

UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"

FACULDADE DE CIÊNCIAS - CAMPUS BAURU

DEPARTAMENTO DE COMPUTAÇÃO

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LUIS FELIPE BUENO DA SILVA

**UTILIZAÇÃO DE TÉCNICAS DE MACHINE LEARNING PARA
DETECÇÃO DE BOTNETS**

BAURU

2018

LUIS FELIPE BUENO DA SILVA

UTILIZAÇÃO DE TÉCNICAS DE MACHINE LEARNING PARA DETECÇÃO DE BOTNETS

Trabalho de Conclusão de Curso do Curso
de Ciência da Computação da Universidade
Estadual Paulista “Júlio de Mesquita Filho”,
Faculdade de Ciências, Campus Bauru.
Orientador: Prof. Dr. Kelton Augusto Pontara
da Costa

BAURU
2018

Luis Felipe Bueno da Silva Utilização de técnicas de Machine Learning para Detecção de Botnets/ Luis Felipe Bueno da Silva. – Bauru, 2018- 54 p. : il. (algumas color.) ; 30 cm.
Orientador: Prof. Dr. Kelton Augusto Pontara da Costa
Trabalho de Conclusão de Curso – Universidade Estadual Paulista “Júlio de Mesquita Filho”
Faculdade de Ciências
Ciência da Computação, 2018.
1. Botnet 2. Machine Learning 3. Inteligência Artificial 4. Python

Luis Felipe Bueno da Silva

Utilização de técnicas de Machine Learning para Detecção de Botnets

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

**Prof. Dr. Kelton Augusto Pontara da
Costa**

Orientador

Universidade Estadual Paulista "Júlio de
Mesquita Filho"

Faculdade de Ciências

Departamento de Ciência da Computação

**Prof^a. Dr^a. Simone das Graças
Domingues Prado**

Universidade Estadual Paulista "Júlio de
Mesquita Filho"

Faculdade de Ciências

Departamento de Ciência da Computação

**Prof. Associado José Remo Ferreira
Brega**

Universidade Estadual Paulista "Júlio de
Mesquita Filho"

Faculdade de Ciências

Departamento de Ciência da Computação

Bauru, 13 de Novembro de 2018.

Agradecimentos

Agradeço minha família por todo o apoio e carinho, em especial a minha mãe Cristiane por sempre me incentivar nos estudos, acreditar em mim e em meu potencial, e todos os esforços em minha criação que permitiram com que eu chegasse até aqui. Agradeço também ao meu padrasto Milton por todos os conselhos e por estar sempre presente quando precisei.

Agradeço minha namorada Vitória por todo o amor e carinho, por estar ao meu lado em todos os momentos desses quatro anos, me apoiando e me aturando, mesmo com toda as dificuldades trazidas pela distância. Que estes anos quem passamos longe um do outro possam ser recompensados com muitos anos de união.

A todos os professores pelos conhecimentos transmitidos durante a graduação.

Ao meu orientador Kelton, por toda a ajuda durante a realização deste trabalho, desde a sua concepção até a conclusão.

Ao meu amigo Guilherme Dias, que foi essencial para que eu descobrisse a minha paixão pela Ciência da Computação.

A todos os amigos que fiz aqui em Bauru, pelo companheirismo, por me ajudarem sempre que precisei, e por todos os momentos vividos nestes quatro anos de graduação.

"The ending isn't any more important
than any of the moments leading to it"
(To the Moon)

Resumo

Esta monografia apresenta o estudo de métodos da área de *Machine Learning* aplicados para a área de detecção de Botnets, redes de computadores comprometidos que são controlados por um invasor com o fim de executar atividades como ataques *DDoS*, roubos de dados, entre outras ações maliciosas. Este trabalho é focado em estudar a eficiência dos classificadores mais utilizados em estudos anteriores da área, utilizando as técnicas de *Naive Bayes*, *Support Vector Machines*, Árvores de Decisão, Florestas Aleatórias, *AdaBoost*, e aplicar técnicas para seleção de características de rede mais relevantes na tarefa de seleção dos tráfegos de Botnet em um ambiente de rede, através de uma abordagem de força bruta e uma abordagem utilizando o algoritmo de seleção de características *Recursive Feature Elimination*. Busca também estudar a relevância de técnicas de otimização de hiper-parâmetros dos estimadores, com o objetivo de aumentar a acurácia. Por fim, são apresentadas conclusões com base nos resultados obtidos no estudo.

Palavras-chave: Botnet, Machine Learning, Inteligência Artificial, Python.

Abstract

This monograph presents the study of Machine Learning methods applied to the detection of Botnets, compromised computer networks that are controlled by an attacker in order to perform malicious activities such as DDoS attacks, data theft, among others. This work is focused on studying the efficiency of the most used classifiers in previous studies of the aream with the application of Naive Bayes, Support Vector Machines, Decision Trees, Random Forests and AdaBoost models, and apply techniques to select the most relevant network characteristics in the task of selecting botnet traffic in a network environment, through a brute-force approach and using the Recursive Feature Elimination algorithm. It also seeks to study the relevance of optimization techniques on estimators hyper-parameters, in order to increase model accuracy. Finally, conclusions are drawn based on the results obtained in the study.

Keywords: Botnet, Machine Learning, Artificial Intelligence, Python.

Lista de figuras

Figura 1 – Arquiteturas de Botnet.	18
Figura 2 – Exemplo de Matriz de Confusão.	22
Figura 3 – Hiperplanos no conceito de SVM.	24
Figura 4 – Caso de uso de transformação em SVM.	25
Figura 5 – Exemplo de Árvore de Decisão.	26
Figura 6 – Funcionamento de Florestas Aleatórias.	28
Figura 7 – Funcionamento do AdaBoost.	28
Figura 8 – Exemplo de utilização do Scikit.	32
Figura 9 – Exemplo de utilização do Pandas.	34
Figura 10 – Exemplo de utilização do Matplotlib.	35
Figura 11 – Exemplo de utilização do Seaborn.	35
Figura 12 – Exemplo de utilização do IPython.	36
Figura 13 – Rotulação dos fluxos.	39
Figura 14 – Exemplo de geração de colunas.	39
Figura 15 – Gerando combinações com a biblioteca <i>itertools</i>	41
Figura 16 – Utilização do <i>Grid Search</i> no Scikit.	43
Figura 17 – Comparativo de Acurácias.	45
Figura 18 – Comparativo de Número de Características.	46
Figura 19 – Frequência de Características na abordagem de Força Bruta.	46
Figura 20 – Características mais frequentes na abordagem de RFE.	47
Figura 21 – Características menos frequentes na abordagem de RFE.	47
Figura 22 – Frequência de Características Históricas na abordagem de RFE.	48
Figura 23 – Comparativo de Acurácia dos Kernels do SVM.	48

Lista de quadros

Quadro 1 – Funções Kernel utilizadas em SVM.	25
Quadro 2 – Estudos da área utilizados como base para escolha dos classificadores. . .	29
Quadro 3 – Exemplo de geração de colunas.	40

Lista de tabelas

Tabela 1 – Resultados da abordagem de Força Bruta	44
Tabela 2 – Resultados da abordagem utilizando RFE	44
Tabela 3 – Resultados da otimização de Hiper-parâmetros do SVM	45

Lista de abreviaturas e siglas

CSV	<i>Comma-separated Values</i>
PCAP	<i>Packet Capture</i>
SVM	<i>Support Vector Machines</i>
RFE	<i>Recursive Feature Elimination</i>
IA	Inteligência Artificial
ML	<i>Machine Learning</i>
CV	<i>Cross Validation</i>
NB	<i>Naive Bayes</i>
UDP	<i>User Datagram Protocol</i>
TCP	<i>Transmission Control Protocol</i>
IAT	<i>Inter-Arrival Time</i>
P2P	<i>Peer to Peer</i>
C&C	Comando e Controle
HTTP	<i>Hypertext Transfer Protocol</i>
HTTPS	<i>Hypertext Transfer Protocol Secure</i>
IRC	<i>Internet Relay Chat</i>

Lista de símbolos

γ

Letra grega Gama

Sumário

1	INTRODUÇÃO	15
1.1	Objetivos	16
1.1.1	Objetivo Geral	16
1.1.2	Objetivos Específicos	16
1.1.3	Organização da monografia	16
2	FUNDAMENTAÇÃO TEÓRICA	17
2.1	Botnets	17
2.2	Machine Learning	19
2.2.1	Supervisionado e Não-supervisionado	19
2.2.2	Fases de treinamento e teste	20
2.2.3	Métricas	21
2.2.4	Algoritmos	22
2.2.4.1	Naive Bayes	22
2.2.4.2	Support Vector Machines	23
2.2.4.3	Árvores de Decisão	25
2.2.4.4	Florestas Aleatórias	27
2.2.4.5	AdaBoost	27
2.2.4.6	Recursive Feature Elimination	29
2.3	Revisão Bibliográfica	29
3	METODOLOGIA	32
3.1	Ferramentas	32
3.1.1	Scikit-learn	32
3.1.2	Numpy	33
3.1.3	Pandas	33
3.1.4	Matplotlib	34
3.1.5	Seaborn	34
3.1.6	Jupyter Notebook	35
4	DESENVOLVIMENTO	37
4.1	Base de dados	37
4.2	Geração de fluxos de rede	37
4.3	Processamento de dados	38
4.4	Execução dos algoritmos	40
4.4.1	Busca de força bruta com base em dados de pesquisas anteriores	41

4.4.2	Utilizando o algoritmo Recursive Feature Elimination	42
4.4.3	Otimização de Hiper-parâmetros	42
5	RESULTADOS	44
5.1	Performance dos Classificadores	44
5.2	Características	44
5.3	Hiper-parâmetros	45
6	CONCLUSÃO	49
6.1	Trabalhos Futuros	50
	REFERÊNCIAS	51

1 Introdução

Uma das ameaças mais graves na área de cibersegurança é o uso coordenado de uma grande quantidade de máquinas para ataques, as chamadas *botnets*. Uma *botnet* é uma rede de computadores infectados, que sem o consentimento de seus donos, têm seus recursos utilizados por um atacante (denominado *botmaster*) para a execução de atividades maliciosas (MILLER; BUSBY-EARLE, 2016b). O *botmaster* envia seus comandos de ataque para uma estrutura central, denominada Servidor de Comando e Controle, que é responsável por repassar a mensagem para todas as máquinas infectadas, e a partir desse momento as ações se iniciam.

Existem várias classificações de *botnets* de acordo com o protocolo utilizado pelo Servidor de C&C, podendo ser HTTP, HTTPS, e IRC. Além disso, existem botnets que não utilizam um servidor de C&C, trabalhando de maneira descentralizada através de protocolos P2P.

Detectar o tráfego de *botnets* é um grande desafio, pois como elas utilizam protocolos de aplicações já existentes, a detecção de sua presença na rede acaba sendo não-trivial, e a classificação do seu tráfego se torna ainda mais desafiadora devido ao uso de encriptação no conteúdo dos pacotes. (LU; GHORBANI, 2008)

Enquanto *honeypots* se provaram eficientes para se analisar o comportamento de um *botnet* e suas características de funcionamento, eles não são capazes de detectá-las. Tendo isso em vista, foram desenvolvidas algumas técnicas para detecção de *botnets* que podem ser classificadas como baseadas em técnicas de assinatura, anomalias, comportamento, ou rede (ASHA; HARSHA; SONIYA, 2016).

Apesar das *botnets* já existirem a muito tempo, o problema se torna muito mais grave com o avanço de tecnologias de Internet das Coisas (IoT). Muitos dispositivos de IoT são feitos com baixa segurança, o que os torna vulneráveis a ataque e consequentemente podem ser usados em botnets. Em 2016 foi descoberta a Mirai, uma botnet que após a infecção buscava na rede por dispositivos IoT que pudessem ser facilmente invadidos e transformados em bots, devido suas fracas configurações de segurança (KAMBOURAKIS; KOLIAS; STAVROU, 2017).

Depois de sua descoberta, o código fonte foi disponibilizado na Internet, o que levou ao surgimento de vários projetos variantes da Mirai. Estima-se que elas foram capazes de controlar aproximadamente meio milhão de dispositivos IoT e responsáveis por alguns dos maiores e mais catastróficos ataques DDoS. (DOSHI; APTHORPE; FEAMSTER, 2018)

Tendo essa nova tendência de *botnets* em vista, o estudo de métodos eficientes para detecção destas se faz necessário para que ataques deste tipo possam ser evitados. Este trabalho se propôs a desenvolver métodos que utilizem algoritmos de *Machine Learning* para detectar

botnets através do comportamento da rede, com base em estudos anteriores da área.

O foco do trabalho foi o estudo da detecção de *botnets* que utilizam o protocolo *Internet Relay Chat*. Este protocolo de aplicação, popularmente utilizado por programas de grandes salas de bate-papo, é historicamente o protocolo predominante entre os Servidores de Comando e Controle das *botnets*, apesar do seu uso ter diminuído nos últimos anos em favor do protocolo HTTP (BAPAT et al., 2018).

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo deste trabalho foi aplicar algoritmos de *Machine Learning* capazes de detectar a presença de *botnets* em uma rede, assim como fazer um estudo de eficiência com base nos resultados obtidos.

1.1.2 Objetivos Específicos

- a) Estudar algoritmos de *Machine Learning* e definir quais foram implementados;
- b) Estudo sobre as características do tráfego de rede relevantes para detecção de *botnets*;
- c) Implementação dos algoritmos;
- d) Treinamento e teste dos classificadores;
- e) Análise e comparação dos resultados;

1.1.3 Organização da monografia

Esta monografia está organizada da maneira que se segue.

O Capítulo 2 contém a fundamentação teórica, apresentando vários conceitos sobre *botnets* e *Machine Learning* utilizados.

O Capítulo 3 apresenta as ferramentas utilizadas para o desenvolvimento do trabalho.

O Capítulo 4 relata todo o processo de desenvolvimento, desde a obtenção e tratamento de dados até a execução dos algoritmos.

O Capítulo 5 apresenta os resultados obtidos ao final da fase de Desenvolvimento.

Por fim, o Capítulo 6 apresenta conclusões obtidas e sugestões de trabalhos futuros baseados nesta monografia.

2 Fundamentação Teórica

2.1 Botnets

Botnets são redes de computadores comprometidos, que são usados para cometer vários tipos de crimes virtuais, entre eles ataques de negação de serviço (*DDoS*), disseminação de vírus, fraudes de cliques para geração de dinheiros em propagandas *online*, e obtenção de dados pessoais (IANELLI; HACKWORTH, 2005).

Uma *Botnet* pode ser vista como o conjunto de três agentes: *bots*, máquinas que foram infectadas por um *malware*; um *botmaster*, pessoa capaz de controlar os *bots* de modo que eles executem ações determinadas por ele sem o consentimento de seus respectivos donos; e o Sistema de Comando e Controle, estrutura responsável por distribuir os comandos do mestre para todas as máquinas comprometidas. (MILLER; BUSBY-EARLE, 2016b)

Existem três topologias principais para as botnets, de acordo com o tipo de Comando e Controle utilizado (JANG et al., 2009). Uma *botnet* centralizada possui um servidor central ao qual o mestre envia os comandos, que depois repassa-os para todos os bots conectados. Elas se diferenciam pelo tipo de protocolo utilizado, sendo HTTP e IRC os mais utilizados.

Já as descentralizadas não possuem uma estrutura central, utilizando ao invés disso comunicações *peer to peer* (P2P) entre suas máquinas para as transmissões de mensagens.

Há ainda uma abordagem híbrida, que diferencia as máquinas infectadas entre servos e clientes, sendo as primeiras responsáveis por receber e enviar as mensagens e as últimas por executar os comandos recebidos.

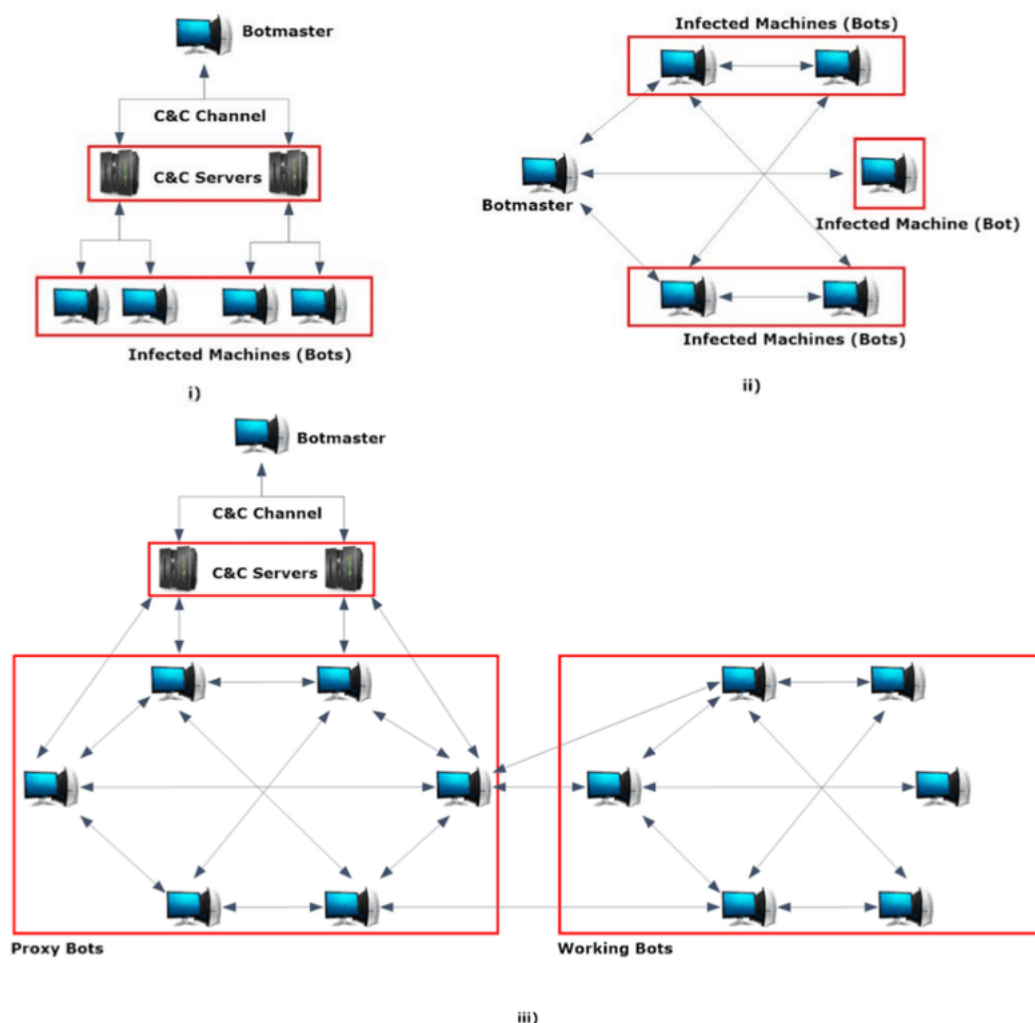
A figura 1 ilustra as diferentes topologias, aonde I representa uma *botnet* centralizada, II descentralizada, e III uma aplicação híbrida.

As *botnets* descentralizadas, por não possuírem uma estrutura central de comando, são mais difíceis de serem rastreadas e derrubadas, enquanto as centralizadas são mais suscetíveis a serem derrubadas, tendo em vista que se o Servidor de C&C for desativado, a *botnet* deixa de funcionar. (FEILY; SHAHRESTANI; RAMADASS, 2009)

Por outro lado, as *botnets* centralizadas são muito mais confiáveis e eficientes na questão da comunicação, enquanto as P2P apresentam problemas de latência que acabam diminuindo o poder de seus ataques. Tendo isso em vista, as topologias centralizadas ainda são utilizadas pela sua facilidade de utilização e gerenciamento, além da baixa latência (STEVANOVIC; PEDERSEN, 2013). Dentre estas, se destacam as que utilizam o protocolo IRC e HTTP.

O Protocolo *Internet Relay Chat* foi desenvolvido para ser utilizado em aplicações de trocas de mensagens em grupo (OIKARINEN; REED, 1993). Ele é baseado num modelo

Figura 1 – Arquiteturas de Botnet.



Fonte: (MILLER; BUSBY-EARLE, 2016a).

de cliente-servidor, onde um processo funciona como um ponto central, guardando o estado global de todos os outros clientes conectados a ele. Devido a um grande número de aplicações *open-source* baseadas no protocolo IRC, sua implementação acaba sendo muito mais fácil, sendo esse o motivo pelo qual é o mais prevalente historicamente, e o motivo de ser o foco deste estudo. (RAJAB et al., 2006)

O ciclo de vida de uma *botnet*, segundo (LEONARD; XU; SANDHU, 2009), pode ser descrito em quatro passos:

- Infecção: nesta etapa, o *botmaster* age infectando várias máquinas com *malwares*, incorporando-as desta maneira à rede;
- Ambiente de Comando e Controle: depois de infectada, uma máquina estabelece então uma conexão com o Servidor de C&C, ficando à disposição do mestre para que quando

ele desejar, possa enviar uma mensagem para o Servidor, e através deste a ordem é então repassada para todos os *bots* conectados à sua rede;

- Ataque: depois que os *bots* recebem as ordens, eles passam então a executar aquilo que foi determinado pela instrução enviada, como enviar *e-mails* de *spam*, por exemplo;
- Pós-Ataque: depois que o ataque foi concluído, alguns *bots* podem ser descobertos, levando a uma remoção do vírus e consequentemente, a remoção da máquina da *botnet*. Sendo assim, o foco do mestre é executar a manutenção da rede, e por fim focar no recrutamento de novas máquinas, ponto onde o ciclo recomeça.

Na área de detecção de botnets, têm se destacado muito por sua eficiência técnicas baseadas na mineração de dados, utilizando algoritmos de ML, como observado em (FEILY; SHAHRESTANI; RAMADASS, 2009). A maioria destas técnicas de detecção costumam trabalhar com fluxos de redes, ao invés de pacotes isolados, devido ao fato dos primeiros apresentarem uma possibilidade muito maior de características a serem utilizadas (BEIGI et al., 2014).

Muitos estudos têm sido feitos nos últimos anos com base nessas técnicas, explorando várias abordagens presentes na literatura de ML, variando desde a utilização de técnicas clássicas como por exemplo o algoritmo de Regressão Logística (BAPAT et al., 2018), até o uso de técnicas mais recentes da área de *Deep Learning*, através da implementação de Redes Neuras Artificiais Convolucionais, como estudado em (CHEN; CHEN; TZENG, 2018) e (KANT; SINGH; OJHA, 2017).

2.2 Machine Learning

Machine Learning é uma subárea da IA que busca desenvolver algoritmos capazes de extrair conhecimento com base em dados, através da junção de conceitos de Estatística e Ciência da Computação (JAMES et al., 2014).

Existem dois tipos principais de algoritmos de ML, os algoritmos supervisionados e não-supervisionados.

2.2.1 Supervisionado e Não-supervisionado

Na área de ML supervisionado, são fornecidos os dados de entrada e os dados de saída respectivos, e com base nestes, o algoritmo busca interpretar e buscar padrões nestes dados que possa diferenciá-los.

Dentro do contexto de algoritmos de ML supervisionados, que é o foco deste trabalho, existem dois tipos de problema principais: Regressão e Classificação.

Problemas de Regressão são capazes de prever variáveis quantitativas, ou seja, valores numéricos. Um exemplo seria encontrar o ganho anual de uma pessoa de acordo com a escolaridade, idade e local de residência. (MÜLLER; GUIDO, 2017)

Já para o caso onde tenta-se prever uma variável qualitativa, ou seja, encontrar a qual classe C_n um certo item de entrada pertence, dado um conjunto de n classes, têm-se um problema de Classificação. Para casos onde $n = 2$, têm-se uma classificação binária, e quando $n > 2$, uma classificação multi-classe.

Já em métodos não-supervisionados, as classes dos dados não são passadas ao algoritmo, de maneira que ele busca identificar padrões e relações entre eles, procurando alguma informação possa ser obtida delas.

Para isso, utilizam-se principalmente as chamadas técnicas de clusterização, que buscam separar os dados em vários grupos de acordo com suas características observadas no conjunto de dados. (MÜLLER; GUIDO, 2017)

2.2.2 Fases de treinamento e teste

Na área de ML costuma-se utilizar dois conjuntos de dados diferentes na fase de criação de modelos: conjunto de teste e conjunto de treino. O primeiro é utilizado pelo algoritmo para aprender as relações entre os dados, e o de teste para validar a sua eficiência.

Para que se obtenham esses conjuntos, uma prática comum é utilizar a técnica de *Holdout*, que separa uma parte do conjunto de dados para que seja utilizada para teste, enquanto o resto é usado para treinar o modelo. Geralmente isso é definido através de uma proporção, como 70% do conjunto para teste e 30% para treino, por exemplo. (AALST et al., 2010)

Utilizando essa abordagem, espera-se observar que o modelo treinado seja generalizado, capaz de prever dados os quais nunca tenha trabalhado antes da maneira correta. No entanto, isto pode não ocorrer devido a:

- *Overfitting*: ocorre quando o modelo fica muito sensível a certas particularidades do conjunto de treinamento. Isto costuma ocorrer em modelos mais complexos que utilizam um número muito grande de características.
- *Underfitting*: ocorre quando o modelo, por ser muito simples, não é capaz de aprender as características dos dados, e dessa maneira falha em apresentar os resultados esperados.

O desafio na área de ML então é obter um modelo que esteja bem balanceado, não sendo nem muito simples nem muito complexo, de modo a evitar um dos problemas descritos acima. Para isto, podem-se utilizar várias técnicas, dentre elas se destacando o *Cross Validation*, que buscam medir a generalização dos dados (MÜLLER; GUIDO, 2017).

A técnica de CV baseada em k-folds consiste em separar o conjunto de treinos em k subconjuntos de tamanhos iguais, onde cada um será usado para treinar um modelo diferente. Estes modelos então são verificados alternando o conjunto de treino e utilizando os outros $k - 1$ como teste, e a performance do algoritmo é obtida através da média das performances individuais.

Algo importante de se notar também é a distribuição do número de amostras. De maneira geral, classificadores costumam ser muito mais efetivos quando trabalham com dados que se encontram balanceados, tendo sua efetividade diminuída caso contrário. (YAP et al., 2014)

Neste contexto, podem ser aplicadas duas técnicas: *Oversampling* e *Undersampling*, que buscam aumentar ou diminuir, respectivamente, o número de amostras de uma classe específica de maneira a deixar o conjunto balanceado. A técnica de *Oversampling* costuma ser mais trabalhosa por envolver a criação e adição de novos dados ao conjunto.

Por fim, outro conceito importante para a generalização dos algoritmos é o dos hiper-parâmetros. Alguns algoritmos possuem parâmetros especiais que definem o seu desempenho e performance, e enquanto alguns algoritmos trabalham bem com seus valores padrão, para outros é necessário executar testes para encontrar os valores ótimos a serem passados.

Neste contexto, utiliza-se a técnica de *Grid Search*. Esta técnica consiste em testar todas as combinações possíveis de um certo conjunto de valores e verificar qual delas gera o melhor modelo possível (BERGSTRA; BENGIO, 2012). É possível utilizá-la em conjunto com a técnica de CV para obter um nível de confiabilidade maior.

2.2.3 Métricas

Para se medir a eficiência de um algoritmo, pode-se classificar cada uma de suas predições dentro de quatro possibilidades: Verdadeiro Positivo (VP), Verdadeiro Negativo (VN), Falso Positivo (FP), Falso Negativo (FN). Estes dados são convenientemente agrupados para melhor visualização na chamada Matriz de Confusão. (FLACH, 2003)

A partir desses dados, é possível obter alguns valores que indicam a eficiência do algoritmo, os quais se destacam:

- Precisão: $\frac{VP}{(VP+FP)}$
- Recall: $\frac{VP}{VP+FN}$
- F-score: $\frac{2 \cdot \text{Precisao} \cdot \text{Recall}}{(\text{Precisao} + \text{Recall})}$
- Acurácia: $\frac{VP+VN}{VP+VN+FP+FN}$

Figura 2 – Exemplo de Matriz de Confusão.

		Valor Previsto	
		Positivo	Negativo
Valor Verdadeiro	Negativo	Verdadeiros Positivos	Falsos Negativos
	Positivo	Falsos Positivos	Verdadeiros Negativos

Fonte: ([ANDREONI, 2014](#)).

2.2.4 Algoritmos

Nesta seção, serão apresentados os algoritmos da área de ML estudados para a execução deste trabalho.

2.2.4.1 Naive Bayes

Naive Bayes é um classe de algoritmos de ML que possuem sua fundação no Teorema de Bayes. Possui o nome *Naive*, pois é baseado na suposição de que todas as características do problema são independentes umas das outras, algo que muito raramente acontece na grande maioria das aplicações do mundo real.

No entanto, apesar de sua suposição "inocente", classificadores NB são um dois mais eficientes e efetivos algoritmos da área de ML, sendo que se destacam de maneira maior na área de classificação de textos. ([ZHANG, 2004](#))

Dada uma classe c , e um conjunto de características $F = x_1, x_2, \dots, x_n$, a relação entre elas com base no Teorema de Bayes pode ser descrita da seguinte fórmula:

$$P(c, F) = \frac{P(F|c)P(c)}{P(F)}, \quad (2.1)$$

e a suposição *naive* de que todas as variáveis são independentes umas das outras implica que probabilidade de uma variável dado o valor da classe não depende do valor das outras variáveis do conjunto. Isso pode ser representado da seguinte forma:

$$P(F, c) = P(x_1, x_2, \dots, x_n|c) = \prod_{i=1}^n P(x_i|y) \quad (2.2)$$

ou seja, a probabilidade do conjunto F sobre c pode ser representada pelo produto das probabilidades individuais de cada característica x_i do conjunto sobre uma classe.

Sendo assim, é possível obter uma equação para Naive Bayes como se segue:

$$r = \operatorname{argmax}_r P(c) \prod_{i=1}^n P(x_i|y) \quad (2.3)$$

que mostra que o rótulo de uma entrada r será atribuído verificando para qual valor de classe c é obtido uma maior probabilidade com base no conjunto de características do problema.

O que vai diferenciar os tipos de algoritmos de Naive Bayes é o modo como $P(x_i|y)$ é calculado, ou seja, qual o tipo de distribuição que será usada. Estas podem ser Multinomial, Gaussiana ou Bernoulli (METSIS; AL., 2006).

2.2.4.2 Support Vector Machines

O algoritmo de *Support Vector Machines* foi criado na década de 90 por (CORTES; VAPNIK, 1995), e desde então vem recebendo muita atenção da comunidade de ML por seu alto desempenho. Ele foi pensado inicialmente como um classificador binário, no entanto, muitas extensões foram desenvolvidas que permitem o uso da técnica para problemas de classificação de mais de uma classe e para regressão.

A ideia básica do SVM é encontrar um hiperplano ótimo que seja capaz de dividir os dados em duas regiões, de maneira que cada uma delas contenham dados de uma só classe. A equação geral de um hiperplano de dimensão n é demonstrada a seguir:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots = \beta_n x_n = 0 \quad (2.4)$$

onde x_1, \dots, x_n representam as coordenadas de um certo ponto no \mathbb{R}^n e β_1, \dots, β_n são constantes.

Considerando que para um ponto x_n que obedeça a igualdade, este ponto está localizado exatamente acima do hiperplano, o objetivo do algoritmo então é que, dado um problema que possui dados de duas classes c_1 e c_2 , todos os pontos de uma destas tenha o seguinte comportamento

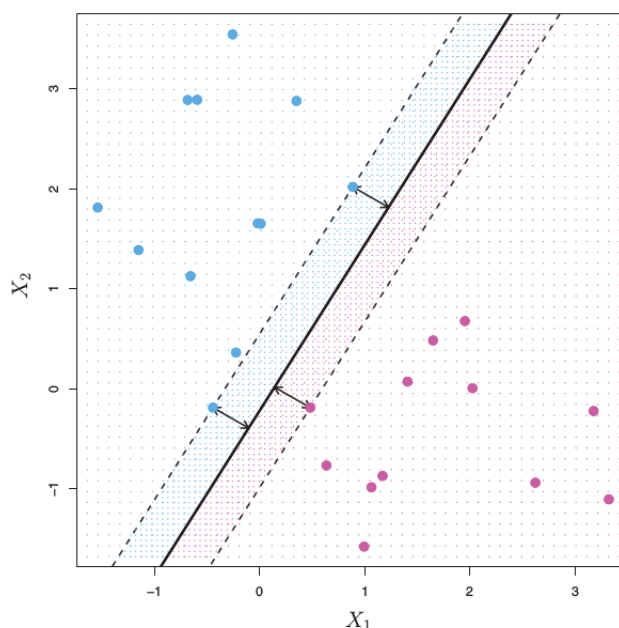
$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots = \beta_n x_n > 0 \quad (2.5)$$

enquanto para a outra classe seja observável que

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots = \beta_n x_n < 0 \quad (2.6)$$

ou seja, cada classe é encontrada somente em uma região, cada uma de um lado do hiperplano. Para definir a localização deste, busca-se um local onde a distância entre os pontos mais próximos do hiperplano de cada classe, neste contexto denominada margem, seja maximizado. Esta técnica é conhecida como *Marginal Margins Classifier* (JAMES et al., 2014) A figura 3 ilustra a ideia básica destes conceitos.

Figura 3 – Hiperplanos no conceito de SVM.



Fonte: (JAMES et al., 2014).

No entanto, um caso de dados totalmente capaz de ser separado linearmente é muito raro de ser obtido em uma situação real. Tendo isso em vista, pode ser utilizada uma abordagem diferente, *Soft Margins Classifier* aonde é permitido que pontos estejam localizados não só na região da margem, mas inclusive do lado errado do hiperplano.

Porém existem ainda os casos aonde os dados não possuem uma distribuição linear, e sendo assim não é possível estabelecer um hiperplano entre as classes. Porém, estes dados podem ser separáveis em um espaço de maior dimensão, como demonstrado na Figura 4 (KIM, 2013).

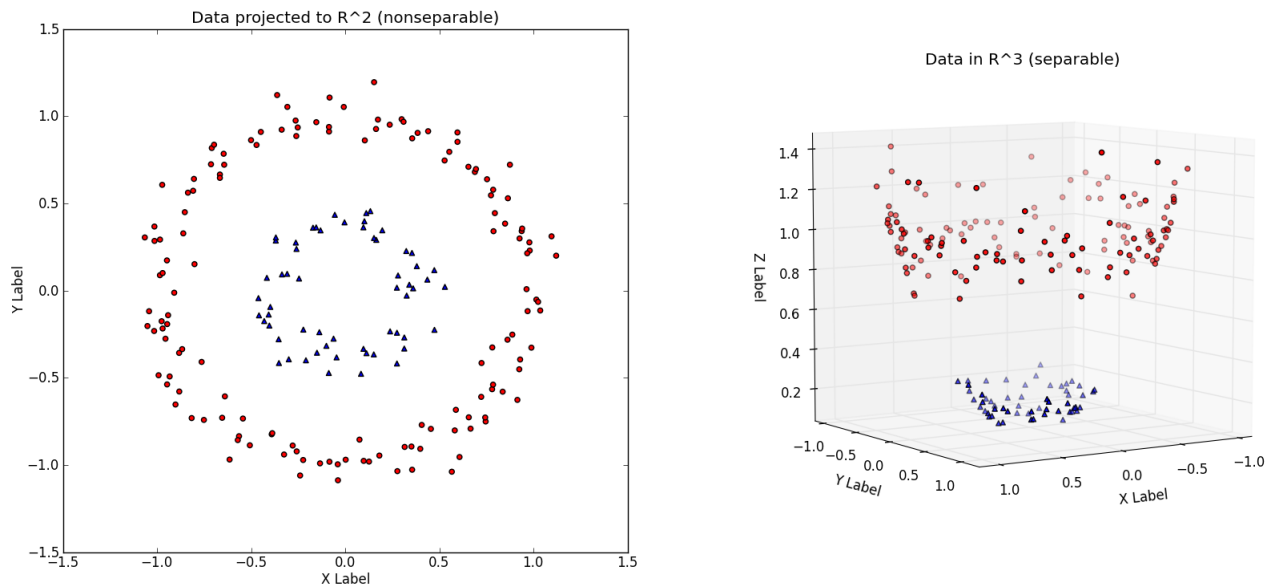
Depois de obtido o plano no espaço de maior dimensão, este pode ser projetado na dimensão anterior e assim se consegue um hiperplano divisor. Porém, o processo dessas transformações é muito caro computacionalmente, e neste contexto se faz útil a utilização do chamado *Kernel Trick*.

É possível observar através de comprovações matemáticas, que os resultados do algoritmo de SVM são obtidos através do produto escalar entre seus dados (JORDAN; THIBAU, 2004). Neste contexto, são utilizadas as funções Kernel, que são capazes de calcular o produto escalar de dois vetores, representado por $\langle x, x' \rangle$, no R^n em um espaço R_m , onde $m > n$, sendo assim capaz de calcular hiperplanos de maneira muito barata computacionalmente.

As funções Kernel mais utilizadas neste contexto são mostradas no Quadro 1.

Uma grande desvantagem do SVM é que eles exigem que seus hiper-parâmetros sejam muito bem otimizados para que ele tenha um bom desempenho, assim exigindo um estudo maior para ser utilizado.

Figura 4 – Caso de uso de transformação em SVM.



Fonte: (KIM, 2013).

Quadro 1 – Funções Kernel utilizadas em SVM.

Nome	Função Kernel
Linear	$\langle x, x' \rangle$
Polinomial	$(\gamma \langle x, x' \rangle + r)^d$
Radial Basis Function	$\exp(-\gamma \ x - x'\ ^2)$
Sigmoidal	$\tanh(\gamma \langle x, x' \rangle + r)$

Fonte: Elaborado pelo Autor.

O Hiper-parâmetro C , constante de margens suaves, é comum a todos os Kernels e determina o peso da minimização dos erros no conjunto de treinamento. Além disso, como observado na Tabela 1, alguns Kernels possuem valores específicos que devem ser determinados como γ , r e d (degree) (LORENA; CARVALHO, 2007).

2.2.4.3 Árvores de Decisão

Árvores de decisão, no contexto de ML, são métodos que se assemelham muito a fluxogramas, pois busca através de perguntas do tipo *if-else* encontrar um certo resultado (HARRINGTON, 2012).

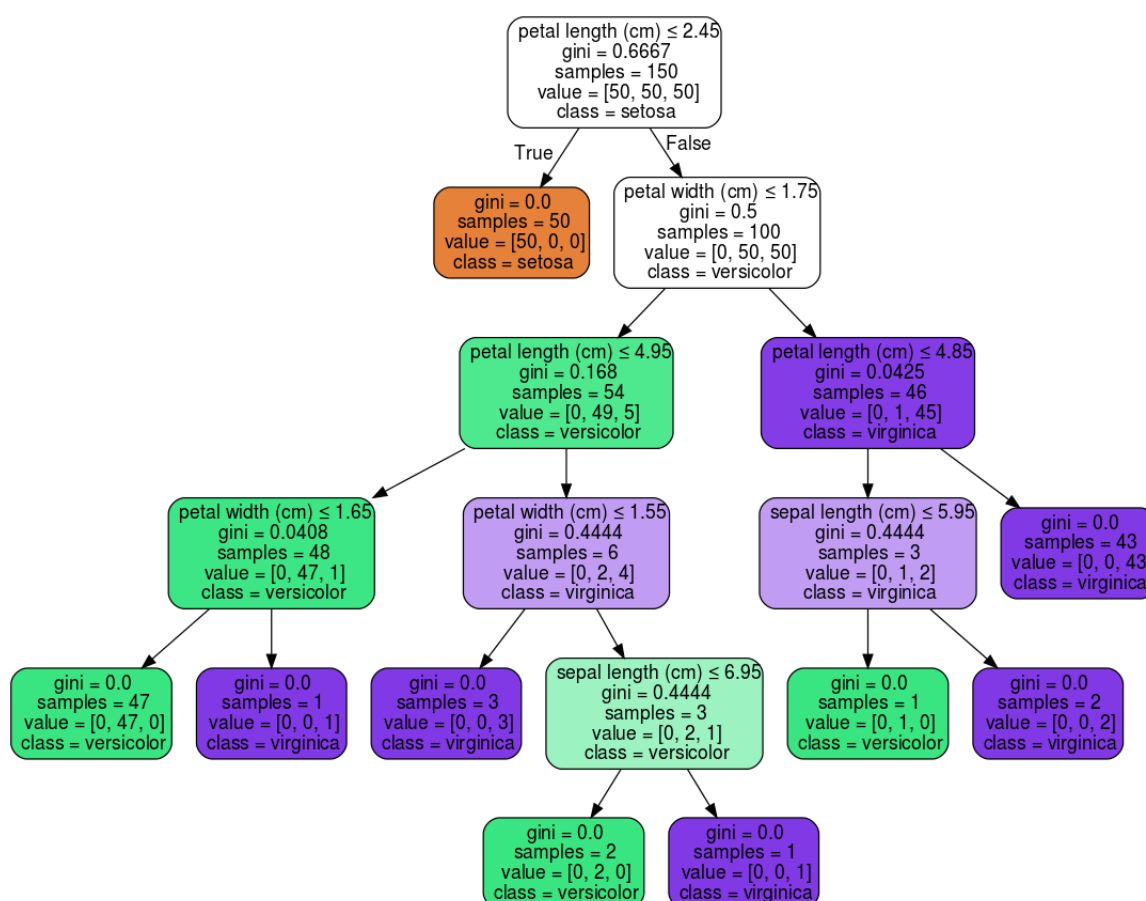
Para definir os testes e divisões, as árvores de decisão trabalham com uma abordagem denominada *splitting*, onde de maneira recursiva os dados são divididos em regiões menores, de modo que eles fiquem de maneira mais dividida possível.

Uma árvore de decisão possui três tipos de nós:

- Nó raiz: o nó inicial da árvore, por onde começa o processo de divisão;
- Nós de decisão: nós que contêm cada uma das decisões feitas pelo algoritmo, definem através de comparações qual o caminho a ser percorrido pela árvore;
- Nós folha: são os nós presentes no final da árvore. Eles indicam qual a classe que o algoritmo foi capaz de prever

Primeiramente, o algoritmo busca uma característica inicial para iniciar as divisões e ser o nó raiz, que no caso é a que pode dividir da melhor maneira possível o conjunto de dados de modo que as classes fiquem bem separadas, podendo inclusive dividir em mais do que duas regiões. Após a divisão, são adicionados nós de teste à árvore, e novas características são selecionadas para a divisão do próximo passo. A figura 5 mostra o exemplo de uma árvore de decisão, baseada no conjunto de dados de flores iris.

Figura 5 – Exemplo de Árvore de Decisão.



Fonte: (BUTINCK et al., 2013).

Existem duas condições de parada para o algoritmos: as características possíveis para o problema se esgotarem, ou até que todas as instâncias das sub-regiões sejam pertencentes a uma mesma classe. (JAMES et al., 2014)

Apesar deste método ser bem vantajoso por ser facilmente entendível e interpretável, infelizmente não são muito robustos por serem muito suscetíveis a perdas de precisão causadas por pequenas mudanças nas características, além de serem mais suscetíveis a *overfitting*.

2.2.4.4 Florestas Aleatórias

Florestas aleatórias é um método baseado em Árvores de Decisão. O algoritmo funciona com base em um número n , que determina quantas árvores são criadas, e para cada árvore n é selecionado de maneira aleatória um subconjunto de p características que esta árvore utilizará, considerando um número t de total de características. De maneira geral, o valor de p é calculado através de $p = \text{raiz quadrada de } t$. (JAMES et al., 2014)

Este procedimento é muito eficiente, pois ao fazer com que as árvores sejam criadas com um número reduzido de características, elimina-se a possibilidade de que uma que seja muito mais forte do que as outras influencie tanto no resultado, de maneira que as árvores geradas sejam bem diferentes umas das outras e possuam um valor de variância mais alto entre elas. (BREIMAN, 2001)

Ao fim deste processo, as predições são feitas através de um sistema de votação. Os dados são passados por todas as árvores da floresta onde cada uma irá predizer uma classe, e enfim os votos serão contados e o resultado será definido pela classe com maior número de predições. Tomando como exemplo o caso de um sistema de classificação, que foi treinado por uma Floresta Aleatória de 100 árvores para separar suas entradas em duas classes X e Y. Cada entrada teria suas características processadas por todas as árvores e suas respectivas saídas obtidas. No caso de terem sido obtidos 60 predições de classe X e 40 predições de classe Y, essa entrada seria classificada como pertencente à classe X. A figura 6 ilustra o funcionamento das Florestas Aleatórias.

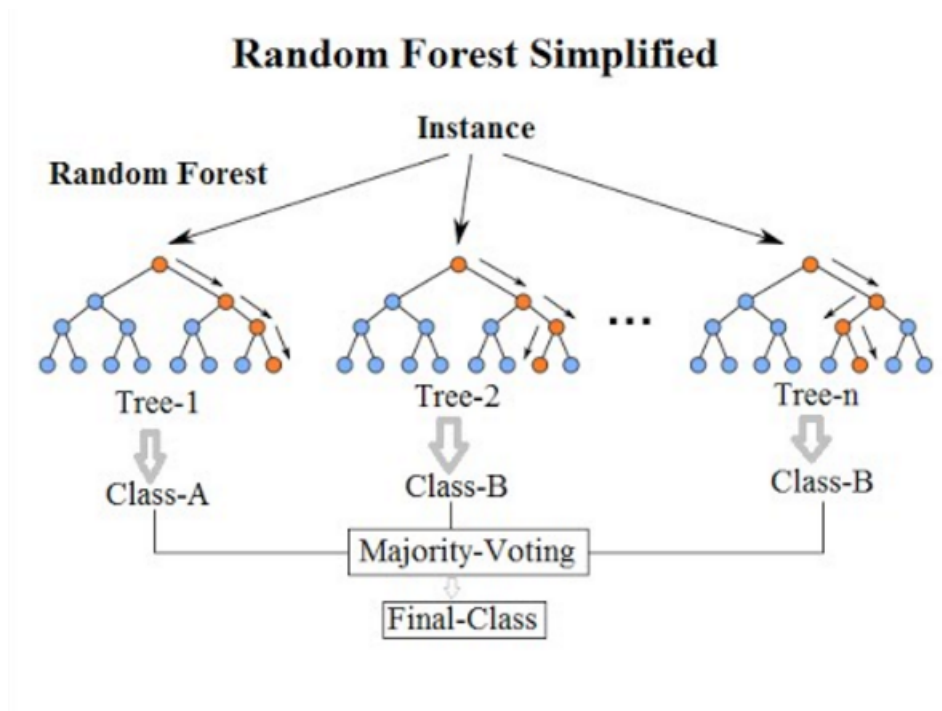
2.2.4.5 AdaBoost

AdaBoost, abreviação de *Adaptive Boosting*, é um algoritmo formulado por (FREUND; SCHAPIRE, 1997), que busca melhorar a performance de outros classificadores. No contexto desse trabalho, será abordado a utilização deste algoritmo em conjunto com Árvores de Decisão.

O AdaBoost possui semelhanças com o algoritmo de Florestas Aleatórias, visto que ele busca trabalhar com um conjunto de classificadores de forma a obter um resultado mais confiável, porém se diferencia ao obter no final de seu processo, um modelo ao invés de utilizar a abordagem de votação.

O algoritmo funciona mantendo um valor de peso w_i para cada dado do conjunto de treino passado para o algoritmo. Depois, são feitas várias iterações onde em cada uma é gerado um novo classificador fraco, que é o nome dado a um modelo que possui uma performance minimamente maior comparado a um algoritmo totalmente randômico, que possui acurácia

Figura 6 – Funcionamento de Florestas Aleatórias.

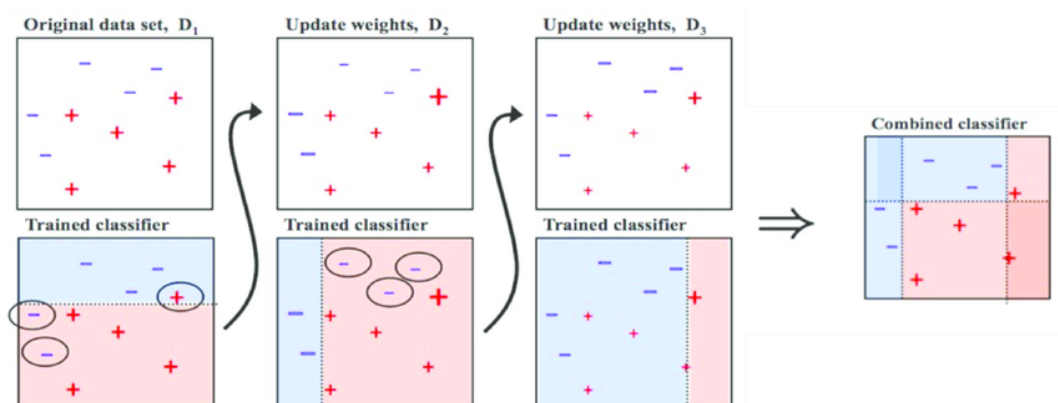


Fonte: (MEDIUM, 2017).

próxima de 50 por cento.

É feita então uma predição com base neste algoritmo, e os pesos das variáveis são atualizados de acordo com o valor obtido pelo classificador, de maneira que se um dado foi classificado de maneira correta, seu peso diminui, e seu peso aumenta em caso contrário (ROJAS, 2009). Depois disso, o subconjunto utilizado pelo próximo classificador fraco é escolhido com base nos valores dos pesos. A figura 7 ilustra o processo do algoritmo de AdaBoost.

Figura 7 – Funcionamento do AdaBoost.



Fonte: (MARSH, 2016).

Dessa maneira, o AdaBoost consegue favorecer em seus treinamentos o uso de variáveis

que são mais difíceis de serem classificadas corretamente, o que faz dele muito eficiente em identificar *outliers*. Ao final do processo, um modelo robusto é gerado com base nos passos anteriores dos classificadores fracos.

2.2.4.6 Recursive Feature Elimination

O RFE é um algoritmo recursivo que busca encontrar um subconjunto ótimo de características para um classificador específico, através do processo de atribuição de um ranking de importância de cada uma delas.

O algoritmo começa sua execução com todas as características do conjunto passado, cria um modelo com base nelas, e calcula a importância de cada uma para a execução e performance do modelo. (MÜLLER; GUIDO, 2017) Depois disso, o algoritmo elimina a característica que teve o menor valor de importância e repete o processo recursivamente.

Ao final da execução, o algoritmo seleciona as n características mais bem colocadas em seu ranking, obtendo assim o melhor conjunto possível, sendo n um número pré-definido.

2.3 Revisão Bibliográfica

Como base para este estudo, foram utilizados alguns artigos. O primeiro, *Machine learning for identifying botnet network traffic* (STEVANOVIC; PEDERSEN, 2013), faz um estudo comparativo entre vários estudos da área, separando-os de acordo com tipo de ML e protocolo estudado, e depois apresentando características de cada um. Desse estudo, foram obtidos os seguintes artigos como base:

Quadro 2 – Estudos da área utilizados como base para escolha dos classificadores.

Autor(es)	Classificadores usados	Características utilizadas
(LIVADAS et al., 2006)	Árvores de Decisão, Naive Bayes, Redes Bayesianas	10
(STRAYER et al., 2008)	Árvores de Decisão, Naive Bayes, Redes Bayesianas	16
(MASUD et al., 2008)	Árvores de Decisão, Naive Bayes, Redes Bayesianas, SVM, Árvores de Decisão com Boosting	20

Fonte: Elaborado pelo Autor.

De acordo com o Quadro 2, foram obtidos os estimadores e características base deste trabalho, com algumas ressalvas. Nota-se que Redes Bayesianas se fazem presente em todos eles, no entanto as bibliotecas deste trabalho não possuem implementação deste algoritmo.

Portanto, Florestas Aleatórias foi incluído como substituto, por também ter sido utilizado em estudos relacionados (KANT; SINGH; OJHA, 2017).

Além disso, como não foi especificado nos trabalhos quais técnicas de *Boosted Decision Trees* e Naive Bayes foram utilizadas, optou-se pelo uso do AdaBoost e pelo teste de Naive Bayes Multinomial e Bernoulli, respectivamente.

O segundo estudo é o *Towards Effective Feature Selection in Machine Learning-Based Botnet Detection Approaches* (BEIGI et al., 2014), que apresenta um estudo detalhado sobre as características de rede predominantes na área. Com base neste e nos estudos utilizados anteriormente, foram definidas quais delas seriam utilizadas.

Em conjunto com estes, também foram observados outros estudos com base que também apresentaram a utilização dos mesmos conjuntos de características, como (ALEJANDRE; CORTÉS; ANAYA, 2017) e (MAHARDHIKA; SUDARSONO; BARAKBAH, 2017).

Entre elas, têm-se:

- IP/Port de Origem e Destino: Estas características são utilizadas para a identificação dos fluxos de rede maliciosos;
- Protocolo: utilizado principalmente para a etapa de filtragem de pacotes, diminuindo dessa maneira o volume de pacotes a serem processados pelo algoritmo. Por exemplo, no contexto de uma *Botnet* que utiliza o protocolo UDP para comunicação, é possível então filtrar os fluxos para que contenham somente os que utilizam o mesmo;
- Duração: característica muito utilizada para identificação de todos os tipos de protocolos de *Botnet*, tendo em vista que elas de maneira geral costumam ter dois tipos bem distintos de conexão, sendo uma conexão inicial bem rápida seguidas de sessões muito longas;
- Características de tamanho de fluxo: Nesta categoria, estão presentes o Total de Bytes/Bits/Pacotes, Bytes por Pacote e Tamanho Médio de *Payload*. Estas características ajudam a distinguir o fluxo de uma *Botnet* de um tráfego normal baseado no fato de que o tráfego de *Botnets* costuma ser muito mais uniforme do que os outros, que apresentam comportamentos mais variados dependentes da aplicação;
- Características de tempo de conexão: Nesta categoria, estão presentes Média de pacotes/bits por segundo, Média e Variância do tempo entre transmissão de pacotes (IAT). São utilizadas pelos mesmos motivos das características de tamanho de fluxo, para distinguir pelo comportamento da rede os tráfegos de *Botnet*;
- IOPR/Porcentagem de pacotes enviados: IOPR representa a relação entre o número de pacotes que chegam sobre o número de pacotes que são enviados em uma conexão. Assim como a porcentagem de pacotes enviados, estas características estão ligadas a estudos

que indicam que a relação entre pacotes enviados e recebidos em redes normais segue uma distribuição uniforme. Porém, como estudos como ([ALMGREN; JOHN, 2010](#)) indicam que em redes infectadas por agentes maliciosos esta relação costuma ser desbalanceada, e portanto, pode ser utilizada para identificação de *Botnets*.

3 Metodologia

3.1 Ferramentas

3.1.1 Scikit-learn

O Scikit-learn é uma biblioteca open source de *Machine Learning* na linguagem Python. O projeto iniciado em 2007 por David Cournapeau foi publicado oficialmente pela primeira vez em 2010, e desde então vem recebendo novas atualizações trimestrais, sendo que atualmente se encontra na versão 0.19.2. (BUTINCK et al., 2013)

A ideia da biblioteca é oferecer acesso facilitado a várias ferramentas da área, que vão desde os algoritmos, supervisionados e não-supervisionados, até outras funcionalidades como transformadores de dados (Por exemplo, o módulo de Principal Component Analysis, e Tfidf para Processamento de Linguagem Natural) e ferramentas gerais como o Grid Search e Pipeline. (PEDREGOSA et al., 2011)

A biblioteca não apresenta interface de linha de comando ou interface gráfica, sendo que todo seu funcionamento se dá através de chamadas de funções Python.

Todas as classes da bibliotecas são construídas com base no mesmo padrão de interface básico, o que facilita muito o seu uso. Cada estimador possui um método 'fit' onde são passados os dados para que seja feito o treinamento, e um método 'predict' que executa as predições com base no modelo treinado no passo anterior. Além disso, todos os hiperparâmetros dos métodos são acessíveis e modificáveis através do seu respectivo construtor de classe.

Na figura 8 é demonstrado um exemplo de utilização da biblioteca para a execução do modelo de Regressão Linear.

Figura 8 – Exemplo de utilização do Scikit.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score

linearReg = LinearRegression() # Instanciando o estimador
linearReg.fit(data_train, values_train) # Aplicando o modelo para o conjunto de dados de treino
predictions = linearReg.predict(data_test) # Obtendo predições com base no modelo treinado
accuracy_score(predictions, values_test) # Obtendo valor de acurácia do modelo treinado
```

Fonte: Elaborada pelo autor.

3.1.2 Numpy

O NumPy é uma biblioteca, integrante do conjunto Scipy, feita em cima da biblioteca Numeric. Seu objetivo é fornecer várias funções e estruturas de dados que possam facilitar o uso da linguagem Python em um contexto científico. Teve sua primeira versão lançada em 2005, e atualmente se encontra na versão 1.15.1. (OLIPHANT, 2006)

Algumas de suas funcionalidades são:

- Um poderoso objeto vetor de N-dimensões (denominado Numpy Array);
- Métodos para se trabalhar com vetores de maneiras mais sofisticadas;
- Funções de álgebra linear, transformações de Fourier e ferramentas para trabalhar com numeros aleatorios;
- Ferramentas de integração para códigos em Fortran e C/C++;

O NumPy se destaca por ser a base de várias outras bibliotecas científicas do Python, como o próprio Scikit que utiliza as estruturas de Numpy Array para seus cálculos.

3.1.3 Pandas

O Pandas é uma biblioteca que busca suprir uma demanda que a linguagem Python não tem por padrão, através de uma poderosa e robusta estrutura para se trabalhar com uma grande quantidade de dados estruturados, muito utilizados nas áreas de Estatística, Mercado Financeiro, Data Science, entre outros. (MCKINNEY, 2011) O Pandas se destaca por ser baseado no NumPy, sendo assim muito facilmente integrado com todas as outras bibliotecas de ML em Python, e também por ser muito rápido e eficiente.

O Pandas apresenta duas estruturas de dados principais, *Series* e *Data Frames*, sendo a primeira uni-dimensional e a segunda bi-dimensional. Dentre essas duas se destaca o *Data Frame*, similar ao *data.frame* que está presente nativamente para a linguagem R, porém com algumas melhorias. A figura 9 ilustra um exemplo de uso desta estrutura.

Algumas das principais funções do Pandas são:

- Fácil manipulação de dados nulos;
- Várias opções para leitura de arquivos, como por exemplo tabelas Excel, CSV, tabelas de Bancos de Dados, além de leitura de dados estruturados em arquivos HTML
- Inserção e remoção de colunas;
- Redimensionamento e obtenção de subsets;

- Muitos métodos utilizados no contexto de Bancos de Dados, como merge, join e groupby.

Figura 9 – Exemplo de utilização do Pandas.

```
In [4]: import numpy as np
import pandas as pd

data_frame = pd.read_csv('iris.csv') # Lendo o conjunto de dados iris salvo em um csv
data_frame.head(5) # Print das primeiras 5 linhas do data frame
```

Out[4]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Fonte: Elaborada pelo autor.

3.1.4 Matplotlib

Matplotlib é uma biblioteca para criação de visualizações 2D em Python, que começou a ser desenvolvida em 2003 por John D. Hunter. O intuito da biblioteca era atender alguns requisitos, entre eles ser possível de ser integrado a uma interface gráfica, funcionar em diversas plataformas, oferecer suportes para publicações e funcionar em aplicações interativas. (HUNTER, 2007)

O Matplotlib funciona com uma interface similar ao do MATLAB, através do módulo pyplot. Com ele, é possível em poucas linhas ser capaz de gerar histogramas, gráficos de barra, gráficos de dispersão, entre outros. A figura 10 demonstra a geração de um gráfico utilizando a biblioteca.

Além disso, os gráficos são altamente customizáveis, oferecendo controle total de elementos como estilos de linha, propriedades dos eixos, etc.

3.1.5 Seaborn

O Seaborn é uma biblioteca Python, que funciona como um *wrapper* para o Matplotlib. Ele busca simplificar ainda mais a geração de gráficos, oferecendo uma interface ainda mais simplificada que permite com que visualizações sejam geradas com apenas uma linha de código, como demonstrado na figura 11. Ele incorpora alguns tipos de gráficos mais avançados como Gráficos de Calor e Gráficos de violino, e também por padrão customiza os gráficos de uma maneira mais elegante, mesmo que eles tenham sido gerados usando somente o Matplotlib.

Todas as customizações são feitas através de parâmetros de função, e além disso é possível utilizar chamadas da biblioteca do Matplotlib para maiores customizações.

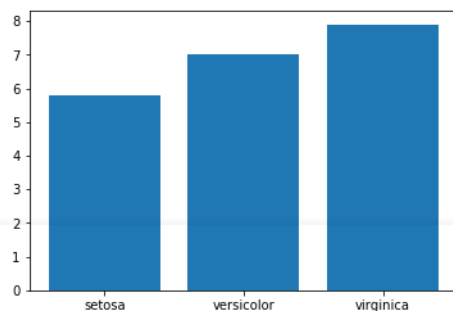
Além disso, o Seaborn traz em sua biblioteca um conjunto de *datasets*, o que facilita na hora de se estudar as possibilidades de visualizações e entender relações entre variáveis.

Figura 10 – Exemplo de utilização do Matplotlib.

```
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline

df = pd.read_csv('iris.csv')
plt.bar(df['species'],df['sepal_length'])
```

<BarContainer object of 150 artists>



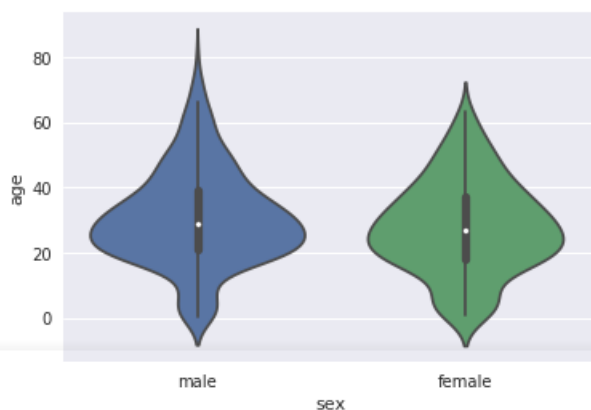
Fonte: Elaborada pelo autor.

Figura 11 – Exemplo de utilização do Seaborn.

```
import seaborn as sns

titanic = sns.load_dataset('titanic')
sns.violinplot(x='sex',y='age',data=titanic)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f760054fba8>



Fonte: Elaborada pelo autor.

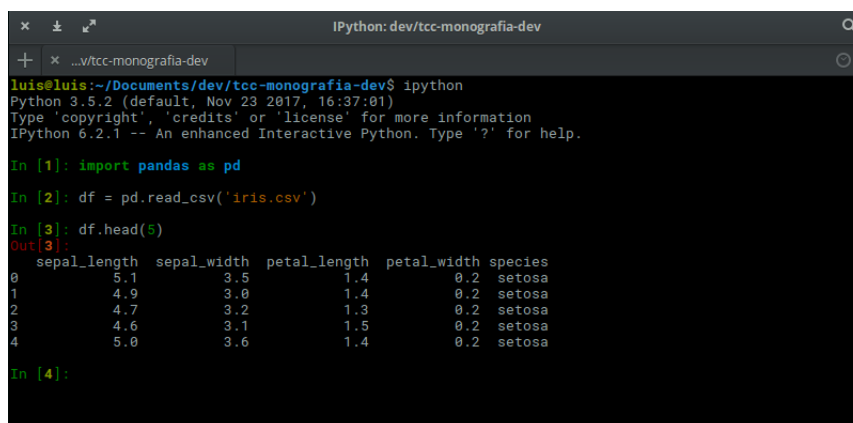
3.1.6 Jupyter Notebook

O Jupyter Notebook é uma aplicação *web open-source*, que permite a criação de documentos, denominados Notebooks, que contenham códigos, equações, visualizações e textos explicativos. Iniciado em 2014, foi construído com base no projeto IPython, porém hoje já apresenta suporte para mais de 50 linguagens que variam de C++ até Bash.

Um Notebook pode ser pensado como uma evolução dos consoles do tipo REPL (Read

Evaluate Print Loop), que permitem com que comandos fossem inseridos de maneira interativa e seus respectivos outputs pudessem ser visualizados em tempo real, como observado pelo uso da ferramenta IPython na Figura 12. O grande diferencial é que os arquivos do Jupyter permitem que as saídas de códigos sejam mais do que simples textos, apresentando suporte para imagens, controles e gráficos interativos e equações matemáticas formatadas. (KLUYVER et al., 2016)

Figura 12 – Exemplo de utilização do IPython.



```

IPython: dev/tcc-monografia-dev
+ x v/tcc-monografia-dev
luis@luis:~/Documents/dev/tcc-monografia-dev$ ipython
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import pandas as pd
In [2]: df = pd.read_csv('iris.csv')
In [3]: df.head(5)
Out[3]:
   sepal_length  sepal_width  petal_length  petal_width  species
0            5.1          3.5           1.4           0.2    setosa
1            4.9          3.0           1.4           0.2    setosa
2            4.7          3.2           1.3           0.2    setosa
3            4.6          3.1           1.5           0.2    setosa
4            5.0          3.6           1.4           0.2    setosa

In [4]:

```

Fonte: Elaborada pelo autor.

Os Notebooks foram pensados com o ambiente científico em mente, permitindo com que seja muito fácil visualizar e analisar dados facilitando o processo de desenvolvimento, assim como o posterior compartilhamento e publicação de modelos, tendo em vista que uma pessoa é capaz de não só visualizar todo o processo de criação do modelo assim como seus outputs e quaisquer textos de explicação adicionados pelo autor, mas também ser capaz de reproduzi-los em seu próprio ambiente simplesmente executando os códigos novamente.

Além disso, existem várias ferramentas que fazem com que o compartilhamento e acesso de Notebooks seja facilitado. Exemplos são o nbconvert, que converte o arquivo para os formatos HTML, PDF e LaTeX, e o nbviewer que permite que *Notebooks* hospedados na internet sejam visualizados no navegador sem necessidade de *download*.

A execução do Jupyter se dá pela linha de comando, no diretório onde os arquivos estão presentes. Depois de executado, uma interface será aberta no navegador permitindo com que os arquivos sejam abertos e editados.

4 Desenvolvimento

4.1 Base de dados

Para este trabalho foi utilizada uma base de dados fornecida pela *University of New Brunswick* ([UNIVERSITY OF NEW BRUNSWICK, 2018](#)). Esta base vem sendo utilizada em vários estudos recentes da área, como ([MAHARDHIKA; SUDARSONO; BARAKBAH, 2017](#)) e ([CHEN; CHEN; TZENG, 2018](#)), e portanto sua utilização é de extrema relevância. Foram fornecidos três arquivos no formato PCAP, sendo um voltado para a fase de treinamento e dois para testes. No entanto, os arquivos de teste apresentaram problemas, com ambos não podendo ser processados pelo programa.

Um deles apresentava problemas de desordenamento dos pacotes, que ao tentar ser resolvido levou a uma descaracterização dos dados, como por exemplo valores de duração negativos, algo que não é possível por se tratar de uma medida de tempo. Já o segundo apresentou problemas de corrompimento, que também impediram seu processamento.

Sendo assim, todo o desenvolvimento deste trabalho, assim como seus resultados, serão baseados somente no conjunto de treino fornecido, utilizando a técnica de *Hold-out* explicada anteriormente.

4.2 Geração de fluxos de rede

Para que os estimadores pudessem trabalhar no conjunto de dados, foi preciso primeiro passar os arquivos de pacotes por um gerador de fluxos bidirecionais, um programa capaz de analisar todos os pacotes e extrair características dos fluxos de rede presentes nele.

Para este fim, foi utilizado o programa *flowtbag*, uma ferramenta *open-source* desenvolvida na linguagem *Go*, disponível no *GitHub* ([ARNDT, 2011](#)). A interface de utilização do programa se dá através da linha de comando, onde são passados o caminho para o arquivo de pacotes a ser analisado, e o caminho para onde o arquivo final, no formato CSV, deve ser salvo.

O arquivo CSV que o programa cria contém uma tabela onde as linhas representam cada fluxo de rede, e as colunas representam as características extraídas. Ao todo são 44 características, sendo que as cinco primeiras são as que definem um fluxo: IP de origem, IP de destino, Porta de origem, Porta de destino, e Protocolo.

Além das acima citadas, se fazem presentes no conjunto:

- Total de Bytes;

- Total de Pacotes;
- Total de bytes alocados para cabeçalhos de pacote;
- Mínimo/Máximo/Média/Desvio Padrão da média de: menor pacote enviado, IAT, tempo ativo, tempo ocioso, ocorrências da flag *Push*, ocorrências da flag *Urgent*;

As características de fluxo acima, com exceção de tempo ativo e tempo ocioso, são divididas em dois tipos: *forward*, indica os valores correspondentes ao que foi enviado do IP de origem ao IP de destino, e *backwards* indica o contrário, ou seja, o que o destinatário enviou a quem iniciou a conexão. Por exemplo, o número de pacotes trocados durante o fluxo é representado por dois valores, *totalfpackets* e *totalbpackets*, onde o primeiro indica quantos pacotes foram transmitido na direção *forward* e o segundo na direção *backward*.

4.3 Processamento de dados

Depois de gerados o arquivo CSV, foi preciso fazer uma análise e processamento dos dados de forma que eles fossem usados pelos estimador. Para isto, foram utilizadas varias funcionalidades da biblioteca Pandas.

O programa gerador utilizado faz o processamento de conexões que utilizam o protocolo TCP e UDP, porém como o foco do projeto é analisar o fluxo de botnets do protocolo IRC, que por sua vez utilizam o protocolo TCP para transporte ([LIVADAS et al., 2006](#)), como explicado anteriormente, foi feita uma filtragem para que somente os protocolos desse tipo fossem mantidos na base de dados.

Os fluxos de protocolo TCP eram representados pelo número 6, enquanto o número 17 representava os fluxos UDP. Dessa maneira, foi necessário somente filtrar os fluxos com número de protocolo igual a 6.

Depois de eliminado os fluxos que não iriam ser utilizados, foi feita a rotulação dos dados, que consiste em atribuir para cada uma das linhas da tabela um valor que indica se aquele fluxo é de botnet ou não. Para essa tarefa, foram utilizados dois recursos: uma lista de IP's fornecida juntamente com a base de dados que informa o IP usado pelas botnets, e uma tabela pertencente ao estudo desenvolvido em conjunto com a base, que informa quais são as botnets presentes em cada conjunto.

Com essas informações, foi possível então desenvolver um método que iterasse por toda a tabela verificando se os IPs dos fluxos correspondiam aqueles dos ataques. Em caso positivo, era atribuído o valor 1, e 0 caso contrário, como visto na figura [13](#).

Além disso, foram também eliminados todos os fluxos de Botnets que não fossem do protocolo IRC, evitando assim com que houvesse interferência dessas informações nos resultados, tendo em vista que apesar de utilizarem protocolos diferentes, as redes ainda podem

Figura 13 – Rotulação dos fluxos.

```
malicious_ips = [...] # lista com todos os IPs
irc_attacks = [...] # lista de tuplas com os IPs envolvidos no ataque

def flow_label(df):
    labels = []

    for index, data in df.iterrows():
        src = data['srcip']
        dst = data['dstip']

        if((src in malicious_ips) or (dst in malicious_ips)):
            labels.append(1)
        elif((src, dst) in irc_attacks or ((dst, src) in irc_attacks)):
            labels.append(1)
        else:
            labels.append(0)

    return labels

df['label'] = flow_label(df)
```

Fonte: Elaborada pelo autor.

compartilhar certos tipos de comportamentos gerais, o que poderia acarretar em predições equivocadas.

Por fim, foi observado que a distribuição de fluxos por classe estava desbalanceada, com 86695 fluxos da classe 0 e somente 6379 da classe 1. Como os classificadores costumam trabalhar melhor com distribuições de dados balanceadas, foi aplicada a técnica do *Undersampling*, diminuindo o número de amostras da classe 0 para o mesmo número de amostras da classe 1.

Foi necessário também criar algumas colunas de dados que não foram geradas automaticamente pelo programa. Como citado na seção anterior, o programa gerador divide as características em dois tipos, *forward* e *backward*, o que permite uma maior flexibilidade ao se trabalhar com os dados, porém exige que alguns cálculos sejam feitos para se gerar certas características mais gerais.

Foi preciso então, criar métodos capazes de trabalhar com os valores presentes para gerar alguns dados que foram utilizados em pesquisas anteriores da área. O Quadro 3 mostra quais são essas características e como elas foram calculadas:

Figura 14 – Exemplo de geração de colunas.

```
def flow_total_bytes():
    total_bytes = []
    for index, data in df.iterrows():

        bytes_forward = data['total_fvolume']
        bytes_backward = data['total_bvolume']

        bytes_sum = bytes_forward + bytes_backward

        total_bytes.append(bytes_sum)

    df['total_bytes'] = total_bytes
```

Fonte: Elaborada pelo autor.

Quadro 3 – Exemplo de geração de colunas.

Característica	Fórmula geradora
Total de Bytes	Soma dos bytes enviados em ambas as direções
Total de pacotes	Soma dos pacotes enviados em ambas as direções
Total de Bits	Número total de bytes multiplicado por 8 (1 byte = 8 bits)
Bytes por pacote	Razão entre o Total de Bytes e Total de pacotes
Bytes por segundo	Total de bits dividido pela duração do fluxo
Pacotes por segundo	Total de pacotes dividido pela duração do fluxo
Média de IAT	Média da soma dos valores IAT médios
Bytes por segundo	Total de bits dividido pela duração do fluxo
Média de variância de IAT	Média dos desvio padrão de IAT ao quadrado
Porcentagem de pacotes enviados	Razão entre o número de pacotes enviado na direção <i>forward</i> e o número total de pacotes do fluxo
IOPR	Razão entre a quantidade de pacotes na direção <i>backward</i> sobre a quantidade da direção <i>forward</i>
Média de tamanho de payload	Número de bytes total do fluxo menos a soma de bytes dos headers em ambas as direções, depois dividido pelo número de pacotes

Fonte: Elaborada pelo autor.

A figura 14 mostra o processo básico para a geração dessas colunas, utilizando como exemplo a coluna "Total de bytes". As linhas da tabela inteira são percorridas, as características necessárias para a geração são obtidas, as operações efetuadas e o resultado é adicionado para uma lista. Por fim, essa lista era adicionada no conjunto de dados como uma nova coluna. O processo é o mesmo para todas as outras características, alterando-se somente os cálculos e valores utilizados.

Por fim, foram removidas algumas colunas em que todos os valores eram 0, ou seja, dados que o gerador não foi capaz de obter nenhuma informação no conjunto de dados. Estas colunas foram as seguintes: Desvio padrão de tempo ativo de conexão, tempo mínimo de ociosidade na conexão, média de tempo de ociosidade na conexão, tempo máximo de ociosidade na conexão, desvio padrão do tempo de ociosidade na conexão, número de vezes em que a *flag Urgent* foi enviada da origem para o destino, e número de vezes em que a *flag Urgent* foi enviada do destino para a origem.

Junto com estas, também foram removidas colunas que não são relevantes para os algoritmos, IP/Porta de origem e de destino e protocolo, que apesar de terem sido muito úteis para a rotulação e filtragem, não serão utilizados para a tarefa de classificação, entre outras.

4.4 Execução dos algoritmos

Com os dados preparados, é possível então usá-los para treinar os algoritmos. Contando todas as colunas válidas do gerador e as características geradas manualmente e desconsiderando

a coluna de rótulos, têm-se 39 colunas de características.

No entanto, não é viável executar os estimadores com todas, mas sim tentar encontrar um subconjunto destas que melhor caracterize o problema. Para este fim, foram aplicadas duas abordagens:

- Busca de força bruta com base em dados de pesquisas anteriores: foram utilizadas somente as características presentes em trabalhos da área, e através de uma pesquisa de força bruta rodar todas as combinações delas com todos os algoritmos e assim encontrar o melhor conjunto de características para cada classificador;
- Busca utilizando um algoritmo específico de seleção de características: o Scikit possui vários algoritmos para seleção de características, que buscam encontrar dado um conjunto a melhor combinação. Para este trabalho, será utilizado o RFE no conjunto inteiro de dados, tanto os do gerador de fluxo quanto os criados com base em estudos anteriores.

4.4.1 Busca de força bruta com base em dados de pesquisas anteriores

Para essa primeira abordagem, a ideia era testar todas os subconjuntos possíveis de características com todos os estimadores, e verificar qual combinação de características seria melhor com cada um deles.

Para obter as combinações de características do conjunto, foi utilizada a biblioteca *itertools* do Python. A figura 15 mostra o processo de geração dos subconjuntos, aonde foi definido um numero mínimo de características por conjunto de 4. O resultado gerado é uma lista contendo todas as combinações. Ao fim do processo, foram geradas 1816 possibilidades.

Figura 15 – Gerando combinações com a biblioteca *itertools*.

```
import itertools
# Lista que guardará todas as combinações
feat_sets = []

# Todas as colunas utilizadas
nomes_colunas = ['duration', 'total_packets', 'iopr', 'total_bytes', 'avg_payload_length', 'bps', 'bpp', 'pps',
                 'pct_packets_pushed', 'var_iat', 'avg_iat', 'mean_active']

# Variando de 4 até o número máximo de características
for T in range(4, len(all_cols)+1):
    # iteração de todos os subsets de tamanho T
    for subset in itertools.combinations(nomes_colunas, T):
        array_nomes = [nome for nome in subset]
        feat_sets.append(array_nomes)
```

Fonte: Elaborada pelo autor.

Depois, todas as combinações foram processadas por todos os estimadores, e as acurácias dos modelos obtidas. Foram criados vetores contendo instâncias de todos os estimadores a serem utilizados no trabalho, e depois este era percorrido, de maneira que a cada iteração o modelo fosse treinado e uma predição fosse obtida, salvando em uma tabela a acurácia e qual conjunto de dados foi usado.

Ao fim deste processo, foi possível filtrar os dados da tabela pelas maiores acurácias de cada estimador, e assim encontrar o subconjunto ótimo para este, assim como a acurácia obtida.

4.4.2 Utilizando o algoritmo Recursive Feature Elimination

A ideia desta abordagem é utilizar o algoritmo RFE em todo o conjunto de dados, e dessa maneira observar quão eficaz são os dados utilizadas historicamente na área na tarefa de classificação, e também identificar se algumas das fornecidas pelo gerador são eficazes nesse processo, através do processo de observação das características escolhidas pelo algoritmo.

Além disso, outro ponto dessa abordagem é verificar se as características utilizadas nos estudos anteriores possuem grande representatividade dado todo o conjunto de características, ou seja, se elas serão escolhidas pelo algoritmo.

O Scikit implementa o RFE em conjunto com a técnica de CV, para todos os números possíveis de parâmetros. Dessa maneira, é possível identificar não só um número fixo de características mas também encontrar qual o número ideal. Para utilizá-lo, passa-se como argumento uma instância do classificador desejado, e depois aplica-se a função *fit* com um conjunto de treino, para que ele encontre o subconjunto ótimo.

Depois de encontrado, é possível acessar tanto o estimador quanto as características do conjunto de maneira ranqueada. Todas as características escolhidas são atribuídas o valor 1, e dessa maneira é possível tanto definir quais as características ótimas quanto já executar previsões com o estimador que a estrutura fornece.

4.4.3 Otimização de Hiper-parâmetros

Obtidos os melhores conjuntos de características para todos os estimadores utilizando as duas abordagens, o último passo a ser feito nesta etapa é aplicar a técnica de *Grid Search* para otimizar hiper-parâmetros no algoritmo de SVM, que como explicado anteriormente na seção 2 é muito depende da otimização de seus hiper-parâmetros para obter melhores resultados.

O Scikit implementa a técnica de *Grid Search* em conjunto com o CV. A interface de utilização é muito parecida com a de um estimador normal, com a diferença de que deve ser passado também um dicionário com os parâmetros que serão testados e seus respectivos valores, como pode ser observado na figura 16.

Foram testados os Kernels Linear, RBF e Sigmoidal. Foram feitas tentativas de cálculo com o Kernel Polinomial, no entanto foram apresentados problemas de execução específicos que impediram a obtenção de resultados.

Foram testados os seguintes valores:

Figura 16 – Utilização do *Grid Search* no Scikit.

```
: param_grid = {'C':[0.1,1,10,100]}  
grid = GridSearchCV(SVC(),param_grid)  
grid.fit(X_train,y_train)
```

Fonte: Elaborada pelo autor.

- C: 0.1,1,10,100;
- γ : 0.0001,0.001,0.01,0.1,1;
- Coeficiente r , específico do Kernel Sigmoidal: -1,0,1.

5 Resultados

5.1 Performance dos Classificadores

As tabelas e gráficos a seguir mostram os resultados que foram obtidos na fase de Desenvolvimento. Nas tabelas 1 e 2, constam os números de características e os valores de acurácia obtidos pelos classificadores ótimos encontrados tanto na abordagem de força bruta quanto na abordagem do RFE, respectivamente. Depois, são apresentados os gráficos 17 e 18 que ilustram as diferenças de valores entre esses dados para as duas abordagens.

Tabela 1 – Resultados da abordagem de Força Bruta

Classificadores	Acurácia	Nº de Características
Árvores de Decisão	0.996604	6
Florestas Aleatórias	0.997910	5
AdaBoost	0.993208	9
NB Multinomial	0.894723	4
NB Bernoulli	0.840125	10
SVC Linear	0.958986	5

Fonte: Elaborada pelo autor.

Tabela 2 – Resultados da abordagem utilizando RFE

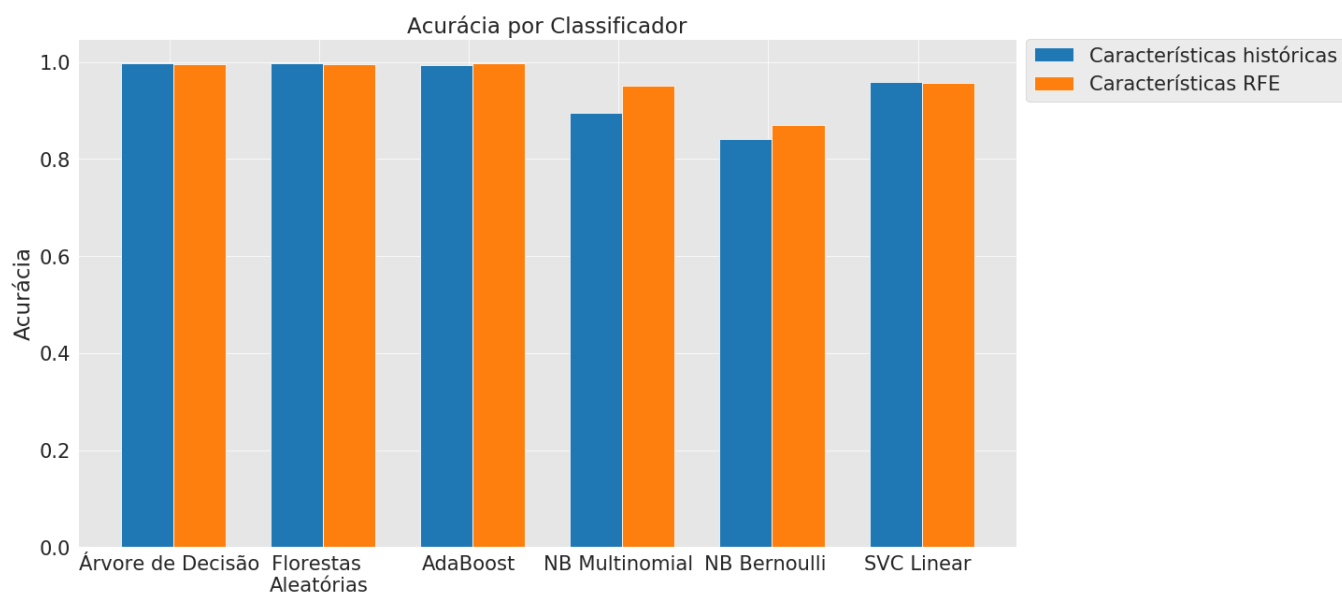
Classificadores	Acurácia	Nº de Características
Árvores de Decisão	0.994775	7
Florestas Aleatórias	0.995559	39
AdaBoost	0.996342	16
NB Multinomial	0.950365	17
NB Bernoulli	0.869905	10
SVC Linear	0.956374	12

Fonte: Elaborada pelo autor.

5.2 Características

Esta seção apresentará alguns gráficos que buscam ilustrar a distribuição de características presentes nas duas abordagens. A figura 19 mostra a quantidade de vezes que uma característica foi utilizada pelos classificadores na abordagem de Força Bruta, enquanto as figuras 20 e 21 mostram as dez características que foram mais e menos utilizadas, respectiva-

Figura 17 – Comparativo de Acurácias.



Fonte: Elaborada pelo autor.

mente, na abordagem RFE. Por fim, o último gráfico da Figura 22 mostra a frequência das características históricas na segunda abordagem de maneira isolada.

5.3 Hiper-parâmetros

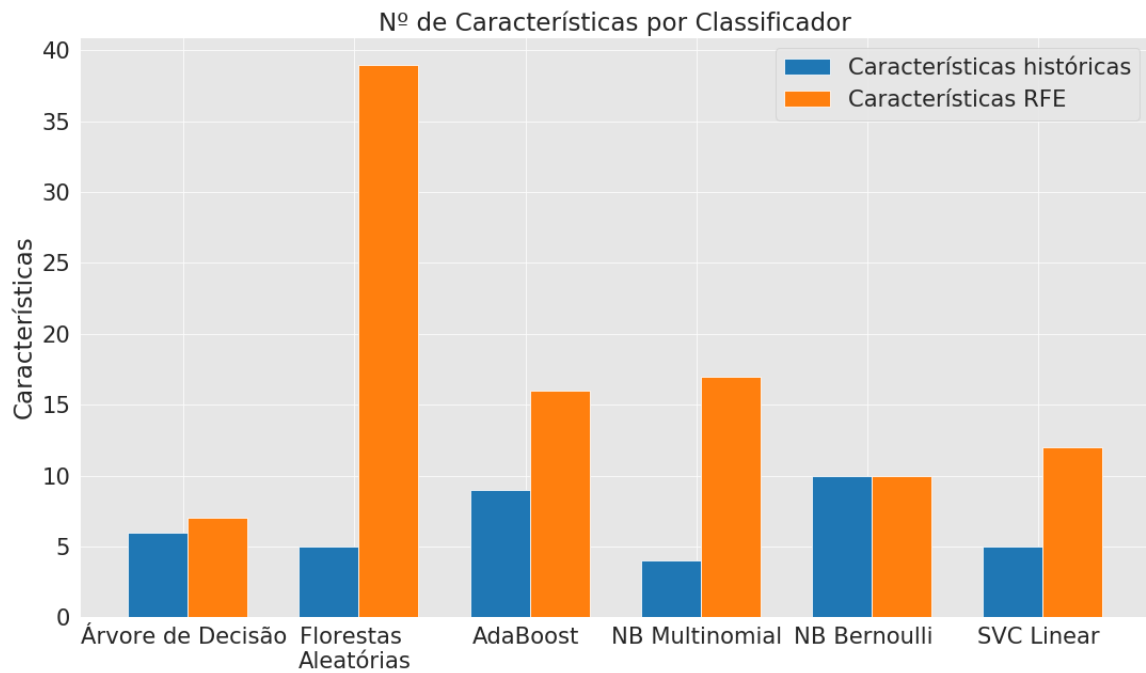
Esta seção irá apresentar os resultados obtidos pela otimização de Hiper-parâmetros feita para o algoritmo de SVM. A tabela 3 contém os valores de acurácia obtidos para cada Kernel, além dos parâmetros ótimos obtidos no processo. Por fim, a figura 23 apresenta a variação na acurácia para cada Kernel.

Tabela 3 – Resultados da otimização de Hiper-parâmetros do SVM

Kernel	Acurácia	Parâmetros
Linear	0.996604	C:100
RBF	0.978840	C: 100, γ : 0.001
Sigmoidal	0.749477	C: 10, γ : 0.0001, r:0

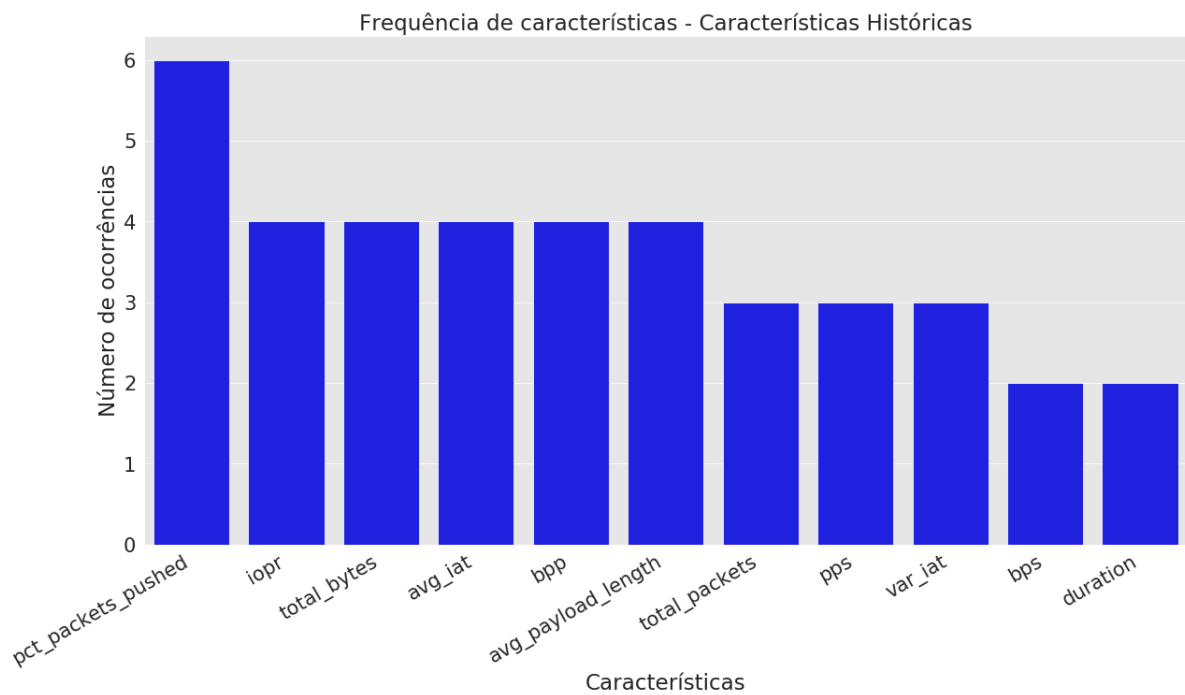
Fonte: Elaborada pelo autor.

Figura 18 – Comparativo de Número de Características.



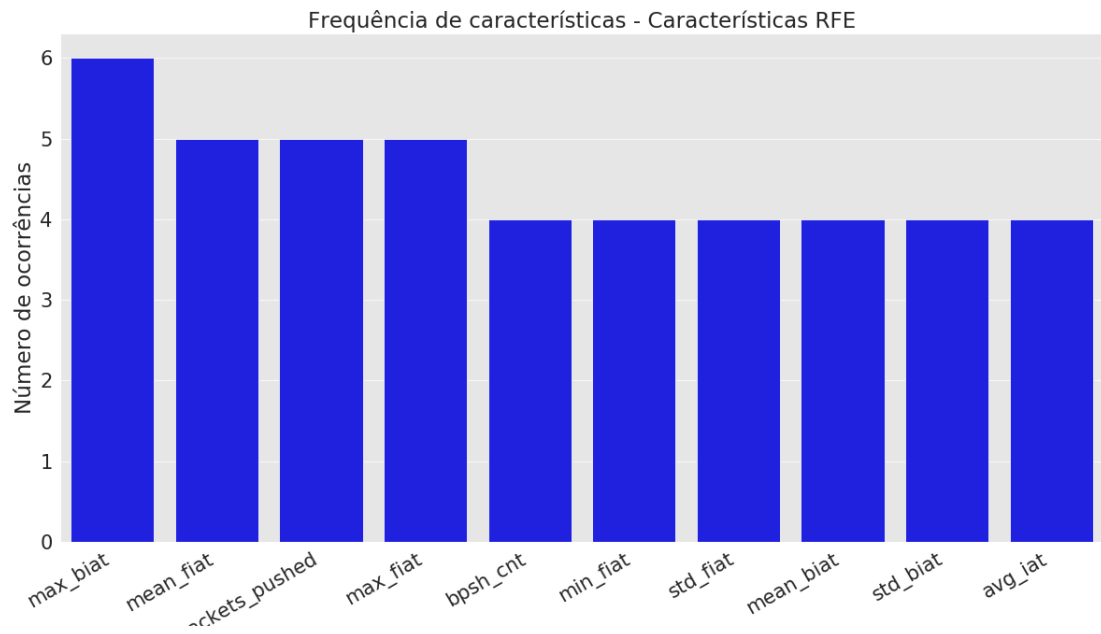
Fonte: Elaborada pelo autor.

Figura 19 – Frequência de Características na abordagem de Força Bruta.



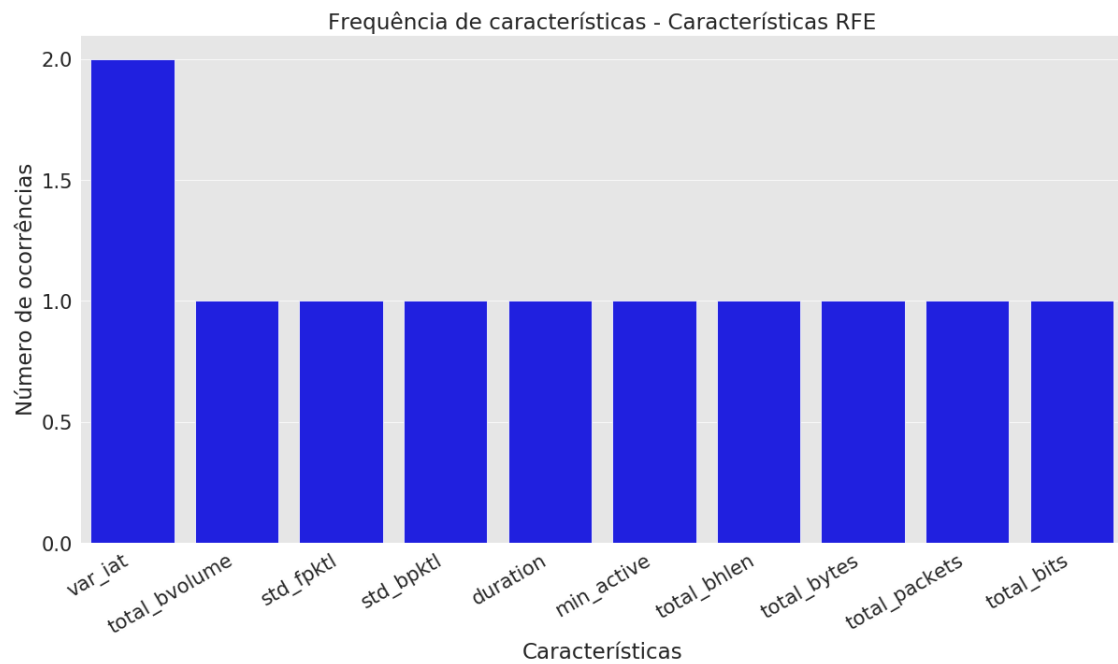
Fonte: Elaborada pelo autor.

Figura 20 – Características mais frequentes na abordagem de RFE.



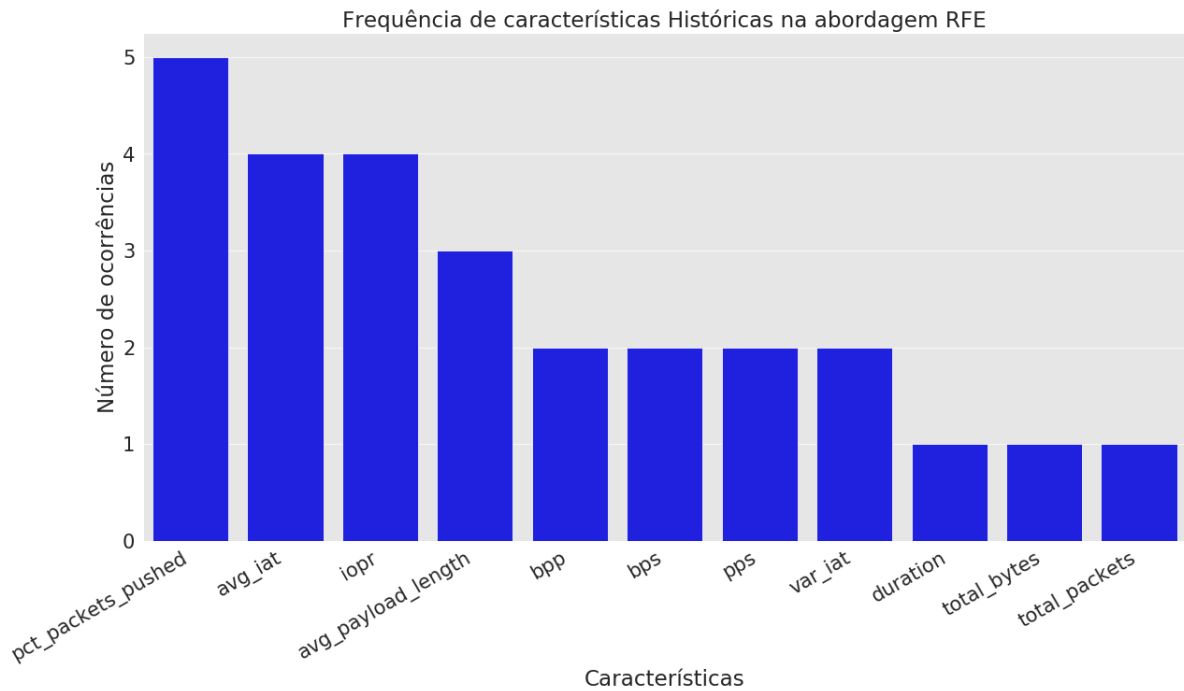
Fonte: Elaborada pelo autor.

Figura 21 – Características menos frequentes na abordagem de RFE.



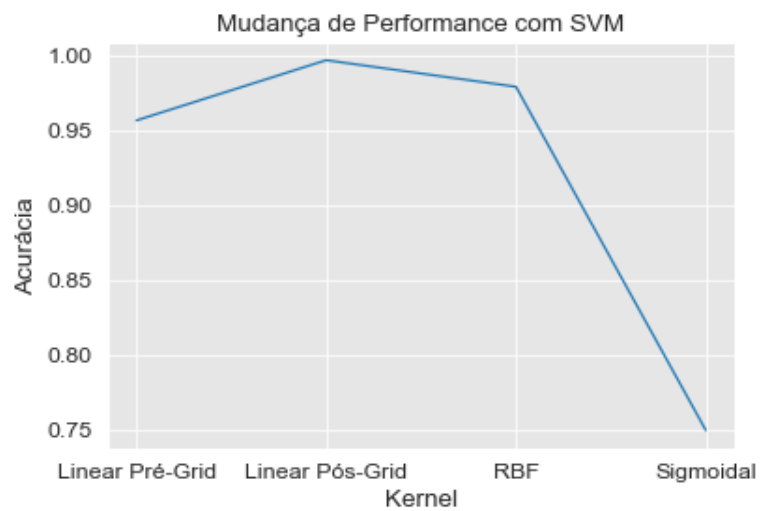
Fonte: Elaborada pelo autor.

Figura 22 – Frequência de Características Históricas na abordagem de RFE.



Fonte: Elaborada pelo autor.

Figura 23 – Comparativo de Acurácia dos Kernels do SVM.



Fonte: Elaborada pelo autor.

6 Conclusão

Este estudo fez uma análise sobre a utilização de técnicas da área de *Machine Learning* para a detecção de *Botnets* baseadas no protocolo IRC em uma rede, através da análise fluxos bidirecionais. Estudos deste tipo se fazem cada vez mais relevante tendo em vista o crescimento do uso de aparelhos conectados em rede que podem ser integrados às *botnets*, aumentando ainda mais o poder de seus ataques.

Sendo assim, é possível afirmar que o presente estudo cumpriu seus objetivos ao estudar a aplicação e eficiência de vários algoritmos da área, além de explorar a utilização de características utilizadas em estudos passados e ainda explorar novas possibilidades de uso.

Com base nos resultados, é possível afirmar que os classificadores testados foram bem sucedidos em suas tarefas, obtendo altos níveis de acurácia. Dentre eles destacam-se os métodos baseados em Árvores, que obtiveram os melhores resultados, seguidos pelos métodos de *Support Vector Machines*, e por último pelos métodos de *Naive Bayes*.

Com relação às duas abordagens de utilização de características, é possível observar que a utilização do RFE para seleção de características impactou diretamente na melhoria da acurácia dos modelos probabilísticos de Naive Bayes, no entanto para os outros métodos os resultados foram parecidos. Porém, a utilização do RFE ainda apresenta menor custo computacional, o que a torna mais efetiva.

Já para o número de características, percebe-se que nenhum deles apresentou uma diferença muito grande com exceção do método de Florestas Aleatórias que acabou utilizando todas as características presentes no conjunto, e assim utilizando 34 características a mais que na abordagem de Força Bruta. Para a primeira abordagem, utilizando 11 características foi obtida uma média de 6,5 características utilizadas por classificador, enquanto para a segunda utilizando 39 características observou-se uma média de 16,8 características.

As características que foram utilizadas nos estudos anteriores se fizeram muito presente nas encontradas pelo RFE, com destaque para algumas que se destacaram em ambas as abordagens: Porcentagem de Pacotes Enviados, IOPR, Média de tamanho do Payload, e o tempo de chegada entre pacotes médio, ou IAT médio.

Observa-se ainda que no que tange a abordagem do RFE, de maneira geral características ligadas a IAT foram amplamente usadas por quase todos os classificadores, enquanto as características de tamanho de fluxo, como total de bytes, não obtiveram muita representatividade em ambas as abordagens, assim como a característica duração do fluxo.

Por fim, foi possível observar o impacto que a otimização de hiper-parâmetros pode ter no desempenho do SVM, tendo em vista que a troca do valor de seu parâmetro C levou a um

considerável aumento em sua acurácia. Além disso, também demonstrou que o Sigmoidal se saiu significativamente pior do que o Linear e o RBF para este tipo de problema.

6.1 Trabalhos Futuros

Como trabalhos futuros baseados neste estudo, têm-se a exploração de outras características possíveis de serem utilizadas na área, avaliar a eficiência dos métodos e características descritos para outros protocolos de *Botnet* centralizadas como o HTTP que vem crescendo em uso recentemente ou descentralizadas que utilizam comunicação P2P, implementação de outras técnicas da área, como Redes Neurais Artificiais ou algoritmos de ML não-supervisionado para encontrar padrões na rede, trabalhar com outras bases de dados, que apresentem maior volume de pacotes, e implementação de um sistema de monitoramento que analise os fluxos em tempo real.

Referências

- AALST, W. M. Van der; RUBIN, V.; VERBEEK, H.; DONGEN, B. F. van; KINDLER, E.; GÜNTHER, C. W. Process mining: a two-step approach to balance between underfitting and overfitting. *Software & Systems Modeling*, Springer, v. 9, n. 1, p. 87, 2010.
- ALEJANDRE, F. V.; CORTÉS, N. C.; ANAYA, E. A. Feature selection to detect botnets using machine learning algorithms. In: *2017 International Conference on Electronics, Communications and Computers (CONIELECOMP)*. [S.l.: s.n.], 2017. p. 1–7. ISSN 2474-9044.
- ALMGREN, M.; JOHN, W. Tracking malicious hosts on a 10gbps backbone link. In: SPRINGER. *Nordic Conference on Secure IT Systems*. [S.l.], 2010. p. 104–120.
- ANDREONI, M. An intrusion detection and prevention architecture for software defined networking. 2014.
- ARNDT, D. *Flowtbag*. [S.l.]: GitHub, 2011. <<https://github.com/DanielArndt/flowtbag>>. Acesso em: 31 out. 2018.
- ASHA, S.; HARSHA, T.; SONIYA, B. Analysis on botnet detection techniques. In: IEEE. *Research Advances in Integrated Navigation Systems (RAINS), International Conference on*. [S.l.], 2016. p. 1–4.
- BAPAT, R.; MANDYA, A.; LIU, X.; ABRAHAM, B.; BROWN, D. E.; KANG, H.; VEERARAGHAVAN, M. Identifying malicious botnet traffic using logistic regression. In: *2018 Systems and Information Engineering Design Symposium (SIEDS)*. [S.l.: s.n.], 2018. p. 266–271.
- BEIGI, E. B.; JAZI, H. H.; STAKHANOVA, N.; GHORBANI, A. A. Towards effective feature selection in machine learning-based botnet detection approaches. In: IEEE. *Communications and Network Security (CNS), 2014 IEEE Conference on*. [S.l.], 2014. p. 247–255.
- BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, v. 13, n. Feb, p. 281–305, 2012.
- BREIMAN, L. Random forests. *Machine Learning*, v. 45, n. 1, p. 5–32, 2001. ISSN 1573-0565. Disponível em: <<https://doi.org/10.1023/A:1010933404324>>. Acesso em: 31 out. 2018.
- BUITINCK, L.; LOUPPE, G.; BLONDEL, M.; PEDREGOSA, F.; MUELLER, A.; GRISEL, O.; NICULAE, V.; PRETTENHOFER, P.; GRAMFORT, A.; GROBLER, J.; LAYTON, R.; VANDERPLAS, J.; JOLY, A.; HOLT, B.; VAROQUAUX, G. API design for machine learning software: experiences from the scikit-learn project. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. [S.l.: s.n.], 2013. p. 108–122.
- CHEN, S.; CHEN, Y.; TZENG, W. Effective botnet detection through neural networks on convolutional features. In: *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. [S.l.: s.n.], 2018. p. 372–378. ISSN 2324-9013.

- CORTES, C.; VAPNIK, V. Support-vector networks. *Machine learning*, Springer, v. 20, n. 3, p. 273–297, 1995.
- DOSHI, R.; APHORPE, N.; FEAMSTER, N. Machine learning ddos detection for consumer internet of things devices. In: *2018 IEEE Security and Privacy Workshops (SPW)*. [S.l.: s.n.], 2018. p. 29–35.
- FEILY, M.; SHAHRESTANI, A.; RAMADASS, S. A survey of botnet and botnet detection. In: IEEE. *Emerging Security Information, Systems and Technologies, 2009. SECURWARE'09. Third International Conference on*. [S.l.], 2009. p. 268–273.
- FLACH, P. A. The geometry of roc space: understanding machine learning metrics through roc isometrics. In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. [S.l.: s.n.], 2003. p. 194–201.
- FREUND, Y.; SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, v. 55, n. 1, p. 119 – 139, 1997. ISSN 0022-0000. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S002200009791504X>>. Acesso em: 31 out. 2018.
- HARRINGTON, P. Machine learning in action. *Shelter Island, NY: Manning Publications Co*, 2012.
- HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007.
- IANELLI, N.; HACKWORTH, A. Botnets as a vehicle for online crime. *CERT Coordination Center*, v. 1, n. 1, p. 28, 2005.
- JAMES, G.; WITTEN, D.; HASTIE, T.; TIBSHIRANI, R. *An Introduction to Statistical Learning: With Applications in R*. [S.l.]: Springer Publishing Company, Incorporated, 2014. ISBN 1461471370, 9781461471370.
- JANG, D.-i.; KIM, M.; JUNG, H.-c.; NOH, B.-N. Analysis of http2p botnet: case study waledac. In: IEEE. *Communications (MICC), 2009 IEEE 9th Malaysia International conference on*. [S.l.], 2009. p. 409–412.
- JORDAN, M. I.; THIBAU, R. The kernel trick. *Lecture Notes*, 2004.
- KAMBOURAKIS, G.; KOLIAS, C.; STAVROU, A. The mirai botnet and the iot zombie armies. In: IEEE. *Military Communications Conference (MILCOM), MILCOM 2017-2017 IEEE*. [S.l.], 2017. p. 267–272.
- KANT, V.; SINGH, E. M.; OJHA, N. An efficient flow based botnet classification using convolution neural network. In: *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*. [S.l.: s.n.], 2017. p. 941–946.
- KIM, E. Everything you wanted to know about the kernel trick. *URI: http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html*, 2013. Acesso em: 31 out. 2018.
- KLUYVER, T.; RAGAN-KELLEY, B.; PÉREZ, F.; GRANGER, B.; BUSSONNIER, M.; FREDERIC, J.; KELLEY, K.; HAMRICK, J.; GROUT, J.; CORLAY, S.; IVANOV, P.; AVILA, D.; ABDALLA, S.; WILLING, C. Jupyter notebooks – a publishing format for reproducible computational workflows. In: LOIZIDES, F.; SCHMIDT, B. (Ed.). *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. [S.l.], 2016. p. 87 – 90.

- LEONARD, J.; XU, S.; SANDHU, R. A framework for understanding botnets. In: IEEE. *Availability, Reliability and Security, 2009. ARES'09. International Conference on*. [S.l.], 2009. p. 917–922.
- LIVADAS, C.; WALSH, R.; LAPSLEY, D.; STRAYER, W. T. Using machine learning techniques to identify botnet traffic. In: IEEE. *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*. [S.l.], 2006. p. 967–974.
- LORENA, A. C.; CARVALHO, A. C. de. Uma introdução às support vector machines. *Revista de Informática Teórica e Aplicada*, v. 14, n. 2, p. 43–67, 2007.
- LU, W.; GHORBANI, A. A. Botnets detection based on irc-community. In: CITESEER. *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*. [S.l.], 2008. p. 1–5.
- MAHARDHIKA, Y. M.; SUDARSONO, A.; BARAKBAH, A. R. An implementation of botnet dataset to predict accuracy based on network flow model. In: *2017 International Electronics Symposium on Knowledge Creation and Intelligent Computing (IES-KCIC)*. [S.l.: s.n.], 2017. p. 33–39.
- MARSH, B. Multivariate analysis of the vector boson fusion higgs boson. 2016.
- MASUD, M. M.; AL-KHATEEB, T.; KHAN, L.; THURASINGHAM, B.; HAMLEN, K. W. Flow-based identification of botnet traffic by mining multiple log files. In: IEEE. *Distributed Framework and Applications, 2008. DFMA 2008. First International Conference on*. [S.l.], 2008. p. 200–206.
- MCKINNEY, W. Pandas: a foundational python library for data analysis and statistics. 2011.
- MEDIUM. *Random Forest Simple Explanation*. 2017. Disponível em: <<https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d>>. Acesso em: 27 set. 2018.
- METSIS, V.; AL. et. Spam filtering with naive bayes – which naive bayes? In: *THIRD CONFERENCE ON EMAIL AND ANTI-SPAM (CEAS)*. [S.l.: s.n.], 2006.
- MILLER, S.; BUSBY-EARLE, C. The impact of different botnet flow feature subsets on prediction accuracy using supervised and unsupervised learning methods. v. 5, p. 474–485, 2016.
- MILLER, S.; BUSBY-EARLE, C. The role of machine learning in botnet detection. In: IEEE. *Internet Technology and Secured Transactions (ICITST), 2016 11th International Conference for*. [S.l.], 2016. p. 359–364.
- MÜLLER, A.; GUIDO, S. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media, Incorporated, 2017. ISBN 9781449369408. Disponível em: <<https://books.google.com.br/books?id=q5pnAQAACAAJ>>. Acesso em: 1 nov. 2018.
- OIKARINEN, J.; REED, D. *Internet relay chat protocol*. [S.l.], 1993.
- OLIPHANT, T. E. *A guide to NumPy*. 2006. [Online; accessed <today>]. Disponível em: <<http://www.scipy.org/>>. Acesso em: 31 out. 2018.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011.

RAJAB, M. A.; ZARFOSS, J.; MONROSE, F.; TERZIS, A. A multifaceted approach to understanding the botnet phenomenon. In: ACM. *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. [S.l.], 2006. p. 41–52.

ROJAS, R. Adaboost and the super bowl of classifiers a tutorial introduction to adaptive boosting. *Freie University, Berlin, Tech. Rep*, 2009.

STEVANOVIC, M.; PEDERSEN, J. Machine learning for identifying botnet network traffic. In: . [S.l.: s.n.], 2013.

STRAYER, W. T.; LAPSELY, D.; WALSH, R.; LIVADAS, C. Botnet detection based on network behavior. In: *Botnet detection*. [S.l.]: Springer, 2008. p. 1–24.

UNIVERSITY OF NEW BRUNSWICK. *Botnet dataset*. 2018. Disponível em: <http://www.unb.ca/cic/datasets/botnet.html>. Acesso em: 27 set. 2018.

YAP, B. W.; RANI, K. A.; RAHMAN, H. A. A.; FONG, S.; KHAIRUDIN, Z.; ABDULLAH, N. N. An application of oversampling, undersampling, bagging and boosting in handling imbalanced datasets. In: SPRINGER. *Proceedings of the First International Conference on Advanced Data and Information Engineering (DaEng-2013)*. [S.l.], 2014. p. 13–22.

ZHANG, H. The optimality of naive bayes. v. 1, n. 2, p. 3, 2004.