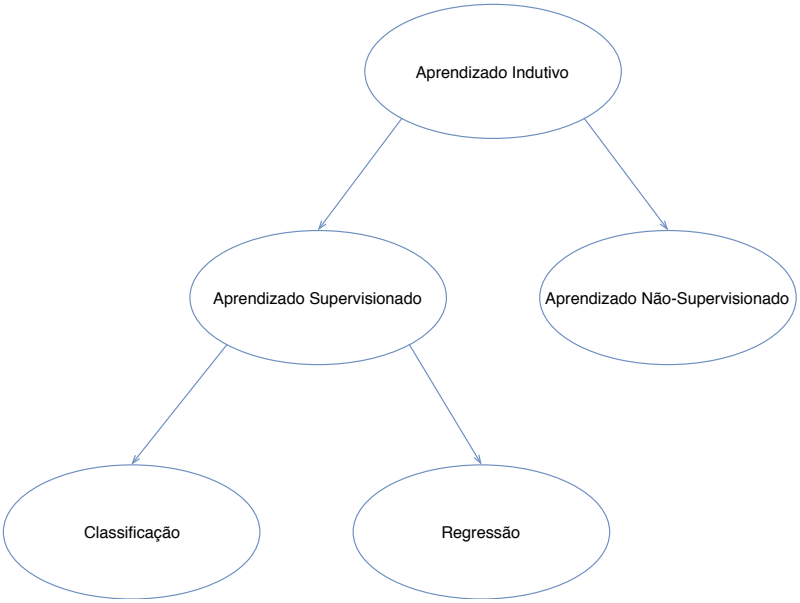

Figura 3.2: Hierarquia do Aprendizado

No aprendizado supervisionado é utilizado um algoritmo de indução que irá realizar um treinamento em um conjunto de instâncias rotuladas com a classe devida. Cada instância é composta por um vetor de valores de características ou atributos e o rótulo da classe. O objetivo é construir um classificador que consiga acertar a classe de novos exemplos não rotulados. Se o valor da classe for discreto, esse aprendizado é conhecido como classificação. A classificação pode ser entre duas classes ou multiclasses, quando há mais categorias. Se o valor da classe for contínuo, o aprendizado é do tipo regressão [49]. Por exemplo, neste trabalho cada instância usada para treinamento é rotulada como maliciosa ou normal. Assim sendo, o algoritmo de aprendizado supervisionado procura por padrões nesses rótulos através das características selecionadas, que podem ser o tamanho do pacote, a duração ou qualquer outra informação da conexão de rede. Após o algoritmo ser treinado, o padrão encontrado será utilizado para realizar previsões para instâncias de teste sem

rótulos.

Já no aprendizado não-supervisionado, o algoritmo indutor analisa as instâncias fornecidas, não rotuladas, e tenta agrupá-las, se possível, de alguma maneira. Com isso são formados os agrupamentos ou *clusters*, agrupamentos esses que devem ser analisados para entender o significado de cada um no contexto do problema trabalhado [49]. Por exemplo, neste trabalho, as instâncias sem o rótulo de classe são fornecidas ao algoritmo de aprendizado não-supervisionado. Através das características selecionadas esse algoritmo deve agrupá-las em dois *clusters*: normal e malicioso.

3.2.2 Paradigmas do Aprendizado

Diversos paradigmas de AM foram propostos. Os mais conhecidos [49] são descritos nesta subseção:

- **Paradigma Estatístico**: O fundamento deste paradigma é utilizar modelos estatísticos para encontrar uma boa aproximação do conceito induzido. Esses métodos possuem parâmetros, então pode-se realizar uma parametrização, encontrando valores apropriados para os parâmetros do modelo a partir dos dados. Métodos estatísticos que se destacam são os de aprendizado Bayesiano. Esses métodos utilizam o teorema de Bayes através de um modelo probabilístico baseado no conhecimento *a priori* do problema. Esse modelo é combinado com exemplos de treinamento para determinar a probabilidade final de uma hipótese. Exemplos de técnicas que utilizam esse conceito são *Naive Bayes* e *Redes Bayesianas*.

- **Paradigma Simbólico**: Neste paradigma o conhecimento é construído através de uma estrutura simbólica por meio da análise de exemplos e contra-exemplos de um conceito. Essas estruturas são representações do conhecimento de alto nível e aparecem na forma de uma expressão lógica, árvore de decisão, regras ou rede semântica.

- **Paradigma Baseado em Exemplos**: Exemplos ou instâncias nunca vistas são classificados através de exemplos conhecidos. Um novo caso será classificado através de um caso similar cuja classe é conhecida. Esse aprendizado é conhecido como *lazy*, pois é preciso manter os exemplos na memória para realizar novas classificações. O oposto a esse sistema é conhecido como *eager*, pois os exemplos são utilizados para induzir o modelo e descartados após

esse processo. A questão mais importante é saber quais exemplos são os mais representativos para serem memorizados por um indutor *lazy* no processo de treinamento. O algorimo *k-nearest neighbors (k-NN)* é um dos mais conhecidos desse paradigma.

- **Paradigma Conexionista**: Esta área estuda as Redes Neurais, que são construções matemáticas inspiradas no modelo biológico do sistema nervoso humano. Esses modelos utilizam unidades processadoras interconectadas, os neurônios. Essa abordagem desperta o interesse de pesquisadores de diversas áreas. A analogia com a biologia vem indicando que as redes neurais possuem grande potencial para resolver problemas que necessitam de processamento sensorial humano. As redes neurais artificiais perceptron com uma camada ou multicamadas são exemplos de técnicas que utilizam esse paradigma.

- **Paradigma Genético ou Evolutivo**: Uma derivação do modelo evolucionário de aprendizado, um modelo genético é formado por uma população de elementos de classificação que competem entre si para realizar a predição. Inicia-se com uma população de elementos onde cada um representa uma solução possível. Esses elementos então competem entre si e os que apresentarem menor desempenho são descartados. Os melhores são selecionados para reprodução, ou *crossover*, e os novos elementos podem ou não sofrer mutação. Com isso, ocorre a evolução por várias gerações até ser encontrada a solução ótima para o problema. Pode-se fazer uma analogia com a teoria da seleção natural de Darwin, onde os mais bem adaptados ao meio têm maiores chances de sobrevivência do que os menos adaptados, deixando um número maior de descendentes. Os organismos mais bem adaptados são selecionados para aquele ambiente. Algoritmos genéticos são exemplos de técnicas desse paradigma.

3.2.3 Avaliação de Algoritmos

É indiscutível a importância e as potencialidades do aprendizado de Máquina. Porém, a descoberta de conhecimento é uma ciência experimental e não existe o melhor algoritmo para todos os problemas. Diversos métodos de AM devem ser experimentados para encontrar o melhor para certos dados [51, 49]. A avaliação dos algoritmos deve então ser um foco, assim é possível verificar a capacidade e a limitação de diferentes algoritmos. Porém, a avaliação não é uma tarefa simples. Um bom desempenho obtido no conjunto de treinamento não é, necessariamente, um indicativo de que este desempenho será mantido em um conjunto de teste independente.

Para problemas de classificação, como é o caso deste trabalho, uma forma usada para medir o desempenho de um modelo é a taxa de erro. O classificador prevê a classe de cada instância: se for correta é um sucesso, se não é um erro. A taxa de erro é a proporção de erros em um conjunto inteiro de instâncias. Convém lembrar que o interesse é no desempenho do classificador em novos dados e não em dados antigos, então a taxa de erro de dados antigos usados para treinar o classificador não é um bom indicador. Isso é muito importante, pois como o classificador aprendeu nesses dados, estimar o desempenho com os mesmos dados será uma análise otimista.

Para saber o desempenho de um classificador em novos dados, necessita-se avaliar sua taxa de erro em um conjunto de dados não utilizados na formação do classificador. Este conjunto de dados independentes é chamado de conjunto de teste. Os dados de teste não podem ser usados de nenhuma maneira na construção do classificador. Ambos os conjuntos devem ser amostras representativas do problema, sendo assim, a taxa de erro no conjunto de testes será um bom indicativo do desempenho futuro. Geralmente, quanto maior a amostra de treinamento, melhor será o classificador, embora o desempenho diminua quando um determinado volume de dados de treinamento for excedido [51]. E quanto maior o conjunto de teste, mais precisa será a estimação do erro. Então, existe um dilema: a partir de um mesmo conjunto de dados, para encontrar um bom modelo deve-se usar o máximo de dados para treinamento e para obter uma boa estimativa de erro deve-se usar o máximo possível para testar [49][51].

3.2.3.1 Métodos de amostragem

A metodologia de avaliação mais utilizada por pesquisadores de AM para comparar algoritmos baseia-se na ideia de amostragem (resampling) [49]. Considerando uma distribuição de exemplos em um domínio do mundo real como *D*, extrai-se um conjunto de exemplos *D'* dessa distribuição, exemplos esses similares à distribuição *D*. Para estimar o erro ou precisão de indutores treinados em *D'*, são extraídas amostras a partir de *D'*. O indutor é treinado e testado com exemplos de *D'*. O teste é realizado em amostras que não foram utilizadas para treinamento, como visto na seção anterior. Diante do exposto, foi visto que o processo de amostragem simula o que ocorre no mundo real, assim *D'* representa o mundo real. Outro fator existente é que, ao estimar uma medida verdadeira, é preciso que a amostra seja aleatória, portanto os exemplos não devem ser pré-selecionados. Existem vários métodos para estimar uma medida verdadeira e tratar do dilema mencionado na seção anterior [47, 49, 51]:

- **Resubstituição**: Nesse método o classificador é treinado e testado no mesmo conjunto de exemplos, ou seja, o conjunto de treinamento é o mesmo de teste. Por isso, o desempenho estimado desse método é otimista e não confiável. Esse *bias* é o motivo para a não utilização desse método e a utilização dos métodos de validação cruzada (*cross validation*), que seguem a regra de não haver exemplos em comum no conjunto de treinamento e teste.

- ***Holdout***: Os dados com exemplos rotulados são divididos em dois conjuntos disjuntos: uma porcentagem fixa p para treinamento e (1-p) para teste. Para evitar uma representação desigual entre os conjuntos de treinamento e teste deveria-se usar uma divisão 50:50, que não é o ideal. Geralmente é usado $p > 1/2$. Valores comuns são $p = 2/3$ e (1-p) $= 1/3$, porém não há uma fundamentação para esses valores.

- **Amostragem aleatória**: Nessa abordagem, as hipóteses são induzidas a partir de cada um dos conjuntos de treinamento. O erro final é a média de todos os erros das hipóteses induzidas e são calculados nos conjuntos de teste independentes e aleatórios. Essa abordagem produz melhor estimativa de erro que a *Holdout*.

- **Validação cruzada**: Nesse estimador, cada registro é usado o mesmo número de vezes para treinamento e exatamente uma vez para teste. No método de validação cruzada de k partes (k-*fold Cross-validation*), os exemplos são divididos de forma aleatória em k divisões mutuamente exclusiva, os *folds*, com tamanho de n/k instâncias. Os (k-1) *folds* são usados para treinamento e a hipótese induzida é testada no que sobrou. Esse processo se repete k vezes, onde cada vez é utilizado um *fold* diferente para testar, de modo que cada *fold* seja utilizado para teste exatamente uma vez. O erro é calculado pela média dos erros em cada um dos *folds*.

- **Validação cruzada estratificada**: Existe um problema na abordagem anterior se o *fold* usado para treinamento ou teste não for representativo. Essa validação é similar a anterior, a diferença é que cada *fold* possui a mesma proporção de distribuição das classes do conjunto original. Por exemplo, se o conjunto original possui duas classes de instâncias, 98% rotuladas como legítimas e 2% como maliciosas, essa proporção será mantida em cada *fold*. Como o objetivo do trabalho é obter o modelo com a maior capacidade de generalização possível, ou seja, maior capacidade de responder corretamente a dados que não foram manipulados no processo de treinamento, é necessária a utilização

de uma técnica que melhor avalie essa capacidade de generalização. Diante dessas características, essa abordagem foi a escolhida para avaliar os modelos neste trabalho.

- ***Leave-one-out***: Este método é um caso específico de validação cruzada. É um *k-fold cross-validation*, onde $k = n$. Sendo n o tamanho da amostra, a hipótese é induzida em *(n-1)* instâncias e testada na instância restante. Esse processo é repetido n vezes. O erro é a média dos erros nos n testes. É recomendado para amostras pequenas, porque é muito custoso computacionalmente.

- ***Bootstrap***: Neste estimador, repete-se muitas vezes a classificação estimando o erro ou o *bias*. Cada experimento é realizado em um novo conjunto de treinamento obtido por amostragem com reposição do conjunto de instâncias originais.

3.2.3.2 Classes Desbalanceadas

No mundo real é comum as distribuições de exemplos de um domínio possuírem as classes desequilibradas. Por exemplo, na detecção de fraudes de transações de cartões de crédito, o número de transações fraudulentas é bem menor que as legítimas; assim também ocorre no problema de detecção de *botnets*, o número de fluxos do tráfego de rede legítimos superam bastante os fluxos maliciosos. Então, por mais que seja infrequente, classificar corretamente a classe rara no exemplo desse trabalho possui um valor maior do que classificar corretamente a classe majoritária, pois o objetivo é detectar o tráfego de *botnets*. Porém, os algoritmos classificadores possuem problemas em conjuntos de dados com distribuição de classes desbalanceadas [47].

A métrica mais utilizada para avaliação é a acurácia, porém esta não é recomendada para bases desbalanceadas [52]. Por exemplo, se 1% dos fluxos do tráfego de rede forem maliciosos, então um modelo que preveja todos os fluxos como legítimos alcançaria uma acurácia de 99% apenas classificando toda a amostra como sendo da classe majoritária, o que condenaria totalmente o classificador pois não seria capaz de detectar qualquer tráfego malicioso. Ademais, medidas utilizadas para guiar um algoritmo de descoberta, por exemplo, ganho de informação para indução de árvores de decisão, podem necessitar ser modificadas para focar na classe rara [47]. Os modelos que trabalham com classes rara são altamente especializados.

Diante do fato da acurácia não ser apropriada para conjuntos de dados desequi-

librados, existem outras métricas adequadas para este cenário: matriz de confusão, precisão, *recall*, *f-measure* e área sob a curva ROC. As métricas servem para avaliar características dos modelos, já que cada um é único e a qualidade é relativa ao problema trabalhado. Essas métricas, juntamente com a acurácia serão utilizadas neste trabalho e suas descrições seguem abaixo [47] [51]. Para as equações abaixo uma classificação positiva é da classe *botnet*, enquanto a classificação negativa é da classe normal. TP (*true positive*) é o número de instâncias classificadas como positivas que são positivas, FP (*false positive*) é o número de instâncias classificadas como positivas que são negativas, TN (*true negative*) é o número de instâncias classificadas como negativas que são negativas e FN (*false negative*) é o número de instâncias classificadas como negativas que são positivas.

- **Matriz de confusão**: a classe rara muitas vezes é mais importante que a classe majoritária, como neste trabalho. Em uma classificação binária, geralmente a classe rara é considerada a classe positiva enquanto a majoritária é a negativa. A matriz de confusão de uma hipótese apresenta uma medição do modelo de classificação, mostrando o número de classificações corretas em comparação com as classificações preditas para cada classe em um conjunto de exemplos como mostrado na Tabela 3.1, o número de acertos para as classes é mostrado na diagonal principal (TP e TN), os demais elementos são erros na classificação.

Tabela 3.1: Matriz de confusão para um problema de classificação binária

		Classe Prevista	
		a	b
Classe Verdadeira	a	TP	FN
	b	FP	TN

- **Precisão** é a porcentagem de verdadeiros positivos (TP) sobre todas as instâncias classificadas como positivas, sejam verdadeiras (TP) ou falsas (FP):

$$\text{Precisão} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.1}$$

- **Acurácia** é a porcentagem de tudo o que foi classificado corretamente (TP e TN):

$$\text{Acurácia} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3.2}$$

- ***Recall*** *ou lembrança* é o número de verdadeiros positivos (TP) sobre o que é realmente positivo, ou seja, os verdadeiros positivos e os falsos negativos

(FN), é medida a fração de exemplos positivos previstos corretamente pelo classificador, equivalente a taxa de positivos verdadeiros (TPR):

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.3}$$

- ***F-measure*** é a média harmônica entre Precisão e *Recall*, construir um modelo que maximize a precisão e o *recall* é o grande desafio para todos os algoritmos de classificação:

$$\text{F-Measure} = \frac{2}{\frac{1}{\text{Prec}} + \frac{1}{\text{Recall}}} \tag{3.4}$$

- **Curva ROC** (*Receiver Operating Characteristic*) é a relação entre Recall ou taxa de verdadeiros positivos (TPR) e a taxa de falsos positivos (FPR), TPR fica no eixo y e a FPR no eixo x. Para reduzir essa curva a um valor escalar é calculada a área abaixo dela, um modelo perfeito terá área da curva ROC igual a 1, se o modelo realizar suposições aleatórias essa área será 0,5. Um modelo que seja melhor que outro, possuirá uma área maior.

3.2.4 Técnicas de Seleção de Atributos

As técnicas de seleção de atributos podem ser divididas em: *embedded*, *wrapper* e filtro [53]. As técnicas *embedded* ficam incorporadas no algoritmo indutor, diferente das técnicas de *wrapper* e filtro que são executadas na fase de pré-processamento. Técnicas de filtro avaliam o subconjunto de atributos por uma medida independente do classificador que será utilizado, essa técnica filtra os atributos antes da classificação através de informações dos dados, podem avaliar cada atributo e escolher os melhores como as técnicas de ranqueamento que foram utilizadas neste trabalho ou avaliar subconjuntos dos atributos, buscando o melhor subconjunto como a técnica *Correlation-based Feature Selection* que também foi utilizada neste trabalho. Já na abordagem *Wrapper* é utilizado um algoritmo de classificação para avaliação dos subconjuntos de atributos.

3.2.4.1 Ranqueamento de Atributos

Neste caso, os atributos são avaliados individualmente, assim, ordenam-se os atributos em um *ranking*. Para selecionar as características relevantes, pode-se usar um *threshold*, então os atributos cuja métrica de avaliação esteja acima do *threshold*, serão selecionados. Também pode ser selecionado um número fixo de atributos melhores avaliados. Para avaliar o grau de associação à classe de cada atributo

podem ser aplicadas diversas métricas.

- ***Gain Ratio***: essa métrica é baseada no ganho de informação, pois o ganho de informação possui um viés que pode sobrestimar atributos com muitos valores. Para evitar isso é calculada a razão entre o ganho de informação do atributo em relação à classe e a entropia [54]. Como mostra a Fórmula 3.5, onde C é a classe, A é o atributo e E a entropia.

$$GainRatio(C|A) = \frac{Ganho(C|A)}{E(A)} = \frac{E(C) - E(C|A)}{E(A)} \tag{3.5}$$

- **Incerteza Simétrica (*Symmetrical Uncertainty*)**: é outro modo de normalizar o ganho de informação, através da Fórmula 3.6:

$$IncSim(C|A) = 2 * \frac{Ganho(C|A)}{E(A) + E(C)} = 2 * \frac{E(C) - E(C|A)}{E(A) + E(C)} \tag{3.6}$$

3.2.4.2 *Correlation-based Feature Selection*

Essa técnica avalia o valor de um subconjunto de atributos considerando a capacidade preditiva individual de cada recurso juntamente com o grau de redundância entre eles. Daí, são escolhidos os subconjuntos de recursos que estejam altamente correlacionados com a classe, tendo baixa intercorrelação entre eles [55]. A Equação 3.7 demonstra o mérito de um subconjunto de atributos.

$$CFS(X_k, C) = \frac{k * r_{kc}}{\sqrt{k + (k-1)r_{kk}}} \tag{3.7}$$

Onde r é o coeficiente de correlação usado, varia de 0 (caso não haja correlação entre as variáveis x e y) a 1 (caso as variáveis sejam linearmente dependentes); $r_{kc} = \overline{r}(X_k, C)$ é o coeficiente de correlação médio entre os k atributos do subconjunto X_k e a classe C; $r_{kk} = \overline{r}(X_k, X_k)$ é o coeficiente de correlação médio entre os atributos [55].

3.2.4.3 *Wrapper*

O *Wrapper* [56] usa estimativas oriundas de outros algoritmos de aprendizado de máquina, aplicando o algoritmo escolhido em subconjuntos dos atributos até encontrar as melhores características, dada uma medida de desempenho. Semelhante as abordagens do tipo filtro, esse método também busca entre os diversos subconjuntos de atributos. A diferença é que na abordagem filtro, a avaliação se dá por

meio de uma métrica, enquanto na abordagem *Wrapper* é utilizado um algoritmo de classificação para cada subconjunto avaliado e a métrica utilizada geralmente é a precisão do algoritmo. Essa técnica possui custo computacional elevado, já que o algoritmo é utilizado múltiplas vezes em diferentes subconjuntos de atributos. Esse fato pode tornar inviável a utilização em bases de dados com muitos atributos. Existem variadas heurísticas que essa técnica pode trabalhar. A abordagem *Greedy* é bastante popular [11], nessa abordagem os atributos são adicionados ou removidos a cada iteração, dependendo se será aplicado o *Forward Selection* ou *Backward elimination*. Na primeira inicia-se com um atributo e então vão sendo adicionados a cada passo do processo, na segunda o subconjunto inicia-se com todos os atributos que vão sendo removidos a cada passo do processo.

3.2.5 Algoritmos de Aprendizado de Máquina

Esta subseção irá explicar os algoritmos utilizados neste trabalho para classificação. Do aprendizado supervisionado, os seguintes paradigmas são avaliados: simbólico com os algoritmos Árvore de Decisão J48 e *Random Forest* que também representa o *Ensemble Learning*; estatístico com Rede Bayesiana e *Support Vector Machines*; conexionista com Redes Neurais Artificiais Multicamadas do tipo Perceptron; baseado em exemplos com *K-Nearest Neighbor*. No aprendizado não supervisionado é avaliado o algoritmo de *Clustering*, *KMeans*. Ainda convém lembrar que o objetivo deste trabalho não é detalhar ou explorar as teorias por trás dos algoritmos, mas utilizá-los como ferramenta de classificação junto com técnicas de seleção de atributos.

3.2.5.1 Árvore de Decisão

Árvore de Decisão é um dos principais algoritmos de Aprendizado de Máquina, é composta por três elementos: nó raiz, arestas e nó folha. O nó raiz corresponde à decisão inicial e geralmente é o atributo mais discriminante entre as classes, fica no topo da árvore. As arestas são os possíveis valores, ramificações, também chamadas de ramos que ligam os nós até atingir o último nó. Nó folha é a resposta, ou seja, a classe na qual os objetos serão classificados. Para que ocorra a classificação, a árvore é percorrida da raiz às folhas, testando os valores dos atributos em cada nó até ser tomada a decisão final, quando o nó folha é alcançado. Existem diversos algoritmos que implementam Árvores de Decisão, como ID3, ID5, C4.5 e J48. Esses algoritmos utilizam a abordagem *top-down* para construir a árvore e são recursivos, só terminando quando constroem a menor árvore com maior acurácia. A partir de

um conjunto de instâncias de treinamento em um nó, é declarado o nó como folha ou busca-se uma maneira de dividir esse conjunto em novos subconjuntos . Essa divisão em subconjuntos é o que diferencia os algoritmos de indução de árvores de decisão . O TDIDT (*Top-Down Induction of Decision Tree*) [54] é a base para esses algoritmos de indução que utilizam a abordagem *top-down*. Este trabalho utiliza a implementação J48. Esse algoritmo, através de uma medida estatística chamada ganho de informação, escolhe o atributo que melhor divide o conjunto de treinamento, buscando subconjuntos mais puros, isto é, menos instâncias pertencendo a classes diferentes.

A entropia é uma medida usada em teoria da informação que representa a homogeneidade de um conjunto de exemplos. Chamando o subconjunto L de instâncias retiradas de um conjunto $P = \{e_1, e_2, ..., e_n\}$. Se todos os elementos de L forem iguais, a entropia é mínima mas se os elementos de P estiverem em igual quantidade em L, a entropia é máxima. A entropia é inversamente proporcional à quantidade de informação. A entropia é calculada através da Fórmula 3.8, onde *k(i/t)* são os registros pertencentes à classe i em um nó t, em um problema com duas classes, a distribuição delas em um nó pode ser escrita como *(p_0,p_1)*, onde p_1 = 1 - p_0.

$$\text{Entropia}(t) = -\sum_{i=0}^{c-1} p(i|t) \log_2 p(i|t) \tag{3.8}$$

O ganho de informação é a informação obtida conhecendo o valor do atributo, mede a redução da entropia causada pela divisão dos exemplos de um nó de acordo com os valores do atributo. O melhor atributo é aquele com maior ganho de informação e pode ser dado por [51] [57]:

Ganho = (entropia da distribuição antes da divisão) - (entropia da distribuição após a divisão) (3.9)

O processo de seleção de atributos e divisão das instâncias repete-se em cada nó descendente não folha, atributos já incorporados na árvore não participam do cálculo, aparecendo apenas uma vez na árvore. O processo continua até que todos os atributos sejam incluídos na árvore ou os exemplos de treinamentos pertencentes a esse nó possuírem o mesmo valor de atributo (entropia = 0). Se o processo de criação da árvore de decisão não decidir quando parar de criar sub-árvores, pode ser necessária a realização da poda. Árvores de decisão totalmente expandidas podem

possuir estruturas desnecessárias, é uma boa prática simplificá-las antes de serem implantadas em um sistema. Assim, buscam-se os pequenos ramos na borda da árvore que não contribuem e os removem. Esse algoritmo possui duas fases: treinamento, onde é construída a árvore com base nos dados rotulados; e classificação, onde os dados dos atributos selecionados são testados a partir da raiz até um dos nós folhas, onde a classe será atribuída [47] [51] [57]. Um exemplo de árvore de decisão pode ser visto na Figura 3.3, o problema é decidir se o tempo está favorável para a prática de esporte, o atributo umidade é selecionado por possuir maior ganho de informação.

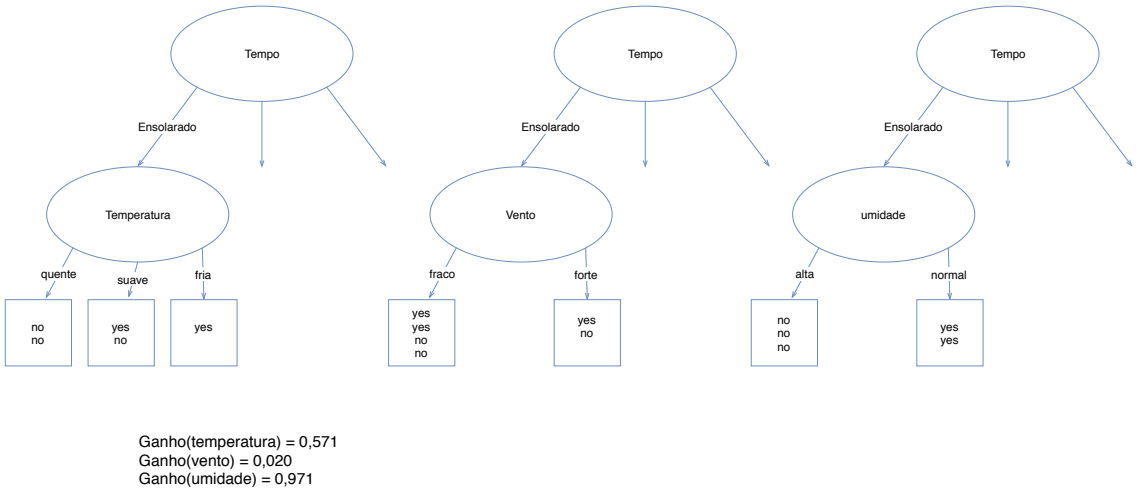

Figura 3.3: Exemplo de árvore de decisão, o atributo umidade é selecionado por possuir maior ganho de informação

3.2.5.2 *Random Forest*

Random Forest ou florestas aleatórias pertence aos métodos que combinam um conjunto de classificadores (*ensemble methods*). Foi projetada para combinar classificadores de árvores de decisão. Espera-se que métodos Ensemble apresentem melhor desempenho e confiabilidade que um único classificador [47] [58].

O objetivo desse algoritmo é criar um conjunto de árvores de decisão de modo que elas sejam muito pouco correlacionadas, quanto mais independentes forem, maior será a generalização e a robustez [58]. Para alcançar isso é utilizado o método *bagging*, esse método reduz a variância e evita o *overfitting*. O método *bagging* extrai subconjuntos de exemplos aleatoriamente a partir de conjuntos de dados de treinamento originais, esses subconjuntos são usados para cada árvore do modelo. Diferentes árvores são construídas através dos diferentes conjuntos de treinamento *bootstrap*. Cada árvore terá sua classificação, em forma de voto que indica sua

decisão. O número de votos de cada classe é contabilizado, a classificação final do *Random Forest* é a classe que receber mais votos, um exemplo pode ser visto na Figura 3.4. Esse algoritmo vem mostrando um bom desempenho, geralmente superior ao algoritmo de árvore de decisão [47] [58].

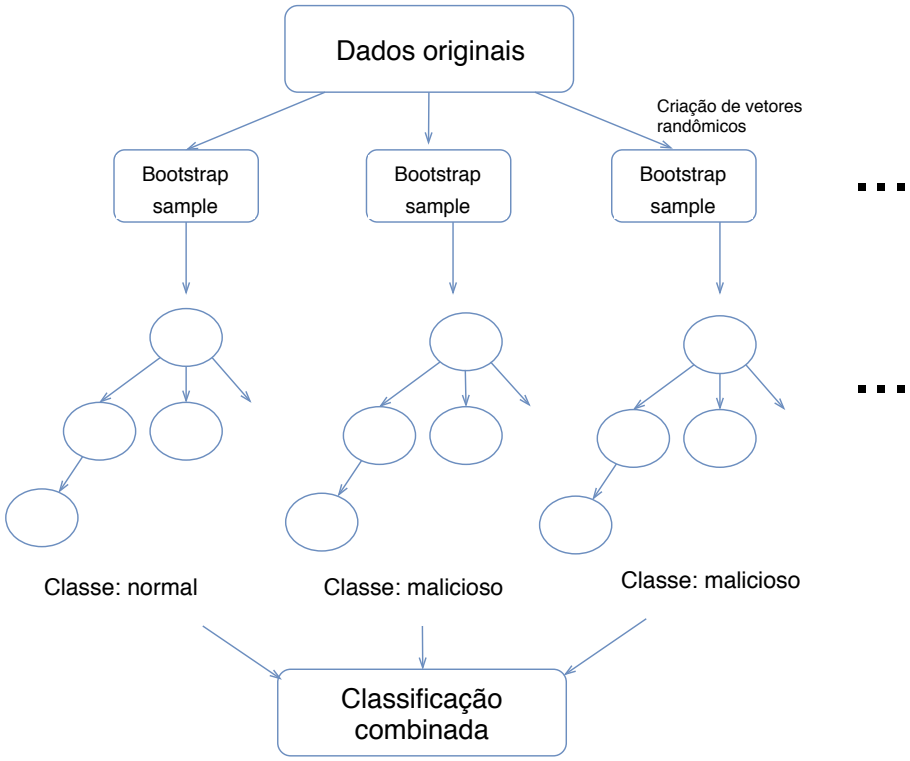

Figura 3.4: Exemplo de classificação com o algoritmo Random Forest

3.2.5.3 Rede Bayesiana

Rede Bayesiana é um modelo descrito por meio de um grafo acíclico direcionado onde são representadas as relações de causalidade entre as variáveis. Os nós representam as variáveis, os arcos as conexões entre elas e para representar as dependências são utilizadas probabilidades. Pode-se dizer que é uma representação enxuta de uma tabela de conjunção de probabilidades do universo do problema. Consiste de duas partes: o grafo acíclico direcionado e um conjunto de distribuições de probabilidades condicionais. Esses modelos são úteis em situações de incerteza, por determinar uma probabilidade de um acontecimento dependente das variáveis e das suas relações. A representação gráfica das relações de causalidade das variáveis facilita uma visualização simplificada do sistema. Essa rede probabilística pode ser definida como $\mathsf{RB} = (\mathsf{Pc}, \mathsf{G})$ onde G é um grafo acíclico direcionado e Pc as probabilidades condicionais de cada variável ou nó do grafo. As variáveis estão dentro de um intervalo finito, para cada uma delas existem probabilidades condicionais. A probabilidade condicional é a probabilidade de um evento A acontecer sabendo da

ocorrência de um evento B ($Pc(A|B)$). As Redes Bayesianas utilizam conhecimento incerto e incompleto através do Teorema de Bayes [59], teorema esse que descreve a probabilidade de um evento, baseado em um conhecimento a priori que pode estar relacionado ao evento, ou seja, relaciona as probabilidades dos dois acontecimentos A e B com as suas probabilidades condicionais mútuas, a Fórmula 3.10 descreve o teorema:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{3.10}$$

A distribuição conjunta de probabilidades de um conjunto de variáveis discretas $\{X_1, X_2, ..., X_n\}$ é o produtório das distribuições condicionais de todos os nós, dado os valores dos seus pais, dada pela regra da cadeia descrita a seguir:

$$P(X_1, X_2, ..., X_n) = \prod_{i=0}^{n} P(X_i|Pai) \tag{3.11}$$

Os parâmetros da rede bayesiana são descritos na Equação 3.12, Θ_i é uma tabela de probabilidades condicionais de X_i dado seus pais Pai. O conjunto de parâmetros é dado então por $\Theta_i = \{\Theta_1, \Theta_2, ..., \Theta_n\}$ que são todas as probabilidades condicionais com variáveis discretas da rede [60].

$$\Theta_i = P(X_i|Pai), i = 1, ..., n \tag{3.12}$$

Os nós das redes bayesianas são condicionalmente independentes dos não descendentes, dado o antecessor do nó. Os grafos não só permitem uma visualização das influências entre variáveis, mas também suportam o cálculo computacional das probabilidades condicionais necessárias para os resultados ou aprendizagem. A aprendizagem das redes pode ser sobre um modelo pré-existente elaborado por conhecimento prévio de especialistas, pelos dados ou ambos. Também há métodos de aprendizagem de parâmetros e estrutura da rede por meio de dados incompletos. Algumas vantagens dessa abordagem são a robustez que permite pequenas alterações sem prejudicar a performance, a possibilidade de classificar mesmo sem ter o valor de todos os atributos e a visão do relacionamento entre todos os atributos envolvidos no problema, assim como independências, dependências e quão forte são esses relacionamentos.

3.2.5.4 *Support Vector Machines*

O SVM é uma técnica que pode ser usada para classificação e regressão. São classificadores que conseguem diminuir os erros empíricos de classificação e aumentar a separação entre as classes espectrais. Foi proposta pelo pesquisador Vladmir Vapnik embasada na teoria de aprendizado estatístico. Pertencem ao grupo de algoritmos de AM considerados teoricamente superiores, pois utilizam algoritmos de otimização para localizar os limites ideais entre as classes tendo como meta diminuir a falha estrutural [61].

Essa técnica de classificação binária realiza a separação ótima entre duas classes por meio de um hiperplano de separação que melhor diferencia essas classes, esse hiperplano será incluído em um espaço de atributos n-dimensional entre as duas classes, de forma que cada classe fique em um lado do hiperplano. A distância entre cada classe e o hiperplano deve ser a maior possível. Esse hiperplano é definido através da identificação dos exemplos de treinamento mais significativos de cada classe, essas amostras recebem o nome de vetores de suporte, que são as mais próximas da superfície separadora entre as classes . O processo desse modelo consiste de três passos: localizar um hiperplano de separação ótima para separar os dados separáveis e não separáveis, conseguindo maximizar a margem dos pontos mais próximos no plano; tratar os dados não separáveis para que se consiga separá-los e usar as funções de decisão não linear para tratar os casos [61].

Para atingir o objetivo de separar os dados, esse algoritmo utiliza a função de *kernel*. Essa função permite a construção da superfície de decisão que é não-linear no espaço de entrada mas linear no espaço de atributos. As funções de *kernel* não lineares permitem a classificação de problemas não linearmente separáveis, projetando vetores de características de entrada em um espaço de alta dimensão, a medida que esse espaço aumenta, aumenta também a probabilidade desse problema tornar-se linearmente separável, isso pode ser visto na Figura 3.5(b) [1].

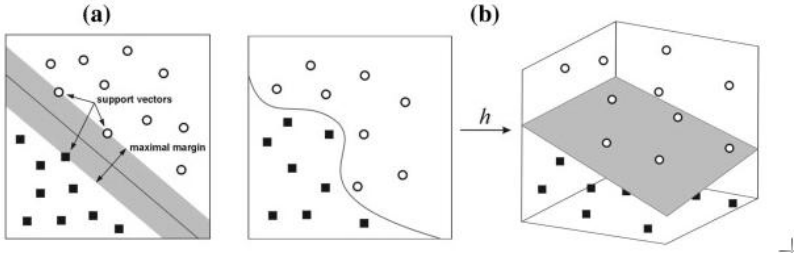

Figura 3.5: (a)SVM Linear (b)SVM não-linear [1]

Uma função *kernel* faz o mapeamento dos dados da posição original para o

produto interno das suas coordenadas como mostrado na Equação 3.13, em que ϕ deve pertencer a um domínio que seja possível o cálculo do produto interno.

$$K(c_i, c_j) = \langle \phi(c_i).\phi(c_j) \rangle \tag{3.13}$$

Dentre os vários tipos de *kernel*, o polinomial com grau p é descrito na Equação 3.14. Quando $p = 1$, o modelo é linear, à medida que p aumenta, geram-se hiperplanos curvos aumentando a eficiência da classificação.

$$K(x_i, x_j) = (x_i.x_j + 1)^p, \forall i, j = 1, ..., m \tag{3.14}$$

Outro *kernel* é o RBF (*radial basis function*) ou Gaussiano, sendo muito utilizado; inclusive é o padrão em diversas bibliotecas de linguagens de programação que utilizam o SVM. Essa função é definida na Equação 3.15, onde σ controla o raio da curva gaussiana. Devido a essas características esse *kernel* foi escolhido para ser utilizado neste trabalho, foi realizado um teste inicial onde esse *kernel* obteve melhor performance que o polinomial.

$$K(x_i, x_j) = exp - \frac{\left\| x_i - x_j \right\|^2}{2\sigma^2} \tag{3.15}$$

Esses modelos podem ser adaptados para classificar multiclasses, combinando vários classificadores binários SVM. Para classificar multiclasses existem dois métodos: *one-against-one* e o *one-against-all*. Métodos esses que não serão detalhados devido ao problema do trabalho ser uma classificação binária. O SVM, diferente de outros modelos de classificação, não busca maximizar o desempenho para o conjunto de treino e sim maximizar a generalização da classificação, evitando o *overfitting*. Algumas vantagens do SVM são a consolidada teoria matemática e estatística, robustez em dados de grandes dimensões e a ótima capacidade de generalização [62].

3.2.5.5 Rede Neural Artificial Perceptron Multicamadas

O algoritmo de Rede Neural assemelha-se ao funcionamento do cérebro humano, onde cada neurônio é responsável por parte do processamento e o resultado é passado ao próximo neurônio, até chegar a uma saída onde é obtido um grau de pertinência para cada classe - o maior grau será a classe escolhida. Esses neurônios calculam funções matemáticas, geralmente não-lineares e são ativados por um limiar [47] [63].

RNA Perceptron Multicamadas é composta de um conjunto de nós que formam a camada de entrada da rede, uma ou mais camadas ocultas e uma camada de saída. Essas redes possuem maior poder computacional do que as RNA sem camadas intermediárias. O número de nós da camada de entrada é definido pela dimensionalidade do espaço de observação, o número de nós da camada de saída é definido pela dimensionalidade da resposta e os pesos sinápticos que conectam os neurônios entre as camadas são definidos pelo algoritmo de treinamento. Então o projetista dessa rede deve definir: o número de camadas escondidas e o número de neurônios de cada uma dessas camadas. A complexidade da rede é determinada pelo quantidade de camadas escondidas e neurônios, não há regras para estabelecer esses números. Essas redes podem ser totalmente conectadas, onde cada neurônio é conectado a todos os neurônios da camada anterior e posterior ou parcialmente conectadas, em que não há todas as conexões entre neurônios. Um exemplo dessa rede pode ser vista na Figura 3.6.

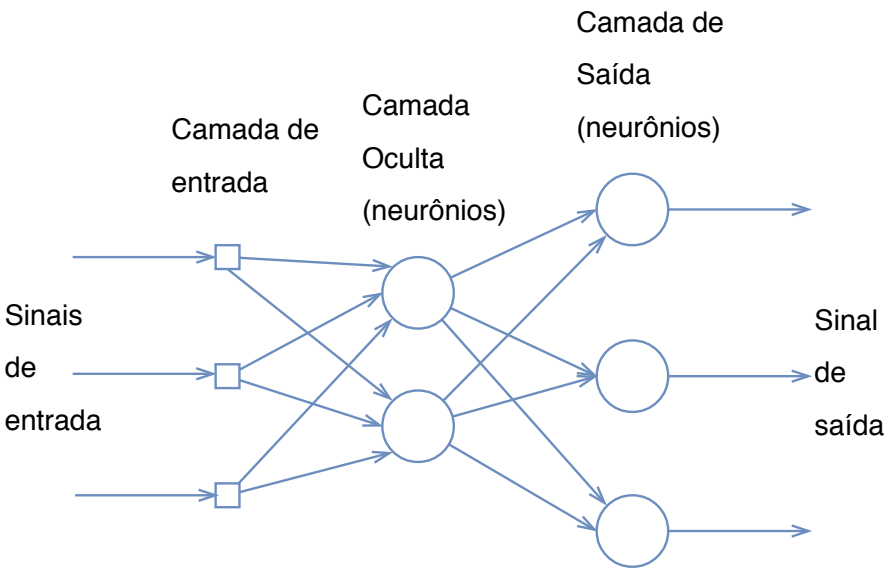

Figura 3.6: Exemplo de RNA

As camadas escondidas adicionam um conjunto adicional de pesos sinápticos e interações neurais e possibilitam a resolução de problemas não-lineares que é uma vantagem das RNA Perceptron Multicamadas. Para isso é necessário que a função de ativação dos neurônios das camadas ocultas seja não-linear, a função mais usada é a *sigmoide* [63], que será utilizada neste trabalho. Essas redes são *feedforward*, ou seja, as saídas do neurônio de qualquer camada se conectam apenas com neurônios da camada seguinte, assim as entradas se propagam através da rede em um sentido progressivo.

Os vetores de peso são calculados durante o treinamento, vetores esses que são as ligações entre os neurônios. Após obtido o espaço amostral de entradas e saídas desejadas, minimiza-se o erro da estimativa de cada parâmetro através de um algoritmo conhecido como *Back Propagation*, esse é o algoritmo de treinamento mais utilizado [63]. Ele é composto de duas fases. Na fase para frente, o vetor de entrada é inserido nos nós de entrada e se propaga através das camadas da rede até produzir a saída, nessa fase os pesos são estáticos. Na fase para trás, todos os pesos sinápticos são ajustados, a resposta dessa rede ao vetor de entrada é subtraída de um padrão de resposta desejado, produzindo um sinal de erro. Esse sinal de erro é propagado de volta, por isso recebe o nome de *Back Propagation*. Com isso, os pesos sinápticos são ajustados para aproximar a resposta do padrão desejado. A Equação 3.16 mostra o cálculo realizado para ajustar os pesos sinápticos, e é o erro, ou seja, a diferença entre o sinal desejado e a saída da rede.

$$e_j(n) = d_j(n) - y_j(n) \tag{3.16}$$

Para determinar a classe, o cálculo realizado por cada camada é determinado pelas equações:

$$u_{(i+1)} = w_{(i)} a_{(i)} \tag{3.17}$$

$$a_{(i+1)} = \sigma(u_{(i+1)}) \tag{3.18}$$

$$\sigma(u) = \frac{1}{1 + exp^{-u\beta}} \tag{3.19}$$

em 3.17 $a_{(i)}$ é a saída correspondente à camada i, $w_{(i)}$ é o vetor de pesos da camada i à $i + 1$; em 3.18 $a_{(i+1)}$ é a saída da camada seguinte; em 3.19 $\sigma(u)$ é a função unipolar *sigmoid* utilizada; e β é a constante de inclinação e não depende dos valores de entrada. Para valores de u maiores que zero, $\sigma(u)$ vale 1, caso contrário ela vale -1.

3.2.5.6 *K-Nearest Neighbor*

Conhecido como classificador do vizinho mais próximo. Esse algoritmo é do tipo *Instance-based learning* e pertence ao grupo de aprendizado preguiçoso, já que não é construído um modelo a partir do conjunto de treinamento. Esses algoritmos armazenam todas as instâncias de treinamento; ao classificar uma nova instância, recupera-se o conjunto de instâncias similares ou próximas à nova instância, conjunto esse que será usado para classificação. Para cada instância que será classificada , o *K-NN* calcula os k vizinhos ($k \geqslant 1$) mais próximos e os recupera, ela receberá a classe mais frequente entre esses k vizinhos. O algoritmo calcula a distância da instância que será classificada para cada uma do conjunto de treinamento, ordena esse conjunto do mais próximo ao mais distante. Dos elementos ordenados, os k primeiros são selecionados, que serve de parâmetro para regra de classificação. A Figura 3.7 ilustra o funcionamento genérico do algoritmo em que a classificação é dada pela maior frequência das classes dos k vizinhos mais próximos da instância a ser classificada. A interrogação é a instância a ser classificada, o triângulo e a esfera são classes de instâncias de treinamento já classificadas; se k for 1, a nova instância será classificada como esfera, já que é a classe do vizinho mais próximo; se k for 3, a instância será classificada como triângulo e se k for 7, o resultado da classificação será esfera.

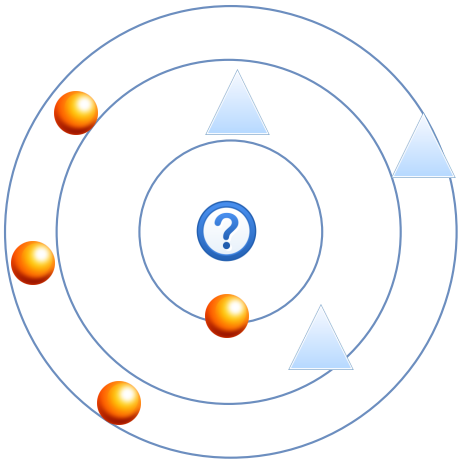

Figura 3.7: Exemplo de classificação com o algoritmo K-NN

No *K-NN* pode-se destacar dois pontos: a regra de classificação e a função utilizada para calcular a distância. A regra de classificação define a métrica para se tratar a relevância das k instâncias selecionadas. São duas regras: maioria por votação, onde cada instância tem igual peso e a classe vencedora será a que possuir maior número de representantes; e peso pela distância, onde cada instância tem um peso inversamente proporcional à distância [64]. A função de cálculo da distância é usada

para medir a diferença entre a instância a ser classificada e as instâncias do conjunto de treinamento, com o objetivo de selecionar as mais próximas. As distâncias utilizadas são a Euclidiana, a Manhattan e a Chebyschev[51]. Uma desvantagem desse algoritmo é o processo de classificação considerado custoso, já que durante a busca é necessária a realização de uma passagem completa por todas as instâncias de treinamento, esse fato é um problema diante de grande volume de dados.

3.2.5.7 *K-Means*

É o algoritmo de agrupamento mais utilizado [51]. Pertence ao aprendizado não-supervisionado, ou seja, trabalha com instâncias não rotuladas com a classe. Essas instâncias são dividas em grupos (*clusters*) bem definidos mais ou menos homogêneos de acordo com a similaridade entre as instâncias. A partição mais adequada é aquela que agrupa instâncias semelhantes no mesmo *cluster*, e separa as não semelhantes em diferentes *clusters*.

Esse algoritmo busca encontrar a melhor partição do conjunto de instâncias em k grupos, cada grupo está associado a um centroide. O parâmetro do algoritmo k determina o número de *clusters* que dividirá as instâncias. Parâmetro esse, que é definido pelo usuário; uma dificuldade é saber quantos *clusters* existem, uma forma de determinar k é tendo conhecimento sobre o problema. Após a definição do parâmetro k, deve-se selecionar as k instâncias, essa escolha pode ser por três formas diferentes: selecionando as k primeiras observações; selecionando aleatoriamente ou selecionando k observações de forma que seus valores sejam bastante diferentes, por exemplo, agrupando por peso, seriam escolhidos o mais pesado, um peso mediano e um peso leve. Essas instâncias serão os centroides iniciais. A partir disso, cada instância é associada ao centroide que esteja mais próxima, geralmente através da distância euclidiana, então os grupos vão se formando. O centroide de cada *cluster* é atualizado para a média das instâncias do seu grupo. O processo continua até nenhuma instância precisar mudar de *cluster*. Com todas as instâncias agrupadas, procura-se verificar se podem ser trocadas para um grupo melhor. Para isso, calcula-se o grau de homogeneidade interna dos grupos através da Soma do erro quadrado, medida usada para avaliar a qualidade dos grupos [47]. Esse algoritmo sempre termina, porém não obrigatoriamente encontra uma configuração ótima de *clusters*.

K-means exige que as variáveis sejam numéricas ou binárias. Para aplicações com dados categorizados, é necessário convertê-los em valores numéricos. Esse algoritmo tem como vantagens a sua simplicidade, eficiência e velocidade para convergir em

poucas iterações em um modelo estável [65]. Porém é bastante sensível aos centroides escolhidos na primeira iteração [66], para evitar esse problema foi proposta uma modificação no algoritmo. Assim, ele evolui para *K-means* ++, que é o algoritmo utilizado neste trabalho. A modificação foi para a melhora na seleção dos centroides iniciais, também conhecidos como sementes, seleção essa que é feita do seguinte modo: primeiro a semente inicial é escolhida aleatoriamente de todo o espaço, com uma distribuição de probabilidade uniforme; logo após, a segunda semente é selecionada com uma probabilidade proporcional ao quadrado da distância da primeira; Prossegue-se, em cada etapa, escolhendo a próxima semente com a probabilidade proporcional ao quadrado da distância da semente mais próxima que foi escolhida. *K-means* ++ adiciona as seguintes vantagens: diminuição do tempo de execução e aumento na performance com relação ao algoritmo original [51].

4. Metodologia de Estudo e Avaliação

Há um grande esforço vindo de pesquisadores que buscam detectar *botnets* nos últimos anos. No entanto, essas redes são de difícil detecção, devido a sua evolução constante e a utilização de criptografia na comunicação, dentre outros métodos de evasão. Na tentativa de resolver esse problema complexo, o aprendizado de máquina vem sendo bastante empregado, por ser considerado a estratégia mais efetiva para a detecção de *botnets* [11, 17, 18, 19].

Neste capítulo, são apresentados na Seção 4.1 os trabalhos relacionados à área de detecção de *botnets* através do aprendizado de máquina e da seleção das características de tráfego de rede que alimentam esses modelos. Na seção seguinte, Seção 4.2, a metodologia de estudo é demonstrada. Primeiramente, é explicado o pré-processamento, o cálculo de novos atributos e a seleção das características relevantes do tráfego de rede, através de técnicas de seleção de atributos. Após isso, são avaliados e comparados os modelos utilizados no trabalho, baseados em Aprendizado de Máquina supervisionado e não supervisionado. A Seção 4.3 descreve o ambiente experimental. Este ambiente contou com a utilização do pacote de software de código aberto *Weka*, para fins de execução dos algoritmos, e com a base de dados CTU-13 da *Czech Technical University* (CTU) [12].

4.1 Trabalhos Relacionados

O trabalho de [42] contribui no contexto de um HIDS, identificando diferentes canais de C&C sem verificar o *payload* do pacote, algo que abordagens anteriores não conseguiam. Nele são avaliados os algoritmos Árvore de Decisão J48 e *Random Forest* em um *dataset* próprio, contendo tráfego legítimo e diversas famílias de *bots*. Foram validadas as importantes hipóteses de que o tráfego de C&C da *botnet* pode ser diferenciado de outros, incluindo do tráfego legítimo, e as características de

diferentes estilos de C&C são semelhantes em diferentes famílias de *botnets*.

Em [67] é apresentada uma abordagem para extrair assinaturas de tráfego de C&C usando um *dataset* produzido com o *bot Anubis*, examinando o *payload* do pacote. Primeiro são extraídas as "sequências" mais frequentes do tráfego, depois é utilizada uma função de ranqueamento que dará uma pontuação maior às "sequências" que aparecem frequentemente em uma classe de conexões e raramente em outras. Isso é devido ao pressuposto de que conexões C&C de uma família de *malware* compartilham similaridades, enquanto conexões que não sejam de C&C são mais diversas. Com isso, os autores obtiveram uma acurácia superior a dos outros trabalhos relacionados.

Em [11] são comparados e avaliados atributos baseados em fluxo, empregados nos estudos de detecção de *botnets* existentes. A abordagem utilizada para essa comparação foi a *Wrapper*, com o algoritmo de classificação de árvore de decisão C4.5. Também foi criado um *dataset* contendo um conjunto diversificado de *botnets* e tráfego de *background*.

No trabalho de [68], empregam-se dois algoritmos de aprendizado de máquinas, o C4.5 e o SBB, usados para gerar modelos de detecção para o *bot* Zeus. O SBB foi verificado como sendo o que teve melhor desempenho.

Em [69] é proposto um sistema para detectar ataques *DDoS* a partir de fluxos IP, sistema esse que utiliza a arquitetura *lambda*, que é a junção do processamento em lotes com o processamento de fluxo de dados *(streaming processing)*. Para validar o sistema, foram utilizados os algoritmos VHT, *Bagging* e *AdaptiveBagging*, todos baseados em Árvore de Decisão, aplicados a dois *datasets*: um próprio com dados reais da rede do Observatório Nacional, misturado com tráfego de ataque simulado, e um com três cenários do *dataset* CTU-13. O sistema alcançou um excelente desempenho.

Em [70] é proposta uma metodologia para seleção de atributos objetivando detectar *botnets* na fase de comando e controle. Foi utilizado um algoritmo genético para selecionar o conjunto de atributos que fornece a maior taxa de detecção ao se usar o algoritmo C4.5 para a classificação dos dados. Experimentos foram realizados no sentido de extrair os melhores atributos para cada *botnet* analisada e para cada tipo de *botnet* em geral.

O trabalho em [67] realiza uma análise do *payload* do pacote, o que hoje é inviável pois a comunicação via C&C geralmente é criptografada. Esse trabalho é validado

em apenas um tipo de *bot*, limitando bastante o entendimento sobre a validade geral dessa metodologia.

Sobre a avaliação de modelos para se detectar *botnets*, a comparação entre os modelos estudados nos trabalhos não é possível de ser realizada, pois não há uma padronização de *datasets* sendo investigados, assim como dos conjuntos de métricas de avaliação usados. Outro fator existente é que, com exceção do trabalho em [70], os parâmetros utilizados nos algoritmos de aprendizado de máquina não são informados, impossibilitando a reprodução dos experimentos. Além disso, há uma escassez de trabalhos que comparem, de forma ampla, diversos algoritmos de classificação envolvendo os paradigmas do AM, para fins de detecção de eventos maliciosos.

Em cima desse problema, foi realizada a pesquisa em [2] e [3], que também utilizou o *dataset* CTU-13 para levantar atributos relevantes à detecção de *botnets* através das técnicas de seleção de atributos. Foram analisadas, nesses trabalhos, a eficiência dos algoritmos de aprendizado de máquina Árvore de Decisão J48, SVM, *Naive Bayes* e k-NN para a detecção do tráfego de *botnets*. Entretanto, ainda faltam paradigmas de AM supervisionado para serem comparados, o que a própria pesquisa aponta como trabalhos futuros. Dentre eles, podemos citar o paradigma conexionista, através do algoritmo RNA. Além disso, o trabalho não abordou o AM não supervisionado.

Buscando preencher essas lacunas, foi desenvolvido o presente trabalho, em que diversos paradigmas do AM supervisionado e não supervisionado são comparados. Além disso, é realizada uma seleção de características, comparando novos atributos que não foram calculados em [2] e [3]. Assim, pretende-se descobrir se existe um algoritmo que satisfaz os diversos cenários de *botnets* estudados ou se uma combinação dessas técnicas deve ser usada para uma classificação mais eficaz, determinando a combinação de características e técnicas de AM ideais para o problema da detecção de *botnets*. A Tabela 4.1 mostra uma comparação entre o presente trabalho e os trabalhos em [2] e [3].

4.2 Metodologia

A metodologia usada neste trabalho está ilustrada na Figura 4.1. O trabalho pode ser dividido em duas partes que serão descritas nas próximas subseções: pré-processamento e extração de características, e avaliação dos algoritmos de AM.

Tabela 4.1: Comparação entre este trabalho e [2] [3]

Trabalho	Dataset	Atributos Avaliados	Técnicas de Seleção de Atributos	AM Supervisionado	AM Não Supervisionado
[2] [3]	CTU-13	Além dos atributos *NetFlow* do dataset: -Média tamanho do pacote por fluxo -Média pacote por segundo por fluxo -Média bits por segundo por fluxo	-*Wrapper* + J48 (*Greedy, Forward Selection*) -Ranqueamento	-J48 -SVM -*Naive Bayes* -K-NN	-
Este	CTU-13	Além dos atributos *NetFlow* do dataset: -Média tamanho do pacote por fluxo -Média pacote por segundo por fluxo -Média bits por segundo por fluxo **-Razão entre número de bits transmitidos pela origem e destino por fluxo** **-Desvio padrão do tamanho do pacote em uma janela de tempo de 180s** **-Desvio padrão da duração do fluxo em uma janela de tempo de 180s**	-*Wrapper* + J48 (*Greedy, Forward Selection*) **-*Wrapper* + J48 (GA, *Forward Selection*)**} **-CFS** -Ranqueamento **(*Gain Ratio* e Incerteza Simétrica)**	-J48 **-*Random Forest*** -SVM **-Rede Bayesiana** -K-NN **-RNA *Perceptron* Multicamadas**	***KMeans***

4.2.1 Pré-processamento e Extração de Características

Num processo de aprendizado de máquina, o número de características que devem ser processadas para fins de classificação dos dados pode aumentar significativamente o tempo de processamento sem que necessariamente melhore a capacidade de detecção do modelo. Atributos redundantes e desnecessários podem causar ruídos no resultado e isso é algo que pode impactar severamente um sistema de detecção de ataques. Por isso, é importante usar técnicas que reduzam o número de características que serão processadas pelos modelos sem degradar a capacidade de classificação.

Para esta fase, buscou-se pesquisas sobre seleção de características para detecção de *botnets*. Em [11], apontou-se as seguintes características: duração do fluxo, média de bits/s, média do tamanho do pacote e razão do número de pacotes de entrada com os de saída. Já em [2] e [3], indicou-se como características: duração do fluxo, estado, total de *bytes* transmitidos, média do tamanho do pacote e média de bits/s. Em [70], selecionou-se: número de pacotes da origem ao destino, média do tamanho do pacote, tamanho do *payload*, total de *bytes* transmitidos, tamanho do primeiro pacote da conexão, número de pacotes nulos e desvio padrão do tamanho do *payload*.

Após essa busca na literatura, foi realizado um pré-processamento no *dataset* para eliminar dados desnecessários, calcular novos atributos e normalizar quando necessário. Nesta etapa, fluxos com duração igual a zero foram eliminados. Também foram eliminados os atributos afetados pelas técnicas de evasão das *botnets*, que trocam essas informações periodicamente: endereço IP de origem, endereço IP de destino, porta utilizada pelo protocolo na origem, porta utilizada pelo protocolo no destino e campo indicador de protocolo da camada de transporte. A partir daí, novos atributos foram calculados em função das características recomendadas pelos

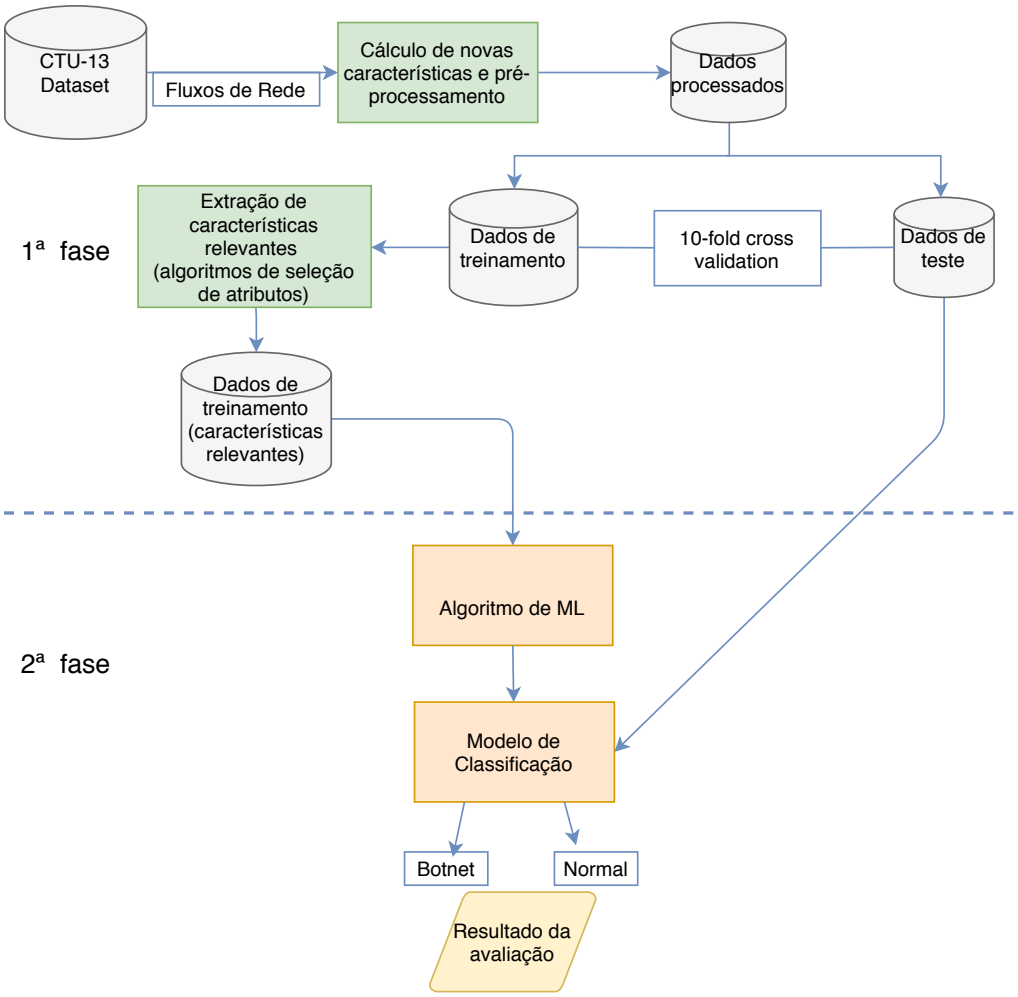

Figura 4.1: Metodologia

trabalhos em [2, 3, 11, 70]: média do tamanho do pacote por fluxo, média de pacotes por segundo por fluxo, média de bits por segundo por fluxo, razão entre número de bits transmitidos pela origem e destino por fluxo. Também foram calculados o desvio padrão do tamanho do pacote em uma janela de tempo de 180s e o desvio padrão da duração do fluxo, também em uma janela de tempo de 180s. A intenção no cálculo desses desvios é verificar características que só são visíveis quando um comportamento de grupo é analisado, assim é calculada a extensão do desvio de um grupo com o todo. Um alto desvio padrão pode indicar atividade normal, e não uma atividade sincronizada de uma *botnet*, devido ao afastamento da média, enquanto um baixo desvio indicaria o oposto. A premissa principal é que o tráfego de *botnet* (principalmente ações predefinidas) é mais uniforme que o tráfego gerado por usuários legítimos que exibem um comportamento muito diversificado [11]. A janela de tempo de 180s foi calculada pelo atributo *StartTime* (tempo de início do fluxo). Nos casos em que na janela de tempo havia a ocorrência de apenas um fluxo, os desvios-padrão do tamanho do pacote e da duração do fluxo foram considerados

iguais a zero. O tempo de 180s foi calculado no trabalho [71] como sendo a janela de tempo que apresentou maior precisão, mostrando que as *botnets* seguem um padrão de comportamento nessa janela, o que pode ser entendido como o tempo de ciclo de vida. O trabalho [71] analisou janelas de tempo entre 60 e 300s, em múltiplos de 60. Verificou-se que, em outras janelas de tempo, a precisão de detecção decresce. Além disso, também foi realizada a padronização do atributo *label*, reduzindo para apenas duas possibilidades, *Normal* e *Botnet*, onde os fluxos rotulados como *Background* foram associados ao rótulo *Normal*.

Após essa etapa, mesmo considerando a redução de características já conseguidas, considerou-se necessário eliminar atributos redundantes, como atributos altamente correlacionados e atributos irrelevantes para a classificação. Para isso, foram utilizados algoritmos de seleção de atributos nos dados de treinamento, com o propósito de chegar às melhores características. Foram empregados o ranqueamento de atributos com as métricas *gain ratio* e incerteza simétrica, a técnica *correlation-based feature selection* e a abordagem *wrapper* com o método de busca *greedy*, aplicando *forward selection*. No *wrapper*, foi utilizado o algoritmo Árvore de Decisão J48 para verificação dos efeitos de classificação. Ainda na abordagem *wrapper*, também utilizou-se o método de busca adotando o algoritmo genético descrito em [72], utilizado no trabalho [70], onde os parâmetros com melhores resultados foram a taxa de *crossover* igual a 0,5 e a taxa de mutação igual a 0,2. A verificação dos efeitos de classificação também se baseou no algoritmo Árvore de Decisão J48.

O algoritmo Árvore de Decisão J48 foi escolhido para fornecer as estimativas de precisão da abordagem *wrapper* por ser amplamente utilizado e apresentar um bom desempenho nos estudos de detecção de *botnet* [2]. Nesse trabalho, ele é utilizado fazendo-se poda com redução de erro. Essa técnica permite melhorar a precisão do algoritmo e reduzir a árvore de decisão gerada. No final dessa fase, busca-se selecionar o menor subconjunto de atributos que levem à maior precisão do classificador. Com isso, consegue-se obter redução do conjunto de dados, melhora no desempenho da classificação e simplificação dos modelos que serão avaliados. Além disso, para os algoritmos Árvore de Decisão J48 e *Random Forest* os atributos numéricos foram discretizados nos dados de treinamento, através do ganho de informação. Essa discretização já é incorporada nesses algoritmos, como mostrou a subseção 3.2.5.1. Sabendo que a normalização melhora o desempenho dos algoritmos que trabalham ponderando as entradas e também os baseados em distância, já que, por exemplo, variáveis com valores grandes podem influenciar no resultado desses modelos [47], uma característica do tráfego de rede que possua uma variação maior terá mais im-

portância do que outra com menor variação. Diante disso, para os algoritmos SVM, RNA, K-NN e *KMeans*, os atributos numéricos foram normalizados nos dados de treinamento, redimensionando os atributos para a escala [0:1] através da Equação 4.1:

$$\mathrm{ValNormalizado[i]} = (\mathrm{Val[i]} - \mathrm{MinVal[i]})/(\mathrm{MaxVal[i]} - \mathrm{MinVal[i]}) \quad (4.1)$$

4.2.2 Avaliação dos Algoritmos de AM

Após ter as características extraídas, é o momento de utilizá-las nos algoritmos de aprendizado de máquina para que sejam analisados os desempenhos desses na detecção de *botnets*. Nessa etapa foi utilizado o software *Weka* [73] para executar os algoritmos. Os modelos gerados são comparados utilizando as métricas para classes desbalanceadas mencionadas na Seção 3.2.3.2 (precisão, *recall* e *f-measure*), lembrando que a acurácia não é apropriada para esse tipo de base de dados. Além dessas métricas, outros aspectos são levados em consideração, como o tempo para treinar o modelo, e também a escalabilidade, através da construção de modelos a partir de grandes quantidades de dados.

Os algoritmos de AM usados neste trabalho fazem uso de classificação supervisionada e não supervisionada. Através do AM não supervisionado, avalia-se a possibilidade de detectar *bots* através da atividade do grupo, assumindo que ocorram atividades coordenadas dos *bots* na mesma *botnet*, com padrões de tráfego semelhantes, identificando os *bots* na rede com base em sua ação coletiva em vez de em suas operações individuais. Já no AM supervisionado, ao contrário da detecção do grupo, os *bots* são classificados com base em suas ações e características individuais, independentemente da atividade do grupo em que possam ser parte. Os paradigmas do aprendizado supervisionado avaliados foram: simbólico (Árvore de Decisão J48 e *Random Forest*), estatístico (Rede Bayesiana e *Support Vector Machines*), conexionista (Redes Neurais Artificiais *Multilayer* do tipo Perceptron) e baseado em exemplos (*K-Nearest Neighbor*). No aprendizado não supervisionado, avaliou-se o algoritmo de *Clustering KMeans*. Além disso, o algoritmo *Random Forest* também representa o *Ensemble Learning* do tipo randomização. Esses modelos foram usados para a detecção de fluxos de pacotes decorrentes de atividades de *botnets* registradas em treze cenários de rede da *Czech Technical University* [12]. Como os cenários dessa base são desbalanceados, ou seja, existem muito mais casos de uma classe que a outra, e neste caso a classe *Botnet* está sempre em menor número, os modelos devem ser muito bem avaliados. Isso porque modelos gerados com classes

raras podem apresentar baixa acurácia para essas classes, pois tendem a destacar a classe predominante [74]. Diante desse viés, a melhor abordagem de avaliação para esses casos é a validação cruzada. Neste trabalho é utilizada a validação cruzada estratificada em dez partes. Com isso é garantido uma variância menor nos resultados da avaliação e garante-se que cada parte da divisão possua cada classe representada adequadamente nos conjuntos de treinamento e teste. Foi selecionada a divisão em dez partes devido ao fato de que testes extensivos em vários conjuntos de dados diferentes, com diferentes técnicas de AM, mostraram que esse número de *folds* apresenta a melhor estimativa de erro. Há também evidências teóricas que corroboram isso, porém esses argumentos não são conclusivos, e há ainda a discussão sobre qual o melhor esquema para avaliação. No entanto, a validação cruzada em dez partes (*tenfold cross-validation*) tornou-se o método padrão em termos práticos. Além disso, testes também mostraram que a estratificação melhora os resultados [51].

4.3 Ambiente Experimental

Nesta seção será apresentado o ambiente onde a metodologia de estudo foi executada, que basicamente se consistiu das bases de dados CTU-13 e da ferramenta Weka, usada para a implementação dos modelos de aprendizado de máquina. Este ambiente contou com um computador com processador Intel(R) Core(TM) i7-5500U, CPU @ 2.40GHz com 4 núcleos e 16GB de memória. O sistema operacional utilizado foi o Linux Ubuntu 17.10.

4.3.1 *Weka*

Weka (Waikato Environment for Knowledge Analysis) é um pacote de software de código aberto que começou a ser escrito em 1993 usando a linguagem Java na Universidade de Waikato, Nova Zelândia. Como ele é licenciado pela *General Public License*, é possível a alteração de seu código fonte. Esse pacote é composto pelos mais diversos algoritmos de aprendizado de máquina de diferentes paradigmas na sub-área da inteligência artificial para tarefas de mineração de dados. Podem ser executadas tarefas de pré-processamento, classificação, regressão, agrupamento, regras de associação e visualização. Os algoritmos podem ser aplicados diretamente a um conjunto de dados ou chamados por um código em Java. As bases de dados devem estar no formato *arff* (*Attribute-Relation File Format*). Pela sua programabilidade, o *Weka* também é bem apropriado para o desenvolvimento de novos

esquemas de aprendizado de máquina [75] [76]. Além disso, costuma apresentar desempenho suficiente para ser aplicado em cenários de *Big Data*. Devido a essas características, esse pacote de software foi escolhido para realizar os experimentos desse trabalho.

4.3.2 Base de dados

Encontrar, ou gerar, bases de dados (*datasets*) realistas que reflitam o problema em questão não é tarefa trivial. Essa base deve conter, além do tráfego não malicioso, *botnets* realizando atividades maliciosas e também se comunicando através do canal de comando e controle *C&C*. Esta base deve ainda ser capaz de refletir as proporções entre tráfego legítimo e tráfego malicioso que representam situações reais de uma *botnet* em ação, em geral caracterizadas por um desbalanceamento entre estes tráfegos. Muitas dessas bases se originam de *honeypots*, que são máquinas aparelhadas para se comportarem como potenciais vítimas e, assim, dedicadas a coletar tráfego malicioso. Nesses cenários, o tráfego legítimo (não malicioso) fica em menor quantidade e acaba por não reproduzir uma situação real.

Neste trabalho são utilizados os treze cenários da base de dados CTU-13 da *Czech Technical University* (CTU) [12]. Trata-se de uma base desbalanceada, conforme a realidade, composta de tráfego de *botnets* realizando diversas atividades maliciosas e também tráfego legítimo. A base é dividida em treze diferentes cenários, cada um com características diferentes. Essa base foi criada para atingir os seguintes objetivos: possuir ataques reais de *botnets* e não oriundo de simulações, possuir tráfego desconhecido de uma rede grande, ter rótulos para treinamento e avaliação dos métodos, incluir diferentes tipos de *botnets* e ter vários *bots* infectados ao mesmo tempo para capturar padrões de sincronização. A topologia usada para criar o conjunto de dados consistia em um conjunto de computadores virtualizados executando sistema operacional *Windows* em um *host Linux Debian*. Cada máquina virtual estava conectada à rede da Universidade em modo *bridge*. O tráfego foi capturado tanto no *host Linux* quanto em um dos roteadores da Universidade. O tráfego do *host Linux* foi exclusivamente composto de tráfego de *botnet* e foi usado para os *labels*. O tráfego do roteador da Universidade foi usado para criar o *dataset* final, a ferramenta usada para capturar o tráfego foi *tcpdump*.

Após a captura, as bases foram convertidas para fluxos do tipo *NetFlow* [77], que é um recurso frequentemente encontrado em roteadores e *switches* para sumarizar as informações do tráfego que passa por uma rede. Isso é feito a partir da descrição dos

fluxos de pacotes de dados existentes na rede, onde um fluxo é definido como uma sequência de pacotes que possuem os mesmos valores para o endereço IP de origem, o endereço IP de destino, o número identificador do protocolo de aplicação de origem (número da porta de origem), o número da porta de destino e o número identificador do protocolo usado no transporte das informações (por exemplo, identificando ser TCP ou UDP). Geralmente considera-se que um fluxo se encerrou quando se detecta o fechamento de uma sessão TCP, para os fluxos que fazem uso deste protocolo, ou quando há ausência de pacotes neste fluxo por mais de 30 segundos. Para cada fluxo IP caracterizado por sua respectiva tupla, são registradas diversas informações relevantes, como o número de pacotes contido no fluxo, o total de *bytes* enviados, o tempo de início e a duração do fluxo, dentre outras.

Dentro de cada cenário do *dataset* CTU-13, foram atribuídos um dos seguintes rótulos (*labels*) para cada fluxo: rótulo *Normal*, atribuído ao tráfego sabidamente não malicioso, cujos fluxos se originaram de equipamentos para os quais havia uma supervisão cuidadosa; o rótulo *Botnet*, sinalizando os fluxos oriundos dos *IPs* sabidamente infectados; e o rótulo *Background*, usado para todo o tráfego restante. Importante ressaltar que, em todos os cenários, a classe associada ao rótulo *Botnet* possui uma quantidade de fluxos muito menor que a das demais classes, o que dificulta a detecção. Desta forma, cada fluxo contido nestas bases possui os seguintes atributos: *StartTime, Dur, Protocol, SrcAddr, Sport, Dir, DstAddr, Dport, State, sTos, dTos, TotPkts, TotBytes, SrcBytes e Label*.

Na composição dessas bases de dados foram utilizados os seguintes *bots*: *Neris, Rbot, Virut, NSIS, Menti, Sogou* e *Murlo*. A Tabela 4.2 mostra os protocolos utilizados e as atividades maliciosas realizadas por esses *bots*, onde as siglas significam: Cen = cenário, CF = Click Fraud, PS = Port Scan, FF = Fast Flux e CC = compilado e controlado pelos autores. A distribuição dos rótulos é mostrada na Tabela 4.4 e as estatísticas de tráfego de cada cenário com a duração da captura, quantidade de pacotes, fluxos e *botnets* utilizadas são mostradas na Tabela 4.3.

Tabela 4.2: Características dos cenários

Cen	IRC	SPAM	CF	PS	DDoS	FF	P2P	CC	HTTP
1	X	X	X						
2	X	X	X						
3	X			X				X	
4	X				X			X	
5		X		X					X
6				X					
7									X
8				X					
9	X	X	X	X					
10	X				X			X	
11	X				X			X	
12							X		
13		X		X					X

Tabela 4.3: Características dos cenários de captura de pacotes:

Cen	Duração(hs)	Qtd Pacotes	Tamanho	Botnet	Qtd Botnets
1	6,15	71.971.482	52 GB	*Neris*	1
2	4,21	71.851.300	60 GB	*Neris*	1
3	66,85	167.730.395	121 GB	*Rbot*	1
4	4,21	62.089.135	53 GB	*Rbot*	1
5	11,63	4.481.167	37,6 GB	*Virut*	1
6	2,18	38.764.357	30 GB	*Menti*	1
7	0,38	7.467.139	5,8 GB	*Sogou*	1
8	19,5	155.207.799	123 GB	*Murlo*	1
9	5,18	115.415.321	94 GB	*Neris*	10
10	4,75	90.389.782	73 GB	*Rbot*	10
11	0,26	6.337.202	5,2 GB	*Rbot*	3
12	1,21	13.212.268	8,3 GB	*NSIS.ay*	3
13	16,36	50.888.256	34 GB	*Virut*	1

Tabela 4.4: Distribuição da quantidade de fluxos para os cenários

Cen	Total	Botnet	Normal	C&C	Background
1	2.824.636	39.933 (1,41%)	30.387 (1,07%)	1.026 (0,03%)	2.753.290 (97,47%)
2	1.808.122	18.839 (1,04%)	9.120 (0,5%)	2.102 (0,11%)	1.778.061 (98,33%)
3	4.710.638	26.759 (0,56%)	116.887 (2,48%)	63 (0,001%)	4.566.929 (96,94%)
4	1.121.076	1.719 (0,15%)	25.268 (2,25%)	49 (0,004%)	1.094.040 (97,58%)
5	129.832	695 (0,53%)	4.679 (3,6%)	206 (1,15%)	124.252 (95,7%)
6	558.919	4.431 (0,79%)	7.494 (1,34%)	199 (0,03%)	546.795 (97,83%)
7	114.077	37 (0,03%)	1.677 (1,47%)	26 (0,02%)	112.337 (98,47%)
8	2.954.230	5.052 (0,17%)	72.822 (2,46%)	1.074 (2,4%)	2.875.282 (97,32%)
9	2.753.884	179.880 (6,5%)	43.340 (1,57%)	5.099 (0,18%)	2.525.565 (91,7%)
10	1.309.791	106.315 (8,11%)	15.847 (1,2%)	37 (0,002%)	1.187.592 (90,67%)
11	107.251	8.161 (7,6%)	2.718 (2,53%)	3 (0,002%)	96.369 (89,85%)
12	325.471	2.143 (0,65%)	7.628 (2,34%)	25 (0,007%)	315.675 (96,99%)
13	1.925.149	38.791 (2,01%)	31.939 (1,65%)	1.202 (0,06%)	1.853.217 (96,26%)

5. Resultados

Este capítulo apresenta os resultados obtidos dos experimentos relativos à seleção de atributos (ou características) para as avaliações dos modelos estudados, aplicados tanto para detecção reativa quanto para detecção preventiva.

5.1 Seleção de Características

Esta fase, descrita na Seção 4.2.1, busca reduzir o número de atributos, selecionando as características do tráfego de rede que se mostram mais relevantes para a detecção das botnets nos diferentes cenários. O resultado das técnicas utilizadas estão na Tabela 5.1. No caso das técnicas de ranqueamento, foram selecionadas as cinco características melhores colocadas, porém, de modo geral, para cada cenário um subconjunto de características relevantes foi selecionado com base nas características que despontaram com maior frequência nas técnicas utilizadas.

Analisando os subconjuntos selecionados dos resultados da aplicação dessas técnicas nos treze cenários, as características que mais se destacaram foram *RazaoBytesOrigDest*, *MedBitsSecond*, *Dur*, *SrcBytes*, *MedPktSecond* e *MedPktSize*, todas elas aparecendo na maioria dos cenários avaliados. As características *DpDur* e *TotBytes* aparecem em quatro dos cenários, enquanto que a característica *DpPkt* aparece como relevante em apenas um cenário.

Comparando com os trabalhos [3], [11] e [70], ratificam-se as seguintes características: razão entre o número de pacotes de entrada com os de saída do fluxo (*RazaoBytesOrigDest*) [11], média de bits/s (*MedBitsSecond*) [3, 11], duração do fluxo (*Dur*) [3, 11] e média do tamanho do pacote de um fluxo (*MedPktSize*) [3, 11, 70]. A quantidade de *bytes* enviadas pela origem no fluxo (*SrcBytes*) e a média do tamanho do pacote (*MedPktSize*) aparecem, no presente trabalho, como novas características

relevantes para a detecção de tráfegos de *botnets*.

As características que foram selecionadas como relevantes em algum cenário, mas que não aparecem na maioria dos cenários, devem ser consideradas para esses cenários específicos. São elas: o total de *bytes* transmitidos no fluxo (*TotBytes*), selecionado nos trabalhos [5] e [70]; o desvio padrão do tamanho do pacote em uma janela de tempo de 180s (*DPPkt*) e o desvio padrão da duração do fluxo em uma janela de tempo de 180s (*DpDur*), selecionadas apenas neste trabalho. Os atributos presentes na coluna "Características Selecionadas" da Tabela 5.1, para cada cenário, serão utilizados na próxima fase deste trabalho, onde os algoritmos de AM serão avaliados por cenário.

5.2 Avaliação dos Modelos de AM

Nesta fase, os modelos gerados pelos algoritmos do AM supervisionado e não supervisionado dos diferentes paradigmas são avaliados e comparados. Para os algoritmos Árvore de Decisão J48 e *Random Forest*, os atributos numéricos foram discretizados através do ganho de informação. Já nos algoritmos SVM, RNA, K-NN e *KMeans*, os atributos numéricos foram normalizados nos dados de treinamento, redimensionando-os para a escala [0:1]. O algoritmo *KMeans* também foi utilizado para classificação, como é um problema de classificação binária, foram considerados dois *clusters*. Todos os cenários da base de dados são desbalanceados, sempre com a maior classe sendo a de tráfego normal, portando o maior *cluster* encontrado foi considerado sendo o de tráfego normal e o menor de tráfego de *botnet*.

Como mencionado anteriormente, foi utilizada a validação cruzada estratificada em dez partes. Os algoritmos foram treinados usando o conjunto de dados de treinamento e um modelo de classificação então foi gerado e testado no conjunto de dados de teste independente, conforme ilustrado na Figura 4.1. Convém lembrar que o objetivo do trabalho é avaliar os modelos de AM para serem usados em dados novos, por isso avaliar os modelos nos mesmos dados usados para treiná-los não é um bom indicador. Isso poderia induzir em uma análise otimista, conforme explicado na Seção 3.2.3. Logo, o conjunto de testes deve ser independente. Para avaliar os modelos gerados, foram utilizadas as seguintes métricas indicadas para bases de dados desbalanceadas: precisão, *recall*, *f-measure* e o tempo de processamento descritas na Seção 3.2.3.2.

Primeiro, cada algoritmo foi avaliado com os parâmetros padrão da sua classe,

Tabela 5.1: Resultado da Seleção de Características

Cen	Ranqueamento		CFS	Wrapper + J48	Wrapper + J48	Características Selecionadas
	Gain Ratio	Incerteza Simétrica	Greedy, Forward Selection	Greedy, Forward Selection	GA, Forward Selection	
1	RazaoBytesOrigDest, MedBitsSecond, Dur, MedPktSecond, SrcBytes	RazaoBytesOrigDest, MedBitsSecond, Dur, MedPktSecond, SrcBytes	Dur, SrcBytes, MedBitsSecond, RazaoBytesOrigDest, DpDur	Dur, TotPkts, TotBytes, SrcBytes, MedPktSize, RazaoBytesOrigDest, DpDur	Dur, TotBytes, SrcBytes, RazaoBytesOrigDest, DPPkt, DpDur	RazaoBytesOrigDest, MedBitsSecond, Dur, SrcBytes, DpDur
2	MedBitsSecond, RazaoBytesOrigDest, Dur, MedPktSecond, TotBytes	MedBitsSecond, RazaoBytesOrigDest, Dur, MedPktSecond, TotBytes	MedBitsSecond, RazaoBytesOrigDest	TotBytes, MedPktSize, MedPktSecond, MedBitsSecond, RazaoBytesOrigDest, DpDur	TotBytes, SrcBytes, MedPktSize, MedBitsSecond, RazaoBytesOrigDest, DpDur	MedBitsSecond, RazaoBytesOrigDest, MedPktSecond, TotBytes
3	MedPktSize, SrcBytes, MedBitsSecond, RazaoBytesOrigDest, Dur	MedPktSize, SrcBytes, MedBitsSecond, RazaoBytesOrigDest, Dur	SrcBytes, MedPktSize, MedBitsSecond	Dur, SrcBytes, MedPktSecond, MedBitsSecond, RazaoBytesOrigDest, DPPkt, DpDur	TotPkts, SrcBytes, MedPktSize, MedPktSecond, MedBitsSecond, RazaoBytesOrigDest, DPPkt, DpDur	MedPktSize, SrcBytes, MedBitsSecond, RazaoBytesOrigDest, Dur
4	RazaoBytesOrigDest, MedPktSize, MedBitsSecond, TotBytes, MedPktSecond	RazaoBytesOrigDest, MedPktSize, MedBitsSecond, TotBytes, MedPktSecond	MedPktSize, MedBitsSecond, RazaoBytesOrigDest	SrcBytes, MedBitsSecond, RazaoBytesOrigDest, DPPkt, DpDur	Dur, SrcBytes, MedPktSize, MedPktSecond, DPPkt, DpDur	RazaoBytesOrigDest, MedPktSize, MedBitsSecond, MedPktSecond
5	Dur, MedPktSecond, MedBitsSecond, TotBytes, MedPktSize	Dur, MedPktSecond, MedBitsSecond, TotBytes, MedPktSize	Dur, TotBytes, MedPktSize, MedPktSecond, MedBitsSecond	Dur, TotPkts, TotBytes, SrcBytes, MedPktSize, RazaoBytesOrigDest, DPPkt, DpDur	Dur, TotPkts, SrcBytes, MedPktSize, RazaoBytesOrigDest, DPPkt, DpDur	Dur, MedPktSecond, MedBitsSecond, TotBytes, MedPktSize
6	RazaoBytesOrigDest, SrcBytes, MedBitsSecond, MedPktSecond, Dur	RazaoBytesOrigDest, SrcBytes, MedBitsSecond, MedPktSecond, Dur	MedPktSize, MedBitsSecond, RazaoBytesOrigDest	Dur, SrcBytes, MedPktSize, RazaoBytesOrigDest	SrcBytes, MedPktSize, MedPktSecond, MedBitsSecond, RazaoBytesOrigDest	RazaoBytesOrigDest, SrcBytes, MedBitsSecond, MedPktSecond, Dur, MedPktSize
7	Dur, RazaoBytesOrigDest, DpDur, MedBitsSecond, DPPkt	Dur, DpDur, RazaoBytesOrigDest, MedBitsSecond, DPPkt	Dur, RazaoBytesOrigDest, DpDur	Dur, TotPkts, SrcBytes, MedPktSize, MedBitsSecond, DpDur	Dur, TotBytes, SrcBytes, DPPkt	Dur, RazaoBytesOrigDest, DpDur, MedBitsSecond, DPPkt
8	MedBitsSecond, Dur, MedPktSecond, RazaoBytesOrigDest, MedPktSize	MedBitsSecond, Dur, MedPktSecond, RazaoBytesOrigDest, MedPktSize	Dur, MedPktSize, MedPktSecond, MedBitsSecond, RazaoBytesOrigDest	Dur, SrcBytes, MedPktSize, MedPktSecond, RazaoBytesOrigDest	Dur, MedPktSize, MedPktSecond, MedBitsSecond, RazaoBytesOrigDest	MedBitsSecond, Dur, RazaoBytesOrigDest, MedPktSecond, MedPktSize
9	RazaoBytesOrigDest, SrcBytes, MedPktSize, TotBytes, MedPktSecond	RazaoBytesOrigDest, SrcBytes, MedPktSize, TotBytes, MedPktSecond	SrcBytes, MedPktSize, MedPktSecond, MedBitsSecond, RazaoBytesOrigDest, DpDur	Dur, TotPkts, TotBytes, SrcBytes, MedPktSize, RazaoBytesOrigDest, DPPkt, DpDur	Dur, TotBytes, SrcBytes, MedPktSize, MedBitsSecond, RazaoBytesOrigDest, DPPkt, DpDur	RazaoBytesOrigDest, SrcBytes, MedPktSize, TotBytes, MedPktSecond, DpDur
10	RazaoBytesOrigDest, SrcBytes, MedPktSize, TotBytes, Dur	RazaoBytesOrigDest, SrcBytes, MedPktSize, TotBytes, Dur	SrcBytes, RazaoBytesOrigDest, DPPkt	Dur, TotPkts, MedPktSize, RazaoBytesOrigDes	Dur, TotPkts, MedPktSize, RazaoBytesOrigDest, DPPkt	RazaoBytesOrigDest, SrcBytes, MedPktSize, Dur
11	MedPktSize, SrcBytes, RazaoBytesOrigDest, Dur, TotBytes	MedPktSize, SrcBytes, RazaoBytesOrigDest, Dur, TotBytes	SrcBytes, MedPktSize	Dur, RazaoBytesOrigDest	SrcBytes , MedPktSize	MedPktSize, SrcBytes, RazaoBytesOrigDest, Dur
12	RazaoBytesOrigDest, SrcBytes, MedBitsSecond, MedPktSize, TotBytes	RazaoBytesOrigDest, SrcBytes, MedBitsSecond, MedPktSize, TotBytes	Dur, SrcBytes, MedPktSize, MedBitsSecond, RazaoBytesOrigDest, DpDur	Dur, TotPkts, SrcByte, MedPktSize, RazaoBytesOrigDest, DpDur	Dur, TotPkts, TotBytes, SrcBytes, MedPktSize, MedBitsSecond, DpDur	RazaoBytesOrigDest, SrcBytes, MedBitsSecond, MedPktSize, TotBytes, Dur, DpDur
13	MedBitsSecond, MedPktSecond, Dur, RazaoBytesOrigDest, SrcBytes	MedBitsSecond, MedPktSecond, Dur, RazaoBytesOrigDest, SrcBytes	Dur, TotPkts, SrcBytes, MedBitsSecond, RazaoBytesOrigDest	Dur, TotPkts, TotBytes, SrcBytes, MedPktSize, MedBitsSecond, RazaoBytesOrigDest, DpDur	Dur, TotPkts, TotBytes, SrcBytes, MedPktSize, MedPktSecond, RazaoBytesOrigDest, DPPkt	MedBitsSecond, MedPktSecond, Dur, RazaoBytesOrigDest, SrcBytes, TotPkts

parâmetros esses mostrados na Tabela 5.2. Segue uma descrição dos parâmetros dos algoritmos:

- **Árvore de Decisão J48**: C é o fator de confiança utilizado para podar a árvore, menores valores desse parâmetro geram maior poda; M é o número mínimo de instâncias em cada folha; *unprunned* indica se irá ocorrer poda na árvore e, caso o valor seja *false*, ocorrerá a poda; *numFolds* estabelece a quantidade de dados utilizados para redução de erros de poda, onde um conjunto é utilizado para poda e os demais para crescimento da árvore.

- ***Random Forest***: I é o número de árvores de decisão a serem geradas pelo algoritmo, sendo que, em geral, mais árvores produzem melhores resultados; K é o número de atributos e, quando o valor é zero, esse número é calculado em função da fórmula $\mathsf{NumAtributos} = \log_2(n) + 1$; M é o número mínimo de instâncias por folha.

- **Rede Bayesiana**: um algoritmo de busca é usado para encontrar as estruturas da rede, onde P é o parâmetro que fixa o número máximo de pais que um nó da rede pode ter; o algoritmo estimador tem a função de encontrar as tabelas de probabilidade condicional, onde α, usado para estimar essas probabilidades, é a contagem inicial para cada valor.

- **SVM**: C é o parâmetro de complexidade usado para construir o hiperplano entre as duas classes, serve para controlar a suavidade das margens das classes; a função *kernel* usada é a RBF, que é a mais utilizada apesar de não ser o padrão dessa classe, conforme apresentada na Seção 3.2.5.4, onde γ é um parâmetro dessa função.

- **RNA**: H identifica o número de camadas ocultas que serão utilizadas na rede neural; L é a taxa de aprendizagem e determina a velocidade de aprendizado do algoritmo *backpropagation*; M é o *momentum*, que modifica a regra inicial do algoritmo *backpropagation*, evitando parar em locais ótimos; N é o número de *epochs*, que é uma medida da quantidade de vezes que todos os vetores de treinamento são usados para atualizar os pesos.

- **K-NN**: K é o número de vizinhos mais próximos usados na classificação; o algoritmo de busca do vizinho mais próximo é o padrão da classe.

- **KMeans**: k é o número de *clusters* e, como esse algoritmo foi utilizado para classificação binária, foram escolhidos dois *clusters*; I é o número máximo de

iterações; a função de distância utilizada é a Euclidiana, usada para a medida de similaridade.

Tabela 5.2: Principais parâmetros dos algoritmos de classificação

Parâmetros dos algoritmos						
Aprendizado Supervisionado						Aprendizado Não Supervisionado
J48	Random Forest	Rede Bayesiana	SVM	RNA	K-NN	KMeans
C = 0,25 M = 2 unprunned = false numFolds = 3	I = 100 K = 0 M = 1	algoritmo de busca = K2 , P =1 algoritmo estimador = Simple estimator, $\alpha = 0.5$	C = 1 kernel = RBF, $\gamma = 0{,}01$	H = (atribs + classes)/2 L = 0.3 M = 0.2 N = 500	K = 1 algoritmo de busca = Linear NN Search	K = 2 I = 500 função de distância = Euclidiana

5.2.1 Análise das Métricas de Avaliação

Nesta subseção, as métricas adequadas para bases de dados desbalanceadas são analisadas. A acurácia não será utilizada por não ser indicada para esse tipo de base, pois levaria a conclusões errôneas sobre o desempenho dos modelos. Os modelos serão analisados quanto ao *malware* utilizado, protocolo, atividade maliciosa, arquitetura da *botnet*, balanceamento das classes e quantidade de fluxos no cenário.

Analisando a precisão conforme o gráfico na Figura 5.2, percebe-se que o algoritmo que mais se destacou nessa métrica foi o *Random Forest*, pois em nenhum cenário teve um desempenho ruim. Em seguida fica o Árvore de Decisão J48, com desempenho muito semelhante, e posteriormente vem o K-NN, porém ambos não obtiveram bom desempenho no cenário 7. Entre todos os algoritmos, somente o *Random Forest* conseguiu uma boa precisão nesse cenário. Em terceiro fica o modelo gerado em Rede Bayesiana, que possui um desempenho inferior aos anteriores, mas destacou-se mais nos cenários 10 e 11, empatando com eles. Todavia, possui um péssimo desempenho nos cenários 4, 5 e 7, desempenho esse inferior a 0,5.

Os modelos gerados pelos algoritmos RNA, SVM e *KMeans* foram muito aquém dos demais. O primeiro ainda atingiu um resultado mediano nos cenários 3, 10 e 11, enquanto os outros praticamente zeraram em todos os cenários. Analisando quanto ao *malware* utilizado pela *botnet*, constata-se que os algoritmos RNA e Rede Bayesiana se destacaram na maioria dos cenários que utilizaram *Rbot* (3, 10 e 11), mas não atingiram bom desempenho no cenário 4, que também utiliza esse *bot*. Também pode ser observado que, no cenário 7, que é o único que representa o *malware* Sogou, os modelos obtiveram baixa precisão, com exceção do *Random Forest*, o que pode ser um indicativo de dificuldade para se detectar o tráfego desse *bot*. Tomando como referência os protocolos utilizados pelas *botnets*, através da precisão é possível perceber que o RNA só alcançou resultado nos cenários que utilizaram IRC.

Realizando uma análise quanto ao balanceamento da classe nos cenários, verifica-se que o cenário 7 possui o menor percentual de tráfego de *botnet* (0,05%). Nesse cenário, apenas o *Random Forest* se destacou. Porém, no cenário 4, que também possui um baixo percentual de tráfego de *botnet* (0,15%), além do Random Forest, os algoritmos Árvore de Decisão J48 e K-NN conseguiram uma precisão acima de 0,6. No cenário 3 (0,56% de tráfego malicioso), com exceção do SVM e *Kmeans*, os modelos alcançaram uma boa precisão, onde a menor foi obtida pelo RNA (0,5). Esse foi um dos três cenários em que o RNA alcançou um resultado mediano, indicando que não foi apenas o desbalanceamento das classes o causador do desempenho inferior dos algoritmos.

Uma análise com relação à quantidade de fluxos por cenário também não é conclusiva, visto que no cenário 11, possuidor do menor número de fluxos, e no cenário 3, possuidor da maior quantidade de fluxos, a maioria dos modelos alcançaram boa precisão.

Examinando a precisão alcançada pelos modelos com relação às atividades maliciosas realizadas nos cenários, verifica-se que *Random Forest*, *J48* e *K-nn* obtiveram precisão acima de 0,6 em todos os cenários onde eram realizados ataques *DDoS* (cenários 4, 10 e 11). Em dois dos três cenários que os modelos RNA alcançaram precisão (10 e 11), também eram realizados ataques *DDoS*, o que indicaria uma possível especialização desse algoritmo para esse tipo de atividade maliciosa. Porém, no cenário 4 a precisão foi zero, inviabilizando essa análise. Para as atividades de escaneamento de portas e SPAM, J48 e *Random Forest* alcançaram precisão superior a 0,7 e K-nn superior a 0,5 nos cenários estudados.

Fazendo uma análise observando a arquitetura das *botnets* nos cenários, percebe-se que os algoritmos do paradigma simbólico (J48 e *Random Forest*) conseguiram altos valores de precisão, independente da arquitetura, ou seja, o que inclui a arquitetura decentralizada do cenário 12. Enquanto Rede Bayesiana e K-NN obtiveram a precisão em torno de 0,5 neste cenário, o RNA só conseguiu precisão nos cenários de arquitetura centralizada, apesar de não ter conseguido em todos os cenários dessa arquitetura. Comparando os resultados da precisão deste trabalho com [2] e [3], os modelos gerados pelo algoritmo *Random Forest* deste trabalho conseguiram um resultado mais consistente, superando em todos os cenários os modelos comparados nos trabalhos [2] e [3].

Através do *recall*, reproduzido no gráfico da Figura 5.3, pode ser compreendida a fração de instâncias da classe *botnet* previstas corretamente pelos modelos, equi-

valente à taxa de positivos verdadeiros. Analisando essa métrica, verifica-se que os modelos gerados pelo algoritmo de Rede Bayesiana se destacaram nos cenários 3, 4 e 5. Nos outros cenários, houve empate com *Random Forest* e J48. Destaca-se o fato de nenhum modelo atingir um bom *recall* no cenário 7: *Random Forest*, Rede Bayesiana e K-NN tiveram um desempenho semelhante e K-NN destacou-se com 0,33. O baixo *recall* no cenário 7 também foi observado no trabalho [2] e [3], onde K-NN também obteve o maior *recall* nesse cenário, porém bem inferior (0,079) ao encontrado aqui. Os modelos RNA, SVM e *KMeans* continuam com desempenho abaixo dos demais: SVM zerou em todos os cenários; RNA obteve valor acima de zero nos cenários 3, 10 e 11, destacando-se nos cenários 10 e 11; *KMeans* teve um *recall* superior à precisão, destacando-se no cenário 12.

Analisando quanto ao *malware* utilizado nos cenários, constata-se que o algoritmo RNA apresentou *recall* acima de zero na maioria dos cenários que utilizaram *Rbot* (3, 10 e 11), mas zerou no cenário 4, que também utiliza esse *bot*. O algoritmo de Rede Bayesiana, que também havia destacado sua precisão nesses cenários, apresentou alto *Recall* em todas as famílias de *bots*, com exceção da Sogou (cenário 7). O algoritmo *Kmeans* alcançou seu maior resultado, e o único acima de 0,5, no *bot* NSIS.ay. Assim como na precisão, pôde ser observado que, no único cenário do *malware* Sogou (cenário 7), os modelos, inclusive o *Random Forest*, obtiveram baixo *recall*.

Quanto ao protocolo utilizado, o RNA só alcançou resultado nos cenários que utilizaram IRC. *KMeans* alcançou o maior *Recall* entre todos os outros modelos dos algoritmos no único cenário que utiliza o protocolo HTTP (cenário 12). Investigando o *Recall* referente ao balanceamento das classes nos cenários, verifica-se que nenhum modelo se destacou no cenário 7, possuidor do menor percentual de tráfego de *botnet* (0,05%). Porém, no cenário 4, que também possui um baixo percentual de tráfego *botnet* (0,15%), Random Forest, J48, Rede Bayesiana e K-NN conseguiram *recall* acima de 0,5. No cenário 3 (0,56% de tráfego malicioso), esses modelos alcançaram *recall* superior a 0,8, sendo que o menor foi obtido pelo RNA (0,36). A análise do *recall* com relação a quantidade de fluxos por cenário é semelhante à análise da precisão.

Examinando o *recall* alcançado pelos modelos com relação às atividades maliciosas realizadas nos cenários, verifica-se que *Random Forest*, *J48*, *K-nn* e Rede Bayesiana obtiveram *recall* acima de 0,5 em todos os cenários onde eram realizados ataques DDoS (4, 10 e 11), lembrando que Rede Bayesiana obteve baixa precisão no cenário 4 (0,182). Em dois dos três cenários que os modelos RNA alcançaram um

bom *recall* (10 e 11), eram realizados ataques do tipo *DDoS*. Porém, no cenário 4 (DDoS), o *recall* foi zero. Para as atividades de escaneamento de portas e SPAM, os resultados foram semelhantes à precisão, porém, no cenário 5, J48, *Random Forest* e K-NN obtiveram um *recall* mediano (0,444, 0,59 e 0,49 respectivamente), enquanto Rede Bayesiana alcançou alto *recall*, inclusive no cenário 5, que tinha obtido baixa precisão (0,229). KMeans alcançou o maior *recall* entre todos os outros modelos dos algoritmos no cenário 1,2 em que a *botnet* realiza sincronização, porém sua precisão foi próxima a zero. Isso é devido a este modelo ter atingido um baixo número de falsos negativos, porém um número muito elevado de falsos positivos, ao ponto de quase igualar ao número de verdadeiros negativos.

A análise, observando a arquitetura das *botnets* nos cenários, é semelhante à da precisão, porém os algoritmos do paradigma simbólico (Árvore de Decisão J48 e *Random Forest*), que haviam conseguido altos valores de precisão independente da arquitetura, apresentam *recall* mediano no cenário 5 e baixo no cenário 7. Entretanto, estes algoritmos continuaram a ter o maior desempenho na arquitetura descentralizada (cenário 12). O RNA só conseguiu bom *recall* nos cenários de arquitetura centralizada, apesar de não ter conseguido em todos os cenários dessa arquitetura. Rede Bayesiana alcançou alto *recall* nas arquiteturas de *botnets* de todos os cenários, com exceção do cenário 7.

A partir da análise da *f-measure* na Figura 5.4, que é uma média harmônica entre *recall* e precisão, pode-se ter uma ideia melhor da qualidade dos modelos. Para alcançar um alto resultado nessa métrica é necessário também ter conseguido uma alta precisão e um alto *recall*. Neste aspecto, os algoritmos do paradigma simbólico se destacaram em todos os cenários, onde o *Random Forest* conseguiu um resultado um pouco melhor que o J48. Em seguida fica o K-NN e, após, Rede Bayesiana, que teve um resultado inferior a 0,5 nos cenários 4, 5 e 7. RNA e SVM mantiveram o fraco desempenho. A análise quanto ao *malware* utilizado nos cenários é semelhante às métricas anteriores, porém Rede Bayesiana, diferente do *recall*, conseguiu *F-Measure* acima de 0,6 apenas nos cenários dos *bots* Neris, Menti, Murlo e Virut. Kmeans teve baixo desempenho em todos os bots, inclusive no NSIS.ay, onde havia conseguido alto *recall*. No único cenário do *malware* Sogou (cenário 7), os modelos, inclusive o *Random Forest*, obtiveram baixo *f-measure*.

Quanto ao protocolo utilizado, a análise é semelhante. O algoritmo *Kmeans* que havia conseguido o maior *recall* no cenário 12, não conseguiu ter *F-Measure* distante de zero em nenhum cenário. A análise da *F-Measure* referente ao balanceamento das classes nos cenários é semelhantes às análises anteriores, concluindo que não

é somente o desbalanceamento das classes o causador do desempenho inferior dos algoritmos.

A análise com relação à quantidade de fluxos por cenário também é semelhante às anteriores. Nos cenários que possuem o maior (cenário 3) e o menor (cenário 11) número de fluxos, os modelos possuíram desempenho semelhante. Conclui-se que a quantidade de fluxos não interferiu no desempenho dos algoritmos. Examinando o *F-Measure* alcançado pelos modelos com relação às atividades maliciosas realizadas nos cenários, *Random Forest*, *J48* e *K-nn* obtiveram *F-Measure* acima de 0,5 em todos os cenários onde eram realizados ataques DDoS (4, 10 e 11), concluindo ser esses três algoritmos os mais indicados para detectar esse tipo de ataque. Para as atividades de escaneamento de portas e SPAM, J48, Random Forest e Rede Bayesiana obtiveram um alto valor de F-Measure, com exceção do cenário 5 (0,56 e 0,66 e 0,35 respectivamente). *K-nn* alcançou desempenho nessas atividades superior a 0,5. No cenário pré-ataque de sincronização (cenário 12), Random Forest conseguiu o maior f-measure (0,807) seguido pelo J48 (0,747), mostrando que os algoritmos do paradigma simbólico são os melhores para detectar as atividades maliciosas estudadas e também detectar o tráfego de comando e controle pré-ataque.

A análise observando a arquitetura das *botnets* nos cenários mostra que os algoritmos do paradigma simbólico (Árvore de Decisão J48 e *Random Forest*) só não desempenharam bem no cenário 7, sendo indicados independente da arquitetura. Através da análise do tempo de treinamento de cada algoritmo, através da Figura 5.5, verifica-se que o algoritmo com o menor tempo de treinamento é o K-NN. Isso é devido ao seu tipo de aprendizado, pois não é construído um modelo no conjunto de treinamento. Em contrapartida, ao se classificar uma nova instância, recupera-se o conjunto de instâncias similares que será utilizado para a classificação, o que torna a classificação mais custosa. Nos experimentos realizados, o algoritmo com maior tempo de treinamento foi o SVM. Com relação ao tempo para os algoritmos realizarem a classificação, após a fase de treinamento, esse tempo é extremamente rápido para todos os modelos e, portanto, não foi comparado neste trabalho. A velocidade de detecção dependerá da arquitetura onde o modelo será implementado.

Em síntese, foram avaliados e comparados os desempenhos dos algoritmos do aprendizado supervisionado dos paradigmas simbólico (Árvore de Decisão J48 e *Random Forest*), estatístico (Rede Bayesiana e *Support Vector Machines*), conexionista (Redes Neurais Artificiais *Multilayer* do tipo Perceptron) e baseado em exemplos (*K-Nearest Neighbor*). No aprendizado não supervisionado, avaliou-se o algoritmo de *Clustering KMeans*. Além disso, o algoritmo *Random Forest* também

representa o *Ensemble Learning*, do tipo randomização. Foi apurado que os algoritmos RNA, SVM e *Kmeans* foram muito aquém dos demais, com apenas o RNA conseguindo um desempenho mediano nos cenários 3, 10 e 11. No cenário 7, os algoritmos alcançaram desempenho inferior e até mesmo o modelo gerado pelo algoritmo *Random Forest*, que alcançou alta precisão, foi avaliado com baixo *recall*. Isso foi causado pela taxa de falsos negativos que esse algoritmo apresentou nesse cenário. Esse é o cenário mais desbalanceado. Porém, em outros cenários desbalanceados (cenário 4 e 3), os resultados gerais foram melhores. Convém lembrar que o cenário 7 é o único que representa o *malware* Sogou e esse também pode ser um fator do baixo desempenho dos modelos nesse cenário. Apesar dos modelos gerados pelo algoritmo de Rede Bayesiana apresentarem alto *recall*, a precisão foi baixa devido a esses alcançarem altas taxas de falsos positivos, e por isso é importante analisar as métricas em conjunto. A métrica *f-measure* consolida essas duas avaliações e, a partir dela, pode-se ter uma melhor visualização da qualidade dos modelos gerados. Conclui-se que os algoritmos do paradigma simbólico desempenharam melhor independente do *malware*, do protocolo, da atividade maliciosa e da arquitetura da *botnet*, além de serem os melhor sucedidos na detecção preventiva. Com exceção do cenário 7, nos outros cenários desbalanceados esses algoritmos também obtiveram melhor desempenho. Verificou-se que a quantidade de fluxos não interferiu no desempenho dos modelos. Dentre esses dois algoritmos que se destacaram, o *Random Forest* conseguiu um desempenho um pouco melhor. Os modelos gerados por esse algoritmo conseguiram melhor desempenho que os modelos gerados pelo algoritmo melhor avaliado em [2] e [3], que foi o K-NN. Esse comparativo pode ser visto no gráfico da Figura 5.1.

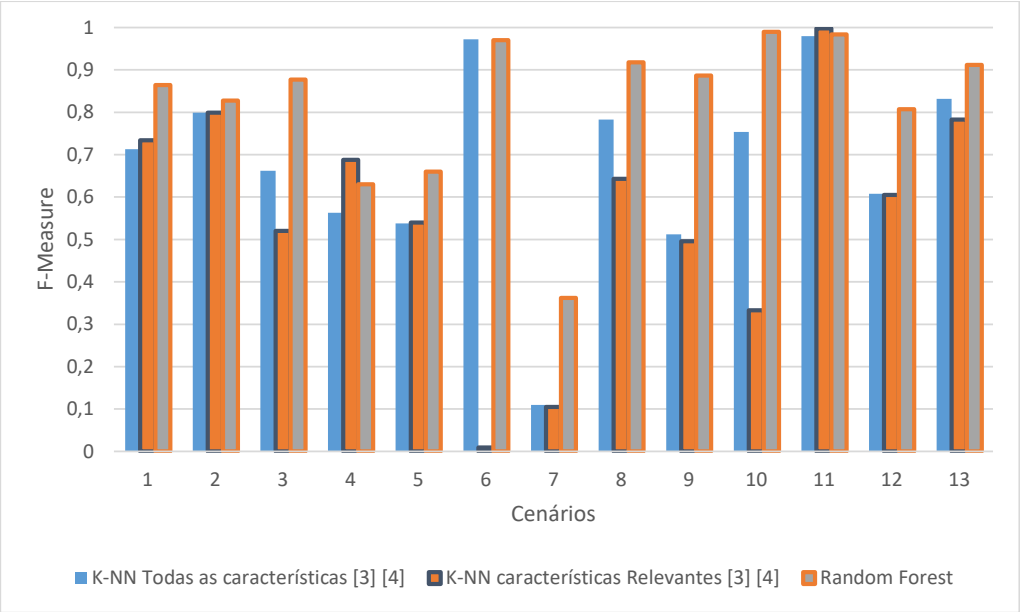

Figura 5.1: *f-measure* do Random Forest em comparação com K-NN [2, 3]

Após esses experimentos, decidiu-se por realizar uma otimização nos três algoritmos que apresentaram os piores desempenhos: RNA, SVM e *KMeans*. Como esses algoritmos são dependentes dos seus parâmetros, faz-se necessário buscar parâmetros que melhorem o desempenho desses modelos. Buscando otimizar o processo de variação dos parâmetros, foi utilizado o metaclassificador *CvParameter* [78]. Trata-se de uma classe do *Weka* que permite otimizar um número arbitrário de parâmetros através de validação cruzada para qualquer classificador. Depois de encontrar a melhor configuração possível, uma instância do classificador base será treinada com esses parâmetros, que serão utilizados para as próximas previsões. A próxima subseção apresenta esse processo.

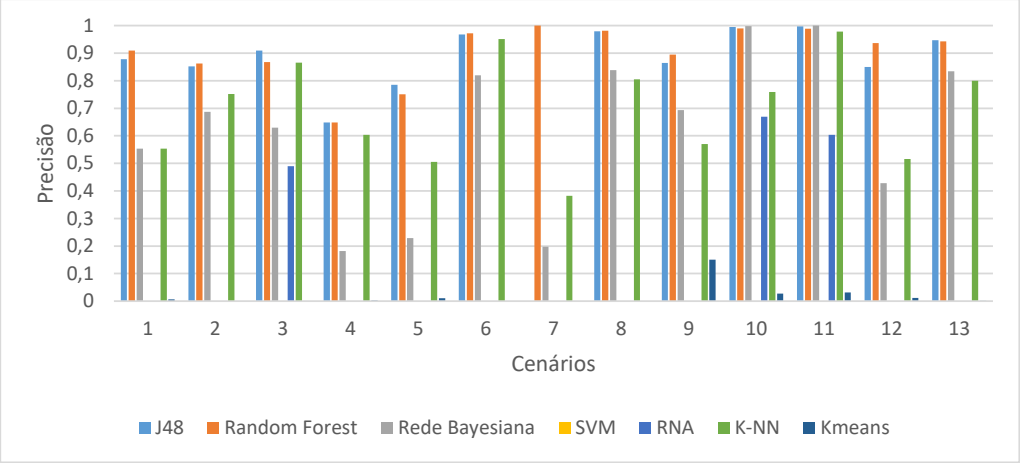

Figura 5.2: Precisão dos modelos nos cenários

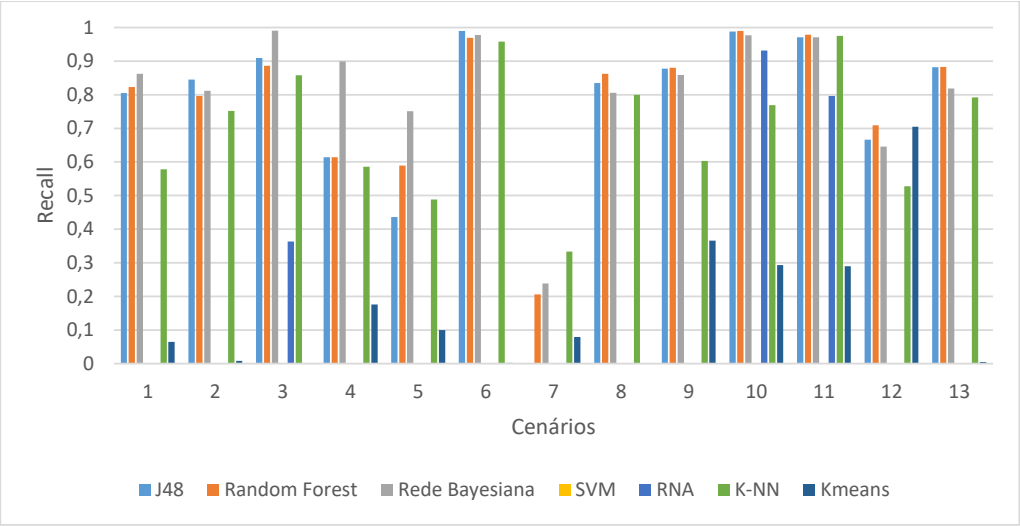

Figura 5.3: *Recall* dos modelos nos cenários

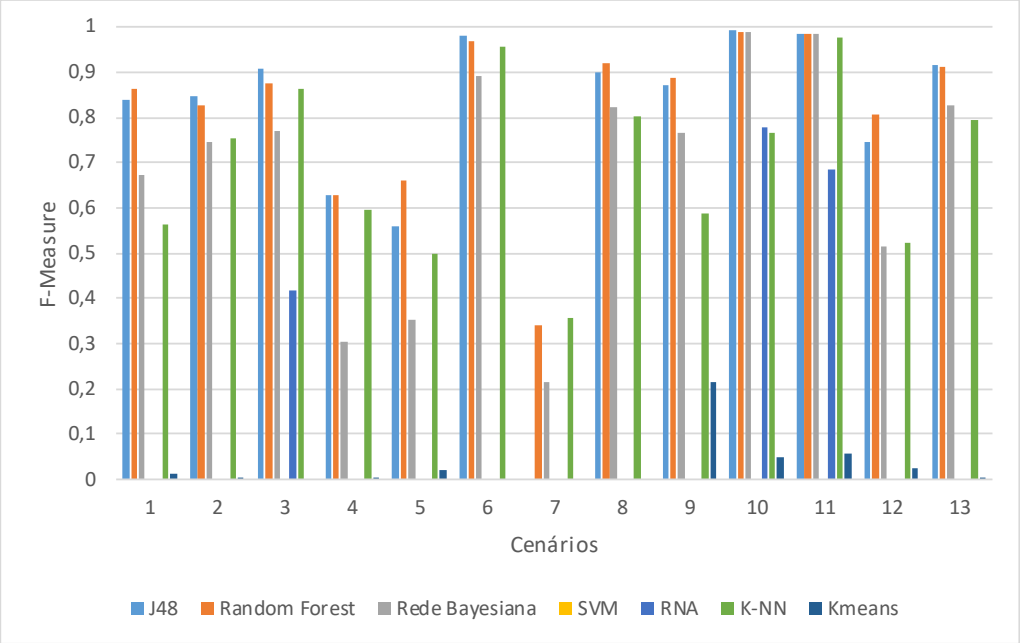

Figura 5.4: *F-measure* dos modelos nos cenários

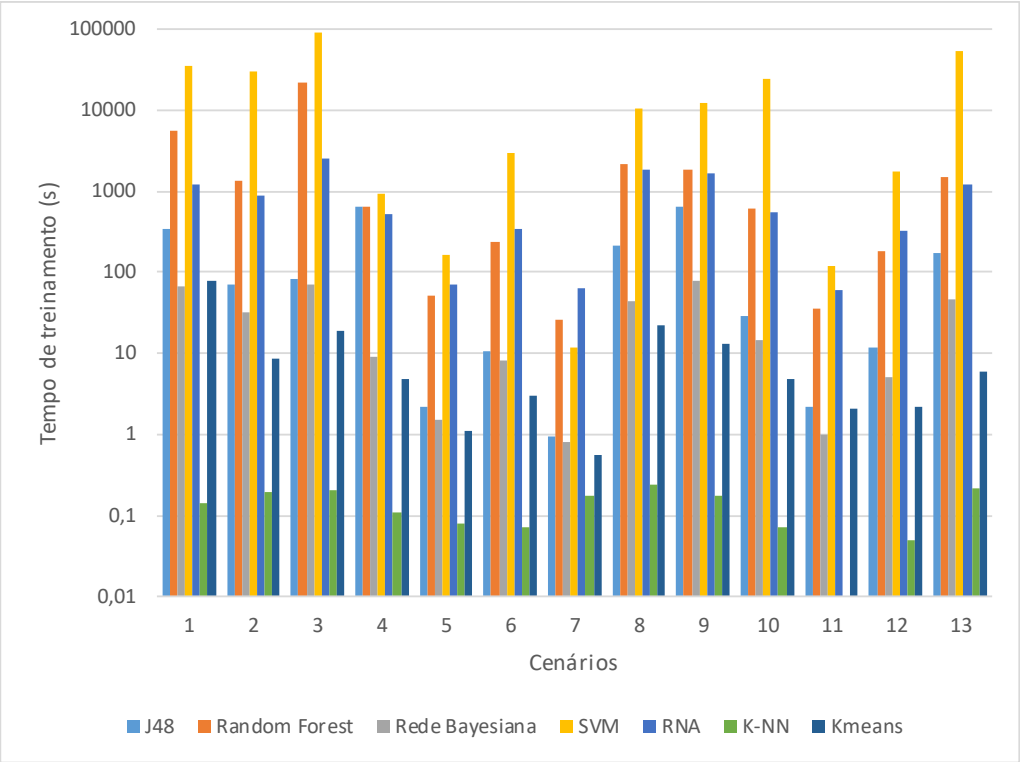

Figura 5.5: Tempo de treinamento dos modelos nos cenários

5.2.2 Otimização dos algoritmos RNA, SVM e *KMeans*

Para os próximos experimentos foi utilizado o metaclassificador *CVParameter* para otimizar os principais parâmetros dos algoritmos de classificação. Esse metaclassificador utiliza validação cruzada, garantindo que a validação dos parâmetros seja realizada em um conjunto separado de teste. Os seguintes parâmetros do algoritmo RNA foram variados: H (número de camadas ocultas) de 0 a 4; L (taxa de aprendizagem) de 0,1 a 0,6 e M (*Momentum*) de 0,1 a 0,6. Para o algoritmo SVM, os parâmetros subsequentes foram variados: C (complexidade) de 2 a 8 e G (γ) de 0,01 a 0,1. Os melhores parâmetros encontrados por cenário estão na Tabela 5.3. Para o algoritmo *KMeans*, a função de distância Euclidiana foi trocada para a Manhattan. Essa função usa como medida de similaridade a associação entre as instâncias e os centroides.

Após essa etapa, o desempenho dos algoritmos não teve melhora significativa, como mostram os gráficos das Figuras 5.6, 5.7 e 5.8. SVM, que antes havia zerado suas métricas em todos os cenários, agora alcançou alta precisão no cenário 11, porém o *recall* foi próximo de zero, indicando que esse algoritmo apresentou, nesse cenário, uma alta taxa de falsos negativos. Acertando muito poucas instâncias que eram *botnets*, o péssimo desempenho desse algoritmo está refletido na baixa *f-measure*

atingida.

RNA não zerou os cenários 3, 10 e 11 novamente, porém conseguiu um desempenho apenas pouco melhor, reflexo da otimização dos parâmetros. Os valores de *F-Measure* anterior eram: cenário 3, 0,417; cenário 10, 0,779 e cenário 11, 0,686. Após a otimização: cenário 3, 0,421; cenário 10, 0,78 e cenário 11, 0,71.

O algoritmo *Kmeans*, em alguns cenários, alcançou um desempenho um pouco melhor e, em outros, pior. Por todos esses aspectos, conclui-se que a otimização dos algoritmos RNA, SVM e *KMeans* não foi suficiente para melhorar o desempenho dos modelos nos cenários dos experimentos.

Tabela 5.3: Parâmetros selecionados por cenário para os algoritmos RNA e SVM

	Parâmetros Selecionados	
Cen	RNA	SVM
1	-H 1 -L 0.1 -M 0.2	-C 2 -G 0.01
2	-H 1 -L 0.1 -M 0.2	-C 2 -G 0.01
3	-H 1 -L 0.1 -M 0.2	-C 2 -G 0.01
4	-H 1 -L 0.1 -M 0.2	-C 2 -G 0.01
5	-H 1 -L 0.1 -M 0.2	-C 2 -G 0.01
6	-H 1 -L 0.1 -M 0.2	-C 2 -G 0.01
7	-H 1 -L 0.1 -M 0.2	-C 2 -G 0.01
8	-H 1 -L 0.1 -M 0.2	-C 2 -G 0.01
9	-H 1 -L 0.1 -M 0.2	-C 2 -G 0.01
10	-H 2 -L 0.1 -M 0.2	-C 1 -G 0.01
11	-H 2 -L 0.1 -M 0.2	-C 8 -G 0.01
12	-H 1 -L 0.1 -M 0.2	-C 2 -G 0.01
13	-H 1 -L 0.1 -M 0.2	-C 2 -G 0.01

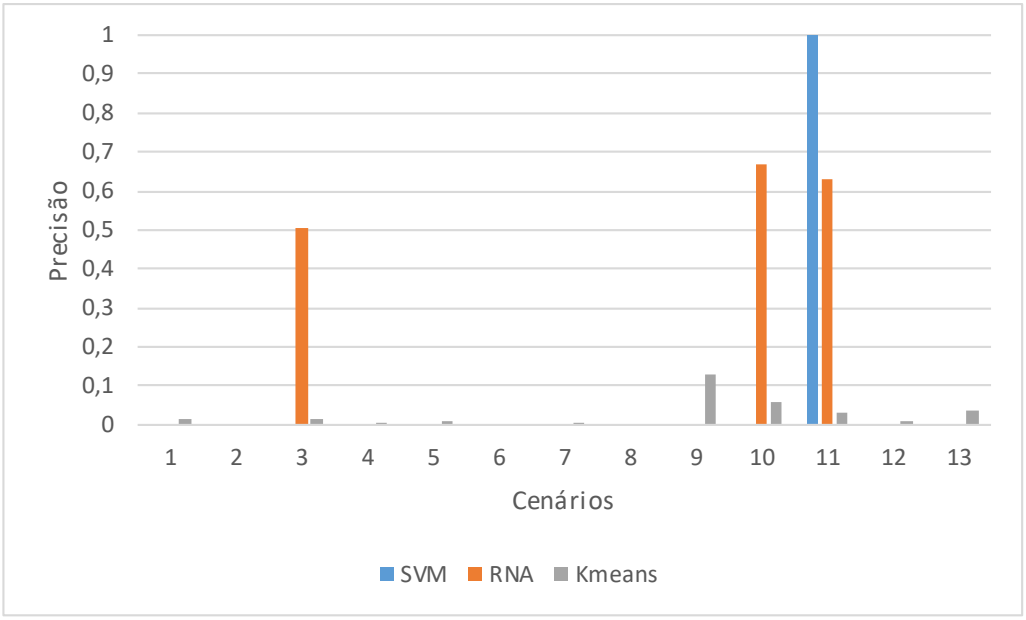

Figura 5.6: Precisão dos modelos nos cenários após otimização dos algoritmos

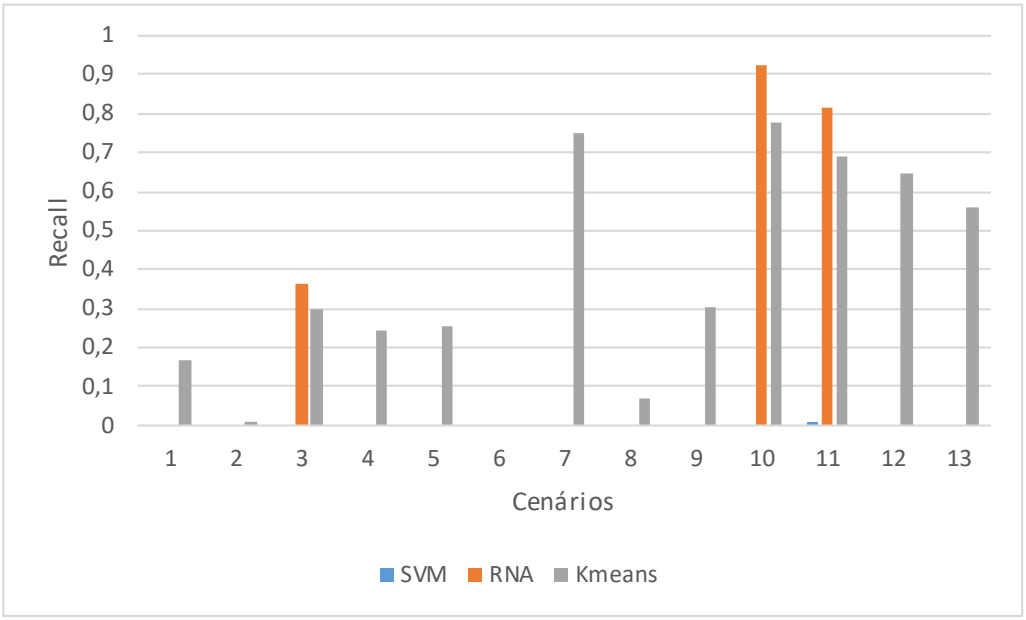

Figura 5.7: *Recall* dos modelos nos cenários após otimização dos algoritmos

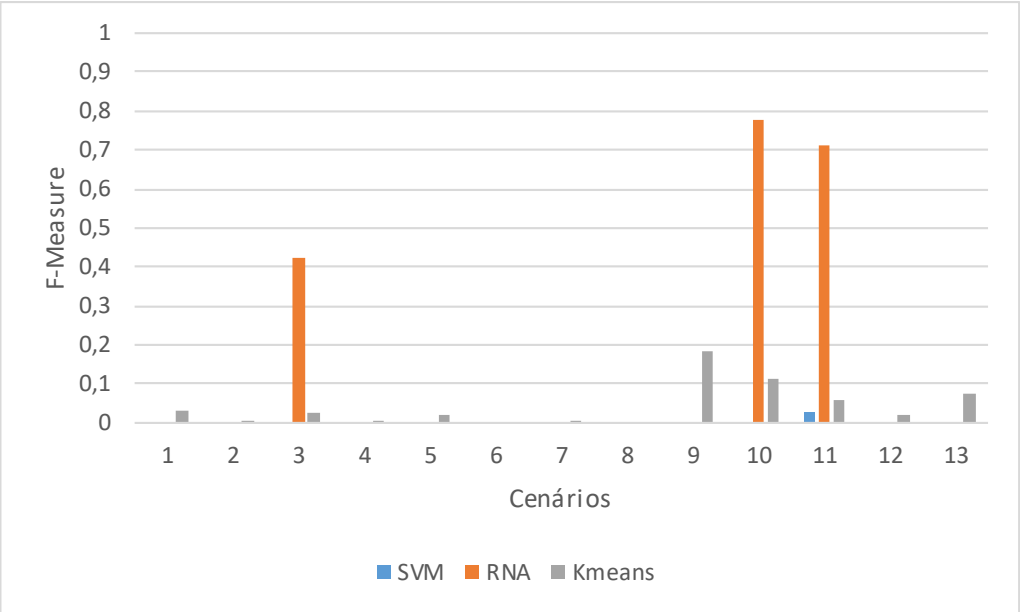

Figura 5.8: *F-measure* dos modelos nos cenários após otimização dos algoritmos

5.2.3 Balanceamento de Classes para os algoritmos RNA e SVM

Sabe-se que modelos utilizados para identificar eventos raros - como é o caso desse trabalho, onde a classe de interesse *botnet* está em proporção muito inferior à classe normal -, possuem maior dificuldade do que em cenários com classes balanceadas. Existem várias abordagens para lidar com esse problema [51], uma delas é a escolha das métricas corretas para avaliar os modelos. Recomenda-se que a acurácia, neste caso, não seja utilizada. Ao invés disso é recomendada a utilização de métricas como precisão, *recall* e *F-Measure*, métricas essas que foram utilizadas neste trabalho. Outra abordagem é a realização de *oversampling* ou *undersampling*. Na primeira são adicionadas cópias de instâncias para repor a classe de menor frequência, podendo ser geradas amostras sintéticas através da técnica SMOTE (*Synthetic Minority Oversampling Technique*) [79]. No *undersampling* são excluídas instâncias da classe de maior frequência para balancear o conjunto de dados. Há ainda a possibilidade da utilização de classificação ou aprendizado sensível ao custo: essas técnicas inserem um custo para as classificações erradas, assim o algoritmo pode ser configurado para ficar mais atento à classe de menor proporção.

A intenção desta etapa do trabalho é realizar o balanceamento das classes sem gerar amostras artificiais, copiando instâncias da classe minoritária ou removendo instâncias da classe majoritária. Sendo assim, foi utilizada a funcionalidade do *Weka* chamada *ClassBalancer*. Através dessa técnica, nenhuma instância é adicionada ou apagada, então o número de instâncias de cada classe permanece o mesmo. Os

pesos das instâncias são ajustados para que cada classe tenha o mesmo peso total e a soma total de pesos em todas as instâncias é mantida. Neste trabalho, o ajuste foi realizado apenas nos dados de treinamento, não havendo alterações nos dados de teste. Exemplificando a técnica *ClassBalancer*: se a classe Normal ocorre N vezes mais que a classe *botnet*, é dado a qualquer instância da classe *botnet*, que seja erroneamente classificada como normal, N vezes o custo de uma instância da classe normal que seja classificada como *botnet*.

Os resultados apresentados nos gráficos das Figuras 5.9, 5.10 e 5.11 mostram que todos os algoritmos obtiveram um grande aumento no *recall*, porém apresentaram baixa precisão. Isso indica que o balanceamento das classes induziu os modelos a diminuir o número de falsos negativos, aumentando o número de acertos da classe *botnet*, como esperado. Porém, houve um grande aumento no número de falsos positivos. O algoritmo RNA piorou nos cenários 3, 10 e 11, apresentando uma precisão inferior após o balanceamento. Nos outros cenários, houve um incremento muito pequeno no desempenho que pode ser visualizado no gráfico *f-measure* da Figura 5.11, concluindo que o balanceamento das classes não alavancou o desempenho dos modelos gerados por esse algoritmo. Para os modelos gerados pelo algoritmo SVM, a mudança foi mais nítida. Antes, esses modelos zeravam em quase todos os cenários, mas após o balanceamento, como dito anteriormente, os modelos tiveram um aumento no *recall*. No entanto, a precisão se manteve muito baixa, invalidando esses modelos para a tarefa de classificação.

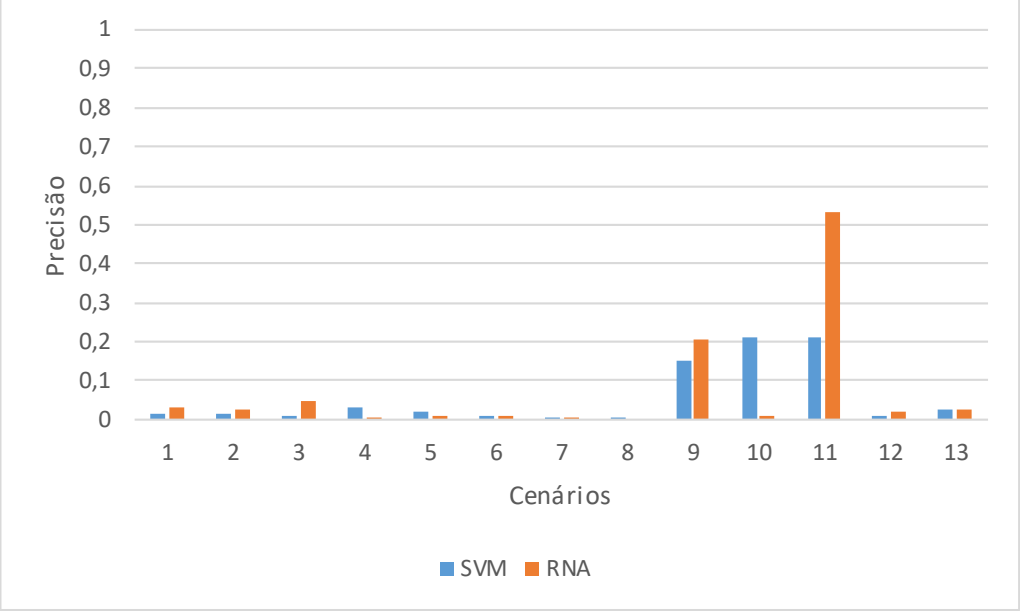

Figura 5.9: Precisão dos modelos nos cenários após balanceamento de classes

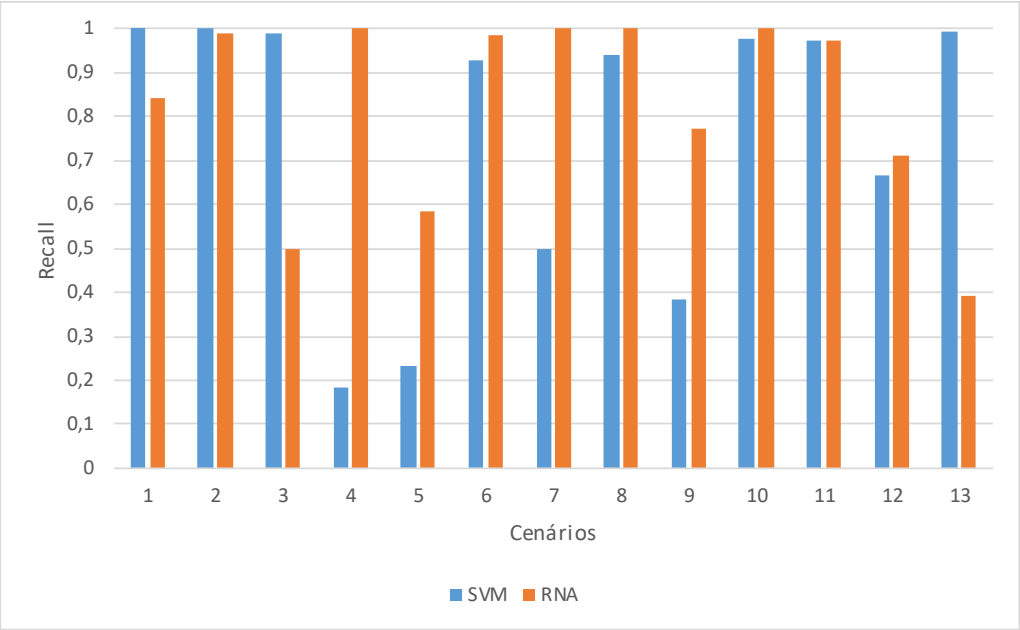

Figura 5.10: *Recall* dos modelos nos cenários após balanceamento de classes

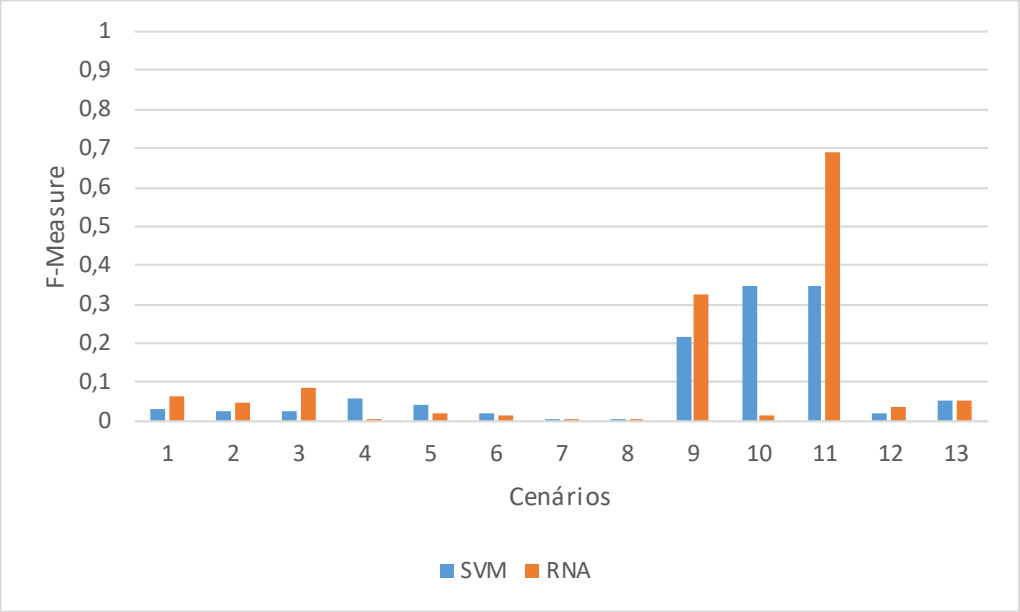

Figura 5.11: *F-measure* dos modelos nos cenários após balanceamento de classes

5.2.4 Balanceamento de classes no cenário 7 para os algoritmos Árvore de Decisão J48 e *Random Forest*

Sabendo que os modelos gerados pelos algoritmos *Random Forest* e Árvore de Decisão J48 obtiveram a melhor performance dentre os algoritmos avaliados, onde apenas no cenário 7 o desempenho foi baixo, mostrou-se necessário descobrir se a baixa performance dos algoritmos do paradigma simbólico foi devido a esse cená-

rio ser o mais desbalanceado entre todos os outros. Para tal, foi então realizado o balanceamento de classes desses dois algoritmos no cenário 7 através da técnica *ClassBalancer*. Os resultados estão nos gráficos das Figuras 5.12, 5.13 e 5.14. O processo de balanceamento reduziu a precisão do algoritmo *Random Forest* e aumentou o seu *recall*. O resultado, portanto, foi um pequeno aumento no *f-measure*. O algoritmo J48 conseguiu o aumento de todas as métricas, porém o desempenho ainda é ínfimo. Esses resultados mostraram que o balanceamento das classes também não elevou, de forma considerável, o desempenho desses algoritmos nesse cenário, indicando que o motivo do baixo desempenho nesse cenário pode ser devido à dificuldade de detecção do tráfego gerado pelo *bot Sogou*.

Após os experimentos, verificou-se que o desempenho dos modelos gerados pelos algoritmos RNA, SVM e *KMeans* foi muito aquém dos demais, mesmo após otimização e balanceamento de classes. Isso pode ser devido a uma certa incapacidade desses modelos para as especificidades do problema deste trabalho, porém uma otimização de hiper-parâmetros mais detalhada pode conseguir alavancar os respectivos desempenhos. Os algoritmos do paradigma simbólico obtiveram o melhor desempenho em todas as análises, indicando que esse paradigma é o mais recomendado para o problema. O *Random Forest* conseguiu desempenho um pouco superior, o que pode ser devido ao fato desse método combinar classificadores (*Ensemble*). Métodos do tipo *Ensemble* tendem a apresentar melhor desempenho do que usar um único classificador [47] [48].

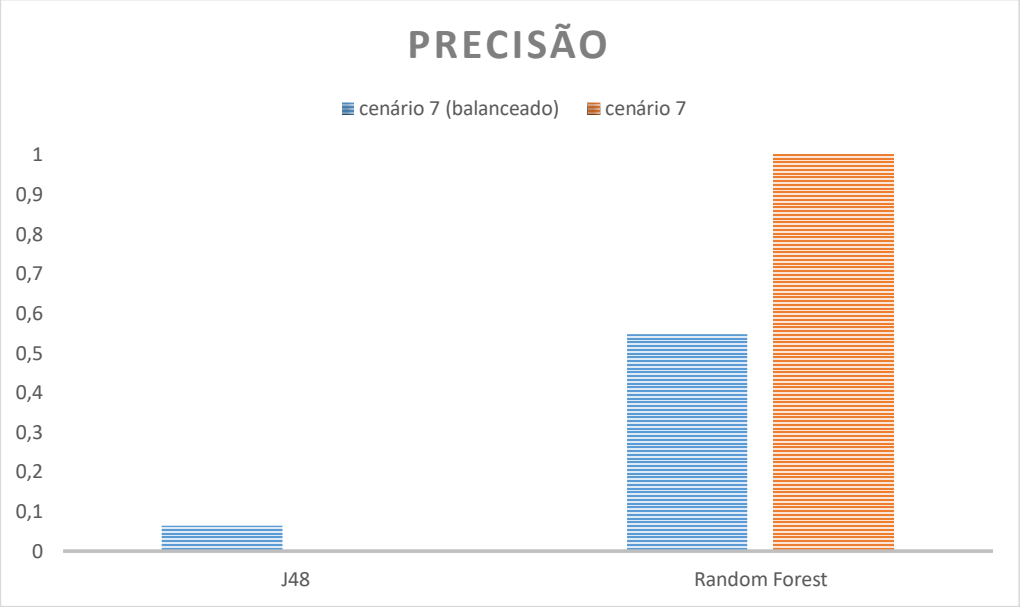

Figura 5.12: Precisão dos modelos do paradigma simbólico no cenário 7 antes e depois do balanceamento de classes

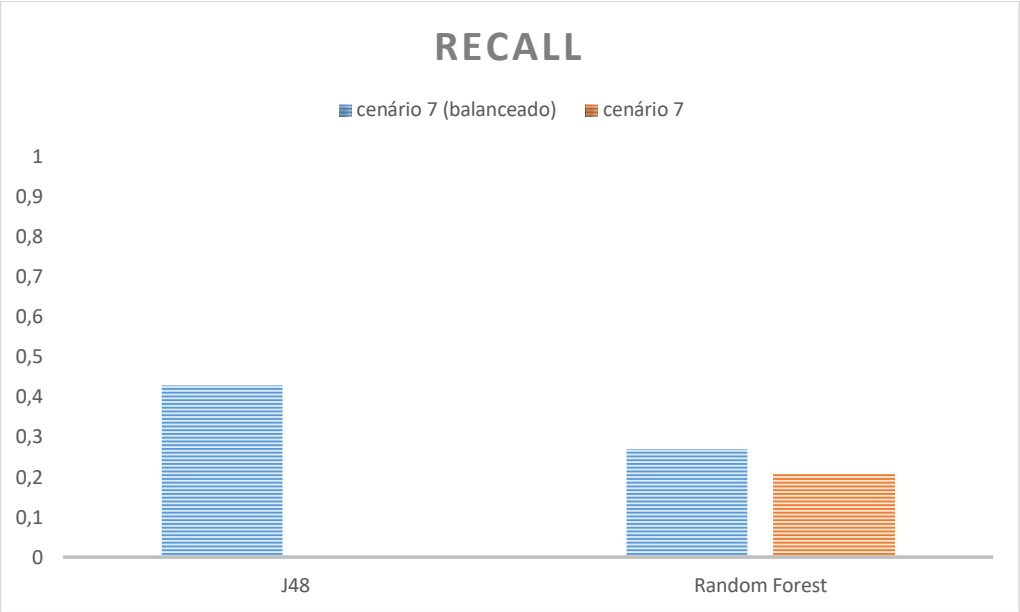

Figura 5.13: *Recall* dos modelos do paradigma simbólico no cenário 7 antes e depois do balanceamento de classes

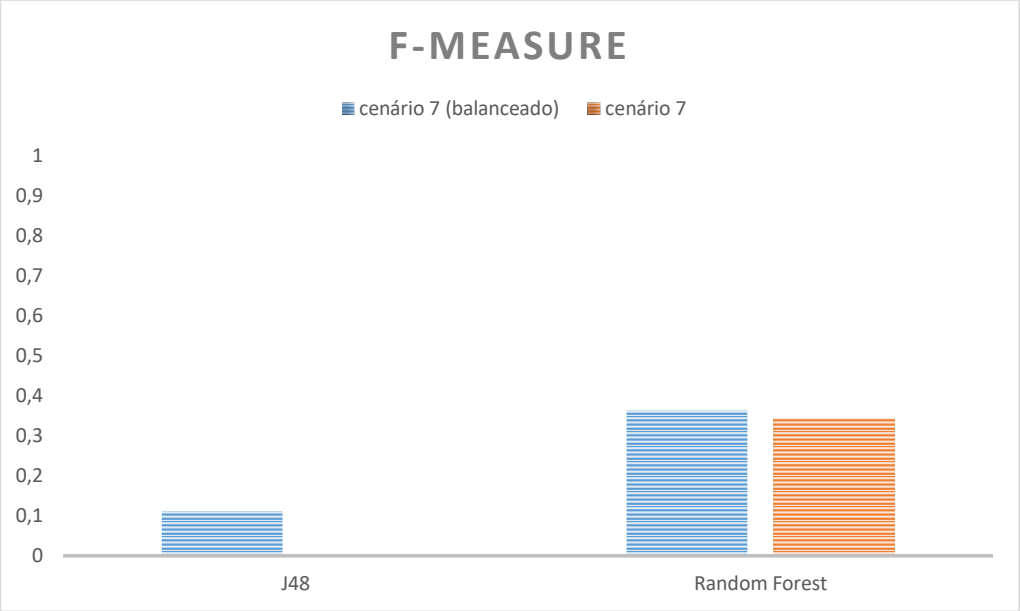

Figura 5.14: *F-measure* dos modelos do paradigma simbólico no cenário 7 antes e depois do balanceamento de classes

6. Conclusão e Trabalhos Futuros

O número de incidentes de segurança na Internet vem aumentando a cada dia, assim como a dependência nos serviços por meio dela oferecidos. Nesse contexto, as *botnets* surgem como fontes geradoras de diversas atividades maliciosas. Detectar essas redes de máquinas "zumbis" é tarefa desafiadora, devido às características de suas arquiteturas e funcionamento.

Diante disso, este trabalho cumpriu o objetivo de avaliar e comparar os modelos de diversos paradigmas de aprendizado de máquina para a detecção de diferentes tipos de *botnets* em cenários de atividades maliciosas, tanto na forma reativa quanto na preventiva. Tal se deu por meio de características extraídas dos fluxos de tráfego, sem a inspeção do conteúdo de dados (*payload*) dos pacotes. Na metodologia utilizada, primeiro foram levantadas características do tráfego de rede utilizadas em trabalhos anteriores e selecionadas as relevantes à detecção de *botnets* através de técnicas de seleção de atributos. Foram realizados o ranqueamento de atributos com as métricas *gain ratio* e incerteza simétrica, foi utilizada a técnica *correlation-based feature selection* e a abordagem *wrapper* com o método de busca *greedy*, aplicando *forward selection*. No *wrapper*, foi utilizado o algoritmo Árvore de Decisão J48 para fins de verificação dos efeitos de classificação. Ainda na abordagem *wrapper*, foi usado também o método de busca adotando algoritmo genético. Após isso, essas características alimentaram modelos de aprendizado de máquina supervisionado conforme os seguintes paradigmas: simbólico (Árvore de Decisão J48 e *Random Forest*), estatístico (Rede Bayesiana e *Support Vector Machines*), conexionista (Redes Neurais Artificiais Multicamadas do tipo Perceptron) e baseado em exemplos (*K-Nearest Neighbor*). No aprendizado não supervisionado, avaliou-se o algoritmo de *Clustering KMeans*. Além disso, o algoritmo *Random Forest* também representa a técnica de *Ensemble Learning* do tipo randomização. Esses modelos de aprendizado de máquina foram avaliados com métricas recomendadas para cenários

desbalanceados.

Como contribuições, destacam-se primeiramente uma seleção de características do tráfego de rede relevantes para detecção de *botnets*, de arquitetura centralizada e descentralizada, geradas por diversas famílias de *bots* (Neris, Rbot, Virut, Menti, Shogou, Murlo e NSIS.ay). Esta detecção considera tanto a fase de execução das diferentes atividades maliciosas como também a fase pré-ataque, onde os *bots* realizam apenas sincronização. Com relação aos trabalhos [2, 3, 11, 70] ratificaram-se a seguinte seleção de características: razão entre o número de pacotes de entrada com os de saída do fluxo (*RazaoBytesOrigDest*) [11], média de bits/s (*MedBitsSecond*) [2, 3, 11], duração do fluxo (*Dur*) [2, 3, 11], média do tamanho do pacote de um fluxo (*MedPktSize*) [2, 3, 11, 70] e o total de *bytes* transmitidos no fluxo (*TotBytes*), selecionado nos trabalhos [5, 70]. A quantidade de *bytes* enviadas pela origem no fluxo (*SrcBytes*) e a média do tamanho do pacote (*MedPktSize*) aparecem, no presente trabalho, na maioria dos cenários como contribuições de novas características do tráfego de rede relevantes para detectar tráfego de *botnets*. O desvio-padrão do tamanho do pacote em uma janela de tempo de 180s (*DPPkt*) e o desvio-padrão da duração do fluxo em uma janela de tempo de 180s (*DpDur*) também foram selecionadas apenas neste trabalho e contribuem para cenários específicos.

Outra contribuição existente foi na extensa análise comparativa do desempenho dos algoritmos. Os algoritmos do paradigma simbólico obtiveram o melhor desempenho em todas as análises, tendo o *Random Forest* conseguido desempenho um pouco superior. O paradigma simbólico foi o que melhor detectou as *botnets* de forma reativa e preventiva, independente de *malware*, protocolo, atividade maliciosa, arquitetura e quantidade de fluxos. Além disso, o desbalanceamento das classes também não interferiu no desempenho dos modelos. No cenário 7, todos os algoritmos obtiveram baixa performance na detecção do tráfego de rede gerado pela *botnet Sogou*. Como esse cenário é o mais desbalanceado, buscou-se avaliar se esse era o motivo do fraco desempenho dos algoritmos do paradigma simbólico. Foi então realizado um balanceamento das classes para averiguação, porém essa técnica não elevou de forma considerável o desempenho desses algoritmos, indicando que o motivo do baixo desempenho nesse cenário pode ser devido à dificuldade de detecção do tráfego gerado pelo *bot Sogou*. Os modelos gerados pelo *Random Forest* conseguiram melhor desempenho que os modelos gerados pelo algoritmo melhor avaliado em [2] e [3], que foi o K-NN. Os piores desempenhos foram dos modelos gerados pelos algoritmos RNA, SVM e *KMeans*, mesmo após a otimização dos algoritmos e o balanceamento de classes para o RNA e o SVM.

A partir dos experimentos realizados, conclui-se que as características do tráfego de rede que se destacaram, em conjunto com os algoritmos de aprendizado de máquina do paradigma simbólico, especialmente o *Random Forest*, são os mais indicados para serem usados em uma arquitetura de um IDS real para a detecção de *botnets*.

Como objetivos para trabalhos futuros, pretende-se analisar outros paradigmas e algoritmos de aprendizado de máquina, assim como avaliar novos atributos estatísticos, como a entropia. Também é de especial interesse para este trabalho estudar a quantidade de dados necessária para o adequado treinamento dos algoritmos, além de realizar novos experimentos onde o algoritmo é treinado em um cenário e testado em outro, assim como treinar e testar esses algoritmos em uma junção de todos os cenários avaliados. Por fim, deseja-se implementar, em uma arquitetura real, o modelo melhor avaliado nesses experimentos.

Referências Bibliográficas

[1] R. Moraes, J. F. Valiati, and W. P. G. Neto, "Document-level sentiment classification: An empirical comparison between svm and ann," *Expert Systems with Applications*, vol. 40, no. 2, pp. 621–633, 2013.

[2] D. C. A. d. Silva, "Análise da eficiência dos algoritmos de aprendizado de máquinas para a detecção de botnets," Master's thesis, Instituto Militar de Engenharia, 2016.

[3] D. C. Silva, S. S. Silva, and R. M. Salles, "Metodologia de detecção de botnets utilizando aprendizado de máquina," *XXXV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*, 2017.

[4] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east," *IDC iView: IDC Analyze the future*, vol. 2007, pp. 1–16, 2012.

[5] S. S. Silva, R. M. Silva, R. C. Pinto, and R. M. Salles, "Botnets: A survey," *Computer Networks*, vol. 57, no. 2, pp. 378–403, 2013.

[6] B. CERT, "Incidentes reportados ao cert.br, janeiro a dezembro de 2016." `https://www.cert.br/stats/incidentes/2016-jan-dec/analise.html`. Acessado em 28/10/2017.

[7] B. CERT, "Incidentes reportados ao cert.br, janeiro a dezembro de 2017." `https://www.cert.br/stats/incidentes/2017-jan-dec/analise.html`. Acessado em 20/03/2018.

[8] N. Arbor, "12º relatório de segurança de infraestrutura mundial anual." `http://br.arbornetworks.com/visibilidade-de-redes/`. Acessado em 28/10/2017.

[9] G. Thornton, "The global impact of cyber crime," *Grant Thornton International Business Report*, 2017.

[10] M. Ficco and M. Rak, "Stealthy denial of service strategy in cloud computing," *IEEE transactions on cloud computing*, vol. 3, no. 1, pp. 80–94, 2015.

[11] E. B. Beigi, H. H. Jazi, N. Stakhanova, and A. A. Ghorbani, "Towards effective feature selection in machine learning-based botnet detection approaches," in *Communications and Network Security (CNS), 2014 IEEE Conference on*, pp. 247–255, IEEE, 2014.

[12] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *computers & security*, vol. 45, pp. 100–123, 2014.

[13] S. Agrawal and J. Agrawal, "Survey on anomaly detection using data mining techniques," *Procedia Computer Science*, vol. 60, pp. 708–713, 2015.

[14] F. C. Freiling, T. Holz, and G. Wicherski, "Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks," in *European Symposium on Research in Computer Security*, pp. 319–335, Springer, 2005.

[15] P. Clay, "A modern threat response framework," *Network Security*, vol. 2015, no. 4, pp. 5–10, 2015.

[16] I. Ponemon, "Ibm (2015). 2015 cost of data breach study: Global analysis," 2015.

[17] M. Stevanovic and J. M. Pedersen, "Machine learning for identifying botnet network traffic," 2013.

[18] T. S. Hyslip and J. M. Pittman, "A survey of botnet detection techniques by command and control infrastructure," *Journal of Digital Forensics, Security and Law*, vol. 10, no. 1, p. 2, 2015.

[19] S. Miller and C. Busby-Earle, "The impact of different botnet flow feature subsets on prediction accuracy using supervised and unsupervised learning methods," 2016.

[20] M. Feily, A. Shahrestani, and S. Ramadass, "A survey of botnet and botnet detection," in *Emerging Security Information, Systems and Technologies, 2009. SECURWARE'09. Third International Conference on*, pp. 268–273, IEEE, 2009.

[21] D. Plohmann and E. Gerhards-Padilla, "Case study of the miner botnet," in *Cyber Conflict (CYCON), 2012 4th International Conference on*, pp. 1–16, IEEE, 2012.

[22] C. Kalt, "Internet relay chat: Architecture," 2000.

[23] E. Cooke, F. Jahanian, and D. McPherson, "The zombie roundup: Understanding, detecting, and disrupting botnets.," *SRUTI*, vol. 5, pp. 6–6, 2005.

[24] D. Dagon, G. Gu, C. P. Lee, and W. Lee, "A taxonomy of botnet structures," in *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pp. 325–339, IEEE, 2007.

[25] R. A. Rodríguez-Gómez, G. Maciá-Fernández, and P. García-Teodoro, "Survey and taxonomy of botnet research through life-cycle," *ACM Computing Surveys (CSUR)*, vol. 45, no. 4, p. 45, 2013.

[26] I. V. Popov, S. K. Debray, and G. R. Andrews, "Binary obfuscation using signals.," in *USENIX Security Symposium*, pp. 275–290, 2007.

[27] A. Nappa, A. Fattori, M. Balduzzi, M. Dell'Amico, and L. Cavallaro, "Take a deep breath: A stealthy, resilient and cost-effective botnet using skype.," in *DIMVA*, pp. 81–100, Springer, 2010.

[28] U. CERT, "Malware tunneling in ipv6." `https://www.us-cert.gov/sites/default/files/publications/IPv6Malware-Tunneling.pdf`. Acessado em 12/11/2017.

[29] P. Porras, H. Saidi, and V. Yegneswaran, "A multi-perspective analysis of the storm (peacomm) worm," tech. rep., Technical report, Computer Science Laboratory, SRI International, 2007.

[30] C. Li, W. Jiang, and X. Zou, "Botnet: Survey and case study," in *innovative computing, information and control (icicic), 2009 fourth international conference on*, pp. 1184–1187, IEEE, 2009.

[31] I. Ghafir, J. Svoboda, and V. Prenosil, "A survey on botnet command and control traffic detection," *International Journal of Advances in Computer Networks and its security (ICJNS)*, vol. 5, no. 1, 2015.

[32] C.-Y. Liu, C.-H. Peng, and I.-C. Lin, "A survey of botnet architecture and batnet detection techniques," *International Journal of Network Security*, vol. 16, no. 2, pp. 81–89, 2014.

[33] G. V. Cormack *et al.*, "Email spam filtering: A systematic review," *Foundations and Trends® in Information Retrieval*, vol. 1, no. 4, pp. 335–455, 2008.

[34] A. van der Merwe, M. Loock, and M. Dabrowski, "Characteristics and responsibilities involved in a phishing attack," in *Proceedings of the 4th international symposium on Information and communication technologies*, pp. 249–254, Trinity College Dublin, 2005.

[35] K. C. Wilbur and Y. Zhu, "Click fraud," *Marketing Science*, vol. 28, no. 2, pp. 293–308, 2009.

[36] N. Daswani and M. Stoppelman, "The anatomy of clickbot. a," in *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pp. 11–11, USENIX Association, 2007.

[37] H. R. Zeidanloo, M. J. Z. Shooshtari, P. V. Amoli, M. Safari, and M. Zamani, "A taxonomy of botnet detection techniques," in *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, vol. 2, pp. 158–162, IEEE, 2010.

[38] S. Kumar, R. Sehgal, P. Singh, and A. Chaudhary, "Nepenthes honeypots based botnet detection," *arXiv preprint arXiv:1303.3071*, 2013.

[39] C. Hoepers, K. S. Jessen, and M. Chaves, "Honeypots e honeynets: Definições e aplicações," *Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil, ver*, vol. 1, 2007.

[40] C. C. Zou and R. Cunningham, "Honeypot-aware advanced botnet construction and maintenance," in *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pp. 199–208, IEEE, 2006.

[41] T. Micro, "Taxonomy of botnet threats," *Whitepaper, November*, 2006.

[42] G. Fedynyshyn, M. C. Chuah, and G. Tan, "Detection and classification of different botnet c&c channels," in *International Conference on Autonomic and Trusted Computing*, pp. 228–242, Springer, 2011.

[43] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in cloud," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 42–57, 2013.

[44] K. Xu, D. Yao, Q. Ma, and A. Crowell, "Detecting infection onset with behavior-based policies," in *Network and System Security (NSS), 2011 5th International Conference on*, pp. 57–64, IEEE, 2011.

[45] S. García, A. Zunino, and M. Campo, "Survey on network-based botnet detection methods," *Security and Communication Networks*, vol. 7, no. 5, pp. 878–903, 2014.

[46] Y. Kugisaki, Y. Kasahara, Y. Hori, and K. Sakurai, "Bot detection based on traffic analysis," in *Intelligent Pervasive Computing, 2007. IPC. The 2007 International Conference on*, pp. 303–306, IEEE, 2007.

[47] T. PANG-NING, M. STEINBACH, and V. KUMAR, "Introdução ao "data mining"-mineração de dados," *Rio de Janeiro: Editora Ciência Moderna*, 2009.

[48] D. Moore, C. Shannon, G. M. Voelker, and S. Savage, "Internet quarantine: Requirements for containing self-propagating code," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 3, pp. 1901–1910, IEEE, 2003.

[49] M. C. Monard and J. A. Baranauskas, "Conceitos sobre aprendizado de máquina," *Sistemas inteligentes-Fundamentos e aplicações*, vol. 1, no. 1, p. 32, 2003.

[50] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*. MIT press, 2012.

[51] M. Hall, I. Witten, and E. Frank, "Data mining: Practical machine learning tools and techniques," *Kaufmann, Burlington*, 2011.

[52] Y. Sun, M. S. Kamel, A. K. Wong, and Y. Wang, "Cost-sensitive boosting for classification of imbalanced data," *Pattern Recognition*, vol. 40, no. 12, pp. 3358–3378, 2007.

[53] H. Liu and H. Motoda, "Less is more," in *Computational Methods of Feature Selection*, pp. 16–31, Chapman and Hall/CRC, 2007.

[54] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[55] M. A. Hall, *Correlation-based feature selection for machine learning*. PhD thesis, University of Waikato Hamilton, 1999.

[56] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[57] R. L. Stange, *Adaptatividade em aprendizagem de máquina: conceitos e estudo de caso.* PhD thesis, Universidade de São Paulo, 2011.

[58] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[59] J. Pearl, "Bayesian networks," *Department of Statistics, UCLA*, 2011.

[60] R. E. Neapolitan *et al.*, *Learning bayesian networks*, vol. 38. Pearson Prentice Hall Upper Saddle River, NJ, 2004.

[61] V. Vapnik, *The nature of statistical learning theory.* Springer science & business media, 2013.

[62] A. J. Smola, *Advances in large margin classifiers.* MIT press, 2000.

[63] S. Haykin, *Redes neurais: princípios e prática.* Bookman Editora, 2007.

[64] H. Wang, "Nearest neighbors by neighborhood counting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 6, pp. 942–953, 2006.

[65] R. Linden, "Técnicas de agrupamento," *Revista de Sistemas de Informação da FSMA*, vol. 4, pp. 18–36, 2009.

[66] L. Bottou and Y. Bengio, "Convergence properties of the k-means algorithms," in *Advances in neural information processing systems*, pp. 585–592, 1995.

[67] A. Zand, G. Vigna, X. Yan, and C. Kruegel, "Extracting probable command and control signatures for detecting botnets," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pp. 1657–1662, ACM, 2014.

[68] F. Haddadi and A. N. Zincir-Heywood, "Botnet detection system analysis on the effect of botnet evolution and feature representation," in *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 893–900, ACM, 2015.

[69] E. d. C. d. Silva, "Detecção de ataques de negação de serviço utilizando aprendizado de máquina," Master's thesis, Universidade Federal do Estado do Rio de Janeiro, 2016.

[70] F. V. Alejandre, N. C. Cortés, and E. A. Anaya, "Feature selection to detect botnets using machine learning algorithms," in *Electronics, Communications and Computers (CONIELECOMP), 2017 International Conference on*, pp. 1–7, IEEE, 2017.

[71] G. Kirubavathi and R. Anitha, "Botnet detection via mining of traffic flow characteristics," *Computers & Electrical Engineering*, vol. 50, pp. 91–101, 2016.

[72] D. E. Goldberg, "Genetic algorithms in search, optimization, and machine learning, 1989," *Reading: Addison-Wesley*, 1989.

[73] F. Azuaje, "Witten ih, frank e: Data mining: Practical machine learning tools and techniques 2nd edition," *BioMedical Engineering OnLine*, vol. 5, no. 1, p. 1, 2006.

[74] C. Phua, D. Alahakoon, and V. Lee, "Minority report in fraud detection: classification of skewed data," *Acm sigkdd explorations newsletter*, vol. 6, no. 1, pp. 50–59, 2004.

[75] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[76] E. Frank, M. A. Hall, and I. H. Witten, *The WEKA Workbench. Online Appendix for Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[77] B. Claise, "Cisco systems netflow services export version 9," 2004.

[78] R. Kohavi, "Wrappers for performance enhancement and oblivious decision graphs," tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1995.

[79] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

A. *Scripts* utilizados para pré-processamento

```
#Script para ler datasets do ctu-13 e realizar parte 1 do pre-
    processamento. Elimina fluxos com duracao zero; calcula as features:
    MedPktSize, MedPktSecond, MedBitsSecond, RazaoBytesOrigDest; converte
    StartTime para timestamp e padroniza o label

from __future__ import division
from datetime import datetime, timedelta
import re

try:

    input_file = open("/home/.../cen.binetflow")
    output_file = open ("/home/.../cen.binetflow2.csv","w")

except:

    print ("Nao foi possivel abrir o arquivo para leitura.")

skip_header = 0

for linha in input_file:

   input_list = linha.split(",")

   #verifica o cabecalho e escreve no novo arquivo
   if skip_header == 0:

      #escrever cabecalho com dv
```

```
        input_list[14:14] = ["MedPktSize", "MedPktSecond", "MedBitsSecond", "
            RazaoBytesOrigDest","DPPkt","DpDur","label"]
        a = ",".join(input_list)
        output_file.write(a)
        skip_header += 1

    #verifica se a duracao eh zero para nao dividir por zero (dur), caso
        positivo, pular essa linha
    elif input_list[1] == "0.000000":
        continue

    #a partir da segunda linha
    else:

        #calcula MedPktSize e insere (total de bytes / total de pacotes),
            arrendondando 6 casas decimais
        MedPktSize = "%.6f" % (float(input_list[12]) / float(input_list[11]))

        input_list[14:14]= [str(MedPktSize)]

        #calcula MedPktSecond e insere (total de pacotes / duracao),
            arrendondando 6 casas decimais
        MedPktSecond = "%.6f" % (float(input_list[11]) / float(input_list[1])
            )
        input_list[15:15]= [str(MedPktSecond)]

        #calcula MedBitsSecond e insere (total de bytes * 8 / duracao),
            arrendondando 6 casas decimais
        totalbits = float(input_list[12]) * 8
        MedBitsSecond = "%.6f" % (totalbits/float(input_list[1]))
        input_list[16:16]= [str(MedBitsSecond)]

        #calcula RazaoBytesOrigDest(Razao entre numero de bytes transmitidos
            pela origem e destino por fluxo) e insere, arrendondando 6 casas
            decimais
        #dstbytes = (totbytes - srcbytes) + 1 [para os casos que dstbytes =
            0]
        dstbytes = (float(input_list[12]) - float(input_list[13])) + 1
        #RazaoBytesOrigDest = srcbytes/dstbytes
        RazaoBytesOrigDest = "%.6f" % (float(input_list[13])/dstbytes)
```

```
      input_list[17:17]= [str(RazaoBytesOrigDest)]

      # Converte StartTime para posix
      epoch = datetime(1970,1,1)
      d_aux = input_list[0].split()
      date = d_aux[0].split("/")
      time = d_aux[1].split(":")
               td = datetime(int(date[0]), int(date[1]), int(date[2]), int(
                   time[0]), int(time[1]), int(float(time[2]))) - epoch
               input_list[0] = str(td.total_seconds())

        #procura a palavra botnet no meio do label e substitui por botnet
      teste_botnet = re.search('Botnet',input_list[18])
      if (teste_botnet != None):

         input_list[18] = "Botnet"

      else:
         input_list[18] = "Normal"

      a = ",".join(input_list)
      output_file.write(a + '\n')

input_file.close()
output_file.close()
```

```
#Segundo script para ler datasets do ctu-13, calcular os desvios-padroes
    do tamanho do pacote e da duracao dos fluxos em uma janela de tempo de
     180s.

from __future__ import division
from datetime import datetime, timedelta
import numpy as np
import pandas as pd
import re

arquivo = '/home/.../cen.binetflow2.csv'

try:
```

```
    input_file = open(arquivo)
    output_file = open ("/.../cen.binetflow-final.csv","w")

except:

    print ("Nao foi possivel abrir o arquivo para leitura.")

skip_header = 0
pula_cabecalho = 0

file_lines = input_file.readlines()
last_line = file_lines[len(file_lines)-1]
last_line = last_line.split(",")

#ultimo starttime da ultima linha do arquivo
ultimoStartTime = last_line[0]
#pega starttime da primeira linha e coloca como janelainicio
janelainicio = file_lines[1]
janelainicio = janelainicio.split(",")

input_file.close()
input_file = open(arquivo)

JanelaMedPktSize = []
JanelaDurFluxos = []
dppkt_Janelas = []
dpdur_Janelas = []

janelacorrente = 0
#faz um for para pegar cada linha o numero de vezes que tiver de janelas
    no arquivo
for linha in input_file:

    #pula cabecalho
   if skip_header == 0:

     skip_header += 1
   else:
     input_list = linha.split(",")
```

```
#adicionando em um vetor o medpktsize dos fluxos
JanelaMedPktSize.append(float(input_list[14]))

#adicionando em um vetor a duracao dos fluxos
JanelaDurFluxos.append(float(input_list[1]))

#janelacorrente recebe o valor de StartTime
janelacorrente = input_list[0]

#testa se os fluxos estao dentro da janela de 180s
if (float(janelacorrente) - float(janelainicio[0])) > 180:
   #calcular desvio padrao da MedPktSize de uma janela de 180s
   panda_series_pkt = pd.Series(JanelaMedPktSize)
   dppkt = panda_series_pkt.std()
   #testa dp e muda para 0
   if (dppkt != dppkt):
      dppkt = 0

   #calcular desvio padrao da dur dos fluxos em uma janela de 180s
   panda_series_dur = pd.Series(JanelaDurFluxos)
   dpdur = panda_series_dur.std()
   #testa dp e muda para 0
   if (dpdur != dpdur):
      dpdur = 0

   #coloca todos os dps dos pacotes das janelas em um vetor para
       depois add o atributo no dataset
   dppkt_Janelas.append(float(dppkt))

   #coloca todos os dps dos pacotes das janelas em um vetor para
       depois add o atributo no dataset
   dpdur_Janelas.append(float(dpdur))

   #Quando acaba a janela de 180s janela de inicio recebe a janela
       corrente
   janelainicio[0] = janelacorrente

   #zera o vetor JanelaMedPktSize para calcular na nova janela
   del JanelaMedPktSize[:]
```

```
        #zera o vetor JanelaDurFluxos para calcular na nova janela
        del JanelaDurFluxos[:]

#calculas os desvios nos fluxos que ficaram fora das janelas
#calcular desvio padrao da MedPktSize
panda_series_pkt = pd.Series(JanelaMedPktSize)
dppkt = panda_series_pkt.std()
#testa dp e muda para 0
if (dppkt != dppkt):
   dppkt = 0

#calcular desvio padrao da dur dos fluxos
panda_series_dur = pd.Series(JanelaDurFluxos)
dpdur = panda_series_dur.std()
#testa dp e muda para 0
if (dpdur != dpdur):
   dpdur = 0

#coloca todos os dps dos pacotes das janelas em um vetor para depois
    adicionar o atributo no dataset
dppkt_Janelas.append(float(dppkt))

#coloca todos os dps dos pacotes das janelas em um vetor para depois
    adicionar o atributo no dataset
dpdur_Janelas.append(float(dpdur))

#Quando acaba a janela de 180s janela de inicio recebe a janela corrente
janelainicio[0] = janelacorrente

#zera o vetor JanelaMedPktSize para calcular na nova janela
del JanelaMedPktSize[:]

#zera o vetor JanelaDurFluxos para calcular na nova janela
del JanelaDurFluxos[:]


##########################################SEGUNDA PARTE GRAVAR NO ARQUIVO OS
    DV ########################################################
#fecha e abre o arquivo
input_file.close()
```

```
input_file = open(arquivo)

#voltando janelainicio para o primeiro StartTime do dataset
janelainicio = file_lines[1]
janelainicio = janelainicio.split(",")

a = 0
skip_header = 0

for linha in input_file:

    #pula cabecalho
   if skip_header == 0:
      #gravando cabecalho
      header = ("Dur","TotPkts","TotBytes","SrcBytes","MedPktSize","
          MedPktSecond","MedBitsSecond","RazaoBytesOrigDest","DPPkt","DpDur
          ","label" + '\n')
      header2 = ",".join(header)
      output_file.write(header2)

      skip_header += 1
   else:
      input_list = linha.split(",")

      #Escrevendo o novo atributo do DP

      input_list[18:18]= [str(dppkt_Janelas[a])]
      input_list[19:19]= [str(dpdur_Janelas[a])]
      output_file.write(input_list[1]+","+input_list[11]+","+input_list
          [12]+","+input_list[13]+","+input_list[14]+","+input_list[15]+","
          +input_list[16]+","+input_list[17]+","+input_list[18]+","+
          input_list[19] + ","+input_list[20])

      #janelacorrente recebe o valor de StartTime
      janelacorrente = input_list[0]

      #testa se os fluxos estao dentro da janela de 180s
      if (float(janelacorrente) - float(janelainicio[0])) > 180:
         a = a+1
         #Quando acaba a janela de 180s janela de inicio recebe a janela
```

```
            corrente
        janelainicio[0] = janelacorrente

input_file.close()
output_file.close()
```

B. Resultados Das Métricas

As Tabelas B.1, B.2, B.3 e B.4 apresentam os resultados das métricas Precisão, *Recall*, *F-Measure* e Tempo de Treinamento dos modelos gerados pelos algoritmos nos treze cenários.

As Tabelas B.5, B.6 e B.7 expõe os resultados das métricas Precisão, *Recall* e *F-Measure* dos modelos gerados após otimização dos algoritmos RNA, SVM e *Kmeans* nos treze cenários.

As Tabelas B.8, B.9 e B.10 mostram os resultados das métricas Precisão, *Recall* e *F-Measure* dos modelos gerados após balanceamento de classes para os algoritmos RNA, SVM nos treze cenários.

As Tabelas B.11, B.12 e B.13 mostram os resultados das métricas Precisão, *Recall* e *F-Measure* dos modelos após balanceamento de classes para os algoritmos Árvore de Decisão J48 e *Random Forest* no cenário 7.

A Tabela B.14 mostra o resultado da métrica *F-Measure* dos modelos gerados pelo algoritmo *Random forest*, que apresentou o melhor desempenho deste trabalho, comparado com os modelos gerados pelo algoritmo K-NN, que apresentou o melhor desempenho no trabalho [2] [3].

Tabela B.1: Precisão dos modelos nos cenários

	Precisão nos cenários												
Algoritmos	1	2	3	4	5	6	7	8	9	10	11	12	13
Árvore de Decisão J48	0,878	0,852	0,909	0,648	0,785	0,968	0	0,979	0,864	0,995	0,997	0,85	0,947
Random Forest	0,909	0,862	0,868	0,648	0,751	0,972	1	0,981	0,895	0,99	0,989	0,936	0,943
Rede Bayesiana	0,553	0,687	0,63	0,182	0,229	0,82	0,197	0,838	0,693	0,998	1	0,428	0,834
SVM	0	0	0	0	0	0	0	0	0	0	0	0	0
RNA	0	0	0,49	0	0	0	0	0	0	0,669	0,603	0	0
K-NN	0,553	0,752	0,865	0,603	0,505	0,951	0,382	0,805	0,57	0,759	0,978	0,516	0,8
Kmeans	0,006	0	0	0,002	0,011	0	0	0	0,151	0,027	0,031	0,012	0,001

Tabela B.2: *Recall* dos modelos nos cenários

	Recall nos cenários												
Algoritmos	1	2	3	4	5	6	7	8	9	10	11	12	13
Árvore de Decisão J48	0,805	0,845	0,909	0,614	0,436	0,99	0	0,835	0,878	0,988	0,971	0,666	0,882
Random Forest	0,823	0,796	0,886	0,614	0,589	0,969	0,206	0,862	0,88	0,99	0,979	0,709	0,883
Rede Bayesiana	0,862	0,812	0,991	0,899	0,751	0,978	0,238	0,806	0,859	0,977	0,971	0,646	0,819
SVM	0	0	0	0	0	0	0	0	0	0	0	0	0
RNA	0	0	0,363	0	0	0	0	0	0	0,932	0,796	0	0
K-NN	0,578	0,752	0,858	0,586	0,488	0,958	0,333	0,8	0,603	0,769	0,975	0,528	0,792
Kmeans	0,065	0,008	0,001	0,176	0,1	0,002	0,079	0	0,366	0,293	0,29	0,705	0,005

Tabela B.3: *F-Measure* dos modelos nos cenários

	F-Measure nos cenários												
algoritmo	1	2	3	4	5	6	7	8	9	10	11	12	13
Árvore de Decisão J48	0,84	0,848	0,909	0,63	0,561	0,979	0	0,901	0,871	0,991	0,984	0,747	0,914
Random Forest	0,864	0,828	0,877	0,63	0,66	0,97	0,342	0,918	0,887	0,99	0,984	0,807	0,912
Rede Bayesiana	0,674	0,744	0,77	0,303	0,351	0,892	0,216	0,822	0,767	0,987	0,985	0,515	0,826
SVM	0	0	0	0	0	0	0	0	0	0	0	0	0
RNA	0	0	0,417	0	0	0	0	0	0	0,779	0,686	0	0
K-NN	0,565	0,752	0,862	0,595	0,497	0,955	0,356	0,802	0,586	0,764	0,977	0,522	0,796
Kmeans	0,011	0,001	0	0,004	0,02	0	0	0	0,214	0,049	0,057	0,024	0,002

Tabela B.4: Tempo de treinamento dos modelos nos cenários

	Tempo de treinamento (s) nos cenários												
Algoritmos	1	2	3	4	5	6	7	8	9	10	11	12	13
Árvore de Decisão J48	340,85	72,25	84,12	626,87	2,18	10,43	0,95	213,45	632,25	28,15	2,13	12,01	175,21
Random Forest	5670,98	1366,88	22148,97	626,87	52,51	242,18	25,98	2166,36	1868,32	611,27	35,9	178,82	1477,58
Rede Bayesiana	66,58	31,46	70,32	8,99	1,49	8,16	0,8	44,06	78,62	14,34	0,99	5,17	45,65
SVM	35906,72	29939,62	88592,2	945,92	162,06	3007,88	11,69	10607,79	12546,49	24946,46	120,57	1791,35	52949,06
RNA	1238,72	898,09	2589,59	530,09	69,53	349,95	64,92	1843,28	1680,01	551,93	59,89	325,25	1197,8
K-NN	0,14	0,19	0,2	0,11	0,08	0,07	0,17	0,24	0,17	0,07	0,01	0,05	0,22
Kmeans	78,49	8,71	19,33	4,74	1,09	2,96	0,55	22,02	12,85	4,82	2,07	2,19	5,85

Tabela B.5: Precisão dos modelos nos cenários após otimização dos algoritmos

	Precisão nos cenários												
Algoritmos	1	2	3	4	5	6	7	8	9	10	11	12	13
SVM	0	0	0	0	0	0	0	0	0	0	1	0	0
RNA	0	0	0,504	0	0	0	0	0	0	0,67	0,63	0	0
Kmeans	0,017	0	0,013	0,001	0,012	0	0,001	0	0,131	0,061	0,029	0,011	0,039

Tabela B.6: *Recall* dos modelos nos cenários após otimização dos algoritmos

	Recall nos cenários												
algoritmo	1	2	3	4	5	6	7	8	9	10	11	12	13
SVM	0	0	0	0	0	0	0	0	0	0	0,012	0	0
RNA	0	0	0,362	0	0	0	0	0	0	0,925	0,815	0	0
Kmeans	0,169	0,009	0,3	0,245	0,254	0	0,75	0,071	0,305	0,779	0,69	0,645	0,558

Tabela B.7: *F-Measure* dos modelos nos cenários após otimização dos algoritmos

	F-Measure nos cenários												
Algoritmos	1	2	3	4	5	6	7	8	9	10	11	12	13
SVM	0	0	0	0	0	0	0	0	0	0	0,024	0	0
RNA	0	0	0,421	0	0	0	0	0	0	0,777	0,71	0	0
Kmeans	0,03	0,001	0,026	0,003	0,023	0	0,002	0	0,183	0,112	0,056	0,022	0,073

Tabela B.8: Precisão dos modelos nos cenários após o balanceamento de classes

	Precisão nos cenários												
Algoritmos	1	2	3	4	5	6	7	8	9	10	11	12	13
SVM	0,016	0,013	0,012	0,034	0,022	0,009	0,001	0,001	0,151	0,212	0,211	0,011	0,026
RNA	0,034	0,024	0,049	0,001	0,01	0,007	0,001	0	0,207	0,009	0,533	0,019	0,028

Tabela B.9: *Recall* dos modelos nos cenários após o balanceamento de classes

	Recall nos cenários												
Algoritmos	1	2	3	4	5	6	7	8	9	10	11	12	13
SVM	0,999	1	0,99	0,182	0,234	0,929	0,5	0,938	0,383	0,976	0,971	0,667	0,994
RNA	0,843	0,988	0,498	1	0,585	0,984	1	1	0,77	1	0,971	0,711	0,39

Tabela B.10: *F-Measure* dos modelos nos cenários após o balanceamento de classes

	F-Measure nos cenários												
Algoritmos	1	2	3	4	5	6	7	8	9	10	11	12	13
SVM	0,032	0,025	0,024	0,057	0,04	0,018	0,001	0,001	0,217	0,348	0,347	0,021	0,051
RNA	0,065	0,047	0,088	0,002	0,021	0,014	0,001	0,001	0,326	0,017	0,688	0,038	0,052

Tabela B.11: Precisão dos modelos do paradigma simbólico no cenário 7 antes e depois do balanceamento de classes

	Precisão nos cenários	
Algoritmos	cenário 7 (balanceado)	cenário 7
Árvore de Decisão J48	0,064	0
Random Forest	0,548	1

Tabela B.12: *Recall* dos modelos do paradigma simbólico no cenário 7 antes e depois do balanceamento de classes

	Recall nos cenários	
Algoritmos	cenário 7 (balanceado)	cenário 7
Árvore de Decisão J48	0,429	0
Random Forest	0,27	0,206

Tabela B.13: *F-Measure* dos modelos do paradigma simbólico no cenário 7 antes e depois do balanceamento de classes

	F-Measure nos cenários	
Algoritmos	cenário 7 (balanceado)	cenário 7
Árvore de Decisão J48	0,111	0
Random Forest	0,362	0,342

Tabela B.14: *F-Measure*

	F-Measure nos cenários												
Algoritmos	1	2	3	4	5	6	7	8	9	10	11	12	13
K-NN todas as características [2] [3]	0,713	0,799	0,662	0,563	0,538	0,972	0,11	0,783	0,512	0,754	0,98	0,608	0,832
K-NN características Relevantes [2] [3]	0,734	0,799	0,52	0,688	0,54	0,009	0,105	0,643	0,496	0,333	0,998	0,605	0,783
Random Forest	0,864	0,828	0,877	0,63	0,66	0,97	0,362	0,918	0,887	0,99	0,984	0,807	0,912