Análise Comparativa da performance e custo computacional de Modelos de *Machine Learning* para detecção de ataques DoS e DDoS

Maria Eduarda de O. Silva[1], **Carlos R. Santos-Junior**[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia de São Paulo - IFSP - Campus Hortolândia

maria_oliveira.s@hotmail.com, carlos.rsantos@gmail.com

Abstract. *As a consequence of the Digital Transformation that occurs in the world, has increasing the numbers of cyber attacks. This paper presents a comparative analysis of the computational cost and performance of two* Machine Learning *models used in the detection of DoS (Denial of Service) and DDoS (Distributed Denial of Service) attacks. Which are known for its use by hackers in order to temporarily disable a network interface, sending packages beyond the supported limit. In this way, creating damages for the victim, such as companies, government systems, or even private and domestics network.*

Resumo. *Em consequência da transformação digital no mundo, há o aumento de ataques cibernéticos. Este artigo apresenta uma análise comparativa do custo computacional e performance de dois modelos de Machine Learning utilizados na identificação de ataques DoS (Denial of Service) e DDoS (Distributed Denial of Service). Os quais são conhecidos por sua utilização por hackers no intuito de indisponibilizar temporariamente uma interface de rede, realizando o envio de pacotes além do limite suportado, desta forma gerando prejuízos para a vítima, como empresas, órgãos governamentais, ou até mesmo redes domésticas e privadas.*

1. Introdução

Diante da transformação digital que ocorre no mundo, há também o aumento do volume de dados trafegados na internet, visto que o uso da Internet se tornou cotidiano na vida das pessoas, tanto como um meio de lazer, como também para facilitar atividades do dia a dia. Um dos principais fatores que impulsionou o aumento no tráfego dos dados foi a popularização dos dispositivos de Internet das Coisas (*Internet of Things* - IoT), traduzido como Internet das Coisas. Segundo [Dahlqvist et al. 2019], o número de negócios que utilizam tecnologias IoT aumentou de 13% no ano de 2014, para 25% em 2019, sendo assim um aumento aproximado de 12% em cerca de 5 anos. Porém, isso abre as portas para o avanço na quantidade de ataques cibernéticos, e também a preocupação com a segurança dessa informação que é trafegada por toda a internet. Como citado no site [CERT.br 2017], todo e qualquer equipamento ligado a Internet é um potencial alvo de ataque, e seus donos, de possíveis golpes.

Com isso, a evolução dos mecanismos de segurança em redes de computadores se torna crucial, como também o desenvolvimento de novas técnicas, ferramentas e *softwares* para detecção de ataques [AllEasy 2019].

Ao analisar os problemas possíveis que podem existir, deve-se considerar diversos tipos de golpes e ataques, sendo destes, um dos mais comuns o DDoS (*Distributed Denial of Service*), conhecido por indisponibilizar um serviço, computador ou uma rede conectada à Internet, trazendo diversos prejuízos dependendo do tipo do alvo, sendo esse, por exemplo, uma corporação pública ou privada. Os prejuízos financeiros podem ser altamente consideráveis, a depender da duração do ataque, como citado na matéria [Vieira 2019]. Segundo [Oliveira 2018], dispositivos diretamente ligados ao paradigma de Internet das Coisas têm maior suscetibilidade à indisponibilização dos seus recursos após ataques DDoS. Isso acontece devido a baixa capacidade de armazenamento e processamento da maioria desses dispositivos.

Atualmente, muitas das soluções propostas para automatizar a identificação deste tipo de ataque utilizam técnicas de *Machine Learning* (Aprendizado de Máquina), assim como mencionado em [Idhammad et al. 2018]. Neste caso, é utilizado modelos supervisionados ou não-supervisionados para treinar e predicar dentre uma base de dados de tráfego de uma rede de Internet quais seriam requisições consideradas normais e quais seriam ataques. Avaliando como sucesso o modelo que tiver maiores resultados de métricas, como precisão ou acurácia [Idhammad et al. 2018]. Porém, parte dessas soluções não levam em conta o custo computacional para a execução de cada modelo, assim comparado aos seguintes trabalhos [Das et al. 2019] e [Hoon et al. 2018].

Assim, o objetivo deste trabalho é apresentar a análise do desempenho de 2 modelos de *Machine Learning* considerando o custo computacional, como consumo de memória RAM e tempo gasto na execução dos algoritmos.

Este artigo está organizado da seguinte forma: o tópico 2, nomeado Revisão de literatura, apresenta os trabalhos que foram utilizados de referência para o desenvolvimento deste artigo. Tópico 3, Segurança em redes de computadores e possuindo a seção 3.1 DoS e DDoS, expõem o conceito desses dois assuntos. Tópico 4, apresenta um breve conceito sobre Aprendizado de máquina, e em seguida nos subtópicos 4.1 e 4.2, é realizada uma introdução dos dois modelos escolhidos (*Naive Bayes* e *Decision Tree*) e seus principais parâmetros. No tópico 5, denominado Experimentos, é abordado a metodologia executada neste trabalho, juntamente aos subtópicos, 5.1, Base de dados, 5.2, Pré-processamento, e 5.3, Métricas de performance, nos quais são detalhados cada um dos assuntos. O tópico 6, Resultados e Discussões, traz os detalhes do ambiente no qual foram realizadas as simulações e apresentação dos resultados obtidos, representados respectivamente pelas seções 6.1 e 6.2. Finalizando, o tópico 6 traz as conclusões deste artigo e em seguida o 7, apresenta as perspectivas para trabalhos futuros.

2. Revisão de literatura

Nesta seção é listado trabalhos que serviram de referência para o desenvolvimento deste artigo e que apresentam técnicas de *Machine Learning* para a detecção de ataques DDoS.

Em [Das et al. 2019], é proposto um novo Sistema de Detecção de Intrusão (do inglês *Intrusion Detection System* - NIDS), capaz de detectar diversos tipos de ataques DDoS, incluindo novos e emergentes. Utilizando um conjunto de modelos de *Machine Learning* e realizando a redução das *features* analisadas do *dataset* NSL-KDD, foi possível, juntamente ao mecanismo desenvolvido, uma obtenção de 99.1% de acurácia.

Em [Hoon et al. 2018], é apresentado uma revisão crítica das abordagens utiliza-

das de *Machine Learning* para a detecção dos ataques DDoS, com o objetivo de recomendar o melhor modelo para este propósito. No trabalho, foi realizada a implementação de algoritmos supervisionados e não supervisionados através das ferramentas de *Data Mining* WEKA e H2O. Para obtenção dos resultados, foram realizadas simulações com o *dataset* NSL-KDD e análises baseadas na métrica de acurácia e eficiência, isto é, tempo gasto nos treinamentos. Com isso, foi concluído que os modelos de algoritmos supervisionados foram superiores aos não supervisionados.

Em [Idhammad et al. 2018], é apresentado uma abordagem distinta do uso de algoritmos de *Machine Learning* em relação aos mecanismos de detecção de ataques DDoS. O método utilizado foi baseado em estimação de entropia, *co-clustering*, taxa de ganho de informações e *Extra Trees*, implementados nos seguintes *datasets*, NSL-KDD, UNB ISCX 12 e UNSW-NB15. Com o objetivo de obter redução do número de falso positivos e aumentar a acurácia, foram realizados experimentos utilizando modelos semi-supervisionados, em que foi obtido resultados satisfatórios em comparação aos métodos atuais de detecção de DDoS. Porém, não houve avaliação da abordagem proposta em cenários reais.

3. Segurança em redes de computadores

De acordo com [Jang-Jaccard and Nepal 2014], com o aumento exponencial do uso de tecnologias computacionais, houve também um aumento significante do que hoje é chamado de ataques cibernéticos, sendo estes ações maliciosas que possuem objetivos como: roubos de dados (*Phishing*), sequestro de dados (*Ransomware*) e indisponibilidade de recursos tecnológicos, por exemplo. Consequentemente, ocorreu a necessidade de um maior investimento na área de segurança de redes, para a garantia da confidencialidade, integridade e disponibilidade dos recursos da rede em questão. Para o aperfeiçoamento da segurança, pode se contar com sistemas e ferramentas, tais como: criptografias, *firewall* e também uma infraestrutura robusta.

3.1. DoS (*Denial of Service*) e DDoS (*Distibuted Denial of Service*)

Distibuted Denial of Service (Ataque Distribuído de Negação de Serviço), assim como o *Denial of Service* (Ataque de Negação de Serviço), visa a indisponibilização de um serviço ou rede, através da realização de muitas requisições simultâneas para um único ponto, conforme explicado por [Mitshashi 2011]. Porém no caso do DDoS, ele tem diversos pontos de partida, portanto, são utilizadas várias máquinas chamadas de atacantes, que apontam as requisições para uma única vítima. Um dos principais exemplos desse ataque é o SYN *Flood*. Em que SYN, de acordo com [Hope 2017], é um diminutivo para *Synchronize* e dentro do contexto de Redes de Computadores é uma ação na qual um pacote TCP é enviado de um computador para outro com a tentativa de garantir uma conexão entre os mesmos. E como o próprio nome do ataque indica, é uma inundação de segmentos TCP do tipo "SYN", onde este inicializa diversas conexões mas nunca as fecha, o que leva ao serviço ser indisponibilizado devido o alto consumo de recursos da máquina como, memória RAM e tempo de CPU, para manter tais conexões.

4. Machine Learning (Aprendizado de Máquina)

Conforme citado por [Cerri and Carvalho 2017], o Aprendizado de Máquina ou *Machine Learning* (ML), é uma área de estudo dentro de Inteligência Artificial e que tem sido cada

vez mais utilizado para resoluções de problemas. Nesta área, é visado o desenvolvimento de programas ou algoritmos capazes de aprenderem e executarem tarefas baseados em suas próprias experiências, isto é, eles são capazes de aprenderem por si só utilizando dados que representem essas experiências prévias, em que quando aplicado à prática, podemos chamar de conjunto de dados para treinamento [Cerri and Carvalho 2017].

Existem 3 tipos conhecidos de aprendizados: o Supervisionado, que consiste em produzir uma função capaz de mapear novos cenários com base nas informações adquiridas de um conjunto de amostras de treinamento, como por exemplo a identificação de *e-mails* considerados como *spam*. A partir de certas características que este tipo de mensagem contém, a aplicação é capaz de identificar novos cenários semelhantes e os classificar da forma correta, conforme apresentado em [Silva 2021]. O Aprendizado Não-Supervisionado, que ao contrário do Supervisionado, consiste em identificar padrões em dados não classificados e agrupá-los com base em suas semelhanças ou diferenças [Silva 2021]. Já o Aprendizado por Reforço, pode ser representado como sendo uma proposta de tentativa e erro, em que cada ação errada ou correta leva o algoritmo a adicionar pesos distintos. Sendo assim, o objetivo é conseguir atingir o melhor resultado, que é representado pela maior recompensa acumulativa dos pesos [Silva 2021], por exemplo, encontrar uma melhor rota para chegar de um ponto ao outro de uma cidade.

No Aprendizado de Máquina é possível encontrar ao menos 3 tarefas principais, que são escolhidas com base nos problemas que se quer resolver: Classificação, Regressão (ambos sendo do tipo de Aprendizado Supervisionado) e Clusterização (Não-Supervisionado). No problema de Classificação, é realizada a atribuição de um valor/categoria pré-definida ao dado, como por exemplo um algoritmo que foi desenvolvido para identificar um cliente que pode ou não receber um empréstimo. Nesse caso, os valores possíveis de resposta são: sim e não [Cerri and Carvalho 2017]. Já o de Regressão, tem como objetivo prever os valores de saída. Como por exemplo, prever o preço de algum produto ou até mesmo valores de ações [Cerri and Carvalho 2017]. Por último o de clusterização, ou agrupamento como também é conhecido. Como o próprio nome indica, realiza o agrupamento de dados para identificar padrões ou diferenças entre eles com o objetivo de realizar a categorização [Silva 2021].

Neste trabalho, foram escolhido dois modelos supervisionados e também utilizado a tarefa de classificação para a realização das simulações. Abaixo será apresentado uma breve explicação de cada um e seus principais parâmetros.

4.1. *Naive Bayes*

Este modelo é baseado no teorema de Bayes, que de forma resumida se baseia na teoria das probabilidades. Pode se dizer que o mesmo, a partir dos dados de treinamento aprende a probabilidade condicional de cada um dos atributos do *dataset*, dado o valor das classes, conforme citado em [WANKE et al. 2014].

Os principais parâmetros desse modelo, os quais estão descritos em sua documentação oficial [Pedregosa et al. 2011] são:

- *priors* (não é obrigatório, mas se informado, deve ser do tipo *array* e representa uma lista de probabilidades já identificadas das classes, dessa forma elas não serão reajustadas durante a execução do algoritmo e seu valor pré-definido é *None*);

- *var_smoothing* que é o valor da maior variância possível de cada atributo, utilizada para a estabilidade do cálculo, seu valor pré-definido é 1e-9.

Nenhum dos parâmetros mencionados são obrigatórios para a execução do modelo, sendo assim, neste trabalho foram usados os valores pré-definidos para os dois casos.

4.2. *Decision Tree* ou Árvore de Decisão

Decision Tree ou também conhecido como Árvore de Decisão, é um modelo que utiliza o conceito de Árvore da computação para suas predições. Neste caso, a Árvore representa um conjunto de elementos que armazenam dados e são conhecidos por "nós". Em seguida, cada um desses nós são divididos em subpartes (ou sub-nó), com o objetivo de simplificar o problema a cada ramificação. De forma recursiva, o modelo vai tomando as melhores decisões. Sua vantagem é alta readaptação, onde o modelo está em constante evolução para encontrar sempre o melhor caminho. [Jijo and Mohsin Abdulazeez 2021]

De acordo com a documentação oficial [Pedregosa et al. 2011], nesse modelo é possível encontrar 13 parâmetros possíveis de serem configurados, sendo estes: *criterion, splitter, max_depth, min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_features, random_state, max_leaf_nodes, min_impurity_decrease, min_impurity_split, class_weight, ccp_alpha*. Porém, neste trabalho e apenas para o *class_weight* que foi informado um valor distinto do pré-definido, sendo este o "*balanced*".

5. Experimentos

Nessa seção, serão apresentadas as simulações que foram realizadas utilizando a base NSL-KDD em conjunto com os modelos de *Machine Learning* citados na seção acima.

Para analisar o desempenho dos algoritmos, foi selecionada uma base de dados pública formada por amostras de pacotes normais e pacotes coletados durante ataques a uma rede de computadores. Todas as amostras foram rotuladas como Normal e Ataque. No caso de ataque, o rótulo corresponde ao tipo de ataque. No que diz respeito ao *dataset*, será tratado como subclasse nesse trabalho. Todos os detalhes da base de dados utilizada são detalhados na seção 5.1.2.

Os testes realizados consistem em avaliar a capacidade de um algoritmo de *Machine Learning* de aprender a distinguir se uma amostra se trata ou não de um ataque. Para isso, os algoritmos serão treinados com parte dos dados rotulados e em seguida seu desempenho avaliado através das métricas *Precision*, *Recall*, *F1-Score* e custo computacional, sendo estes tempo de processamento e consumo de memória RAM. Já na fase, chamada de teste, os dados são enviados aos algoritmos sem os rótulos de identificação.

5.1. Base de dados

Entre as bases de dados mais populares para estudos relacionados ao tráfego de redes de internet e disponíveis de forma pública, existem os *datasets* KDD99 Cup e NSL-KDD, os quais auxiliam pesquisadores na identificação dos melhores métodos para detecção de ataques cibernéticos [Devi and Abualkibash 2019]. Abaixo será apresentado as duas bases, com foco na utilizada neste trabalho, NSL-KDD, e suas principais características.

5.1.1. KDD99 Cup

KDD99 Cup é um *subset* do *dataset* DARPA98 [Devi and Abualkibash 2019], o qual foram coletados os dados puros e a partir disso extraídos atributos [Özgür and Erdem 2016]. Como resultado final, foi apresentado o novo conjunto de dados chamado KDD99, que desde então tem sido usado em diversos treinamentos de algoritmos para detecção de ataques cibernéticos [Özgür and Erdem 2016]. O mesmo possui 41 atributos, sendo estes, informações de uma conexão de rede, onde parte de seus valores são categóricos ou numéricos. O conjunto de dados KDD99 possui aproximadamente 4.8 milhões de amostras de pacotes, as quais estão categorizadas em 5 classes [Devi and Abualkibash 2019] e subdividas em 38 subclasses (tipos de ataque) em toda a base de dados. Na Tsuvidaabela 1 é possível visualizar a relação de cada subclasse e sua classe.

Tabela 1. Relação de classe e subclasse do KDD99 [Archive 1999]

Classe	**Subclasse**
DoS	*back*
DoS	*land*
DoS	*neptune*
DoS	*pod*
DoS	*smurf*
DoS	*teardrop*
Probe	*nmap*
Probe	*portsweep*
Probe	*ipsweep*
Probe	*satan*
R2L	*ftp_write*
R2L	*guess_passwd*
R2L	*imap*
R2L	*multihop*
R2L	*phf*
R2L	*spy*
R2L	*warezclient*
R2L	*warezmaster*
U2R	*butter_overflow*
U2R	*loadmodule*
U2R	*perl*
U2R	*rootkit*

Apesar do KDD99 fornecer os dados para diversos trabalhos, o mesmo apresenta pontos de melhorias que dificultam as análises dos experimentos realizados, o principal deles é a quantidade de dados redundantes e duplicados, que representam boa parte do conteúdo. Se removidos, o número total de dados é reduzido de 4.8 milhões para aproximadamente 1 milhão de registros [Devi and Abualkibash 2019]. A Tabela 2 apresenta a distribuição das amostras de cada classe presentes nas porções de dados reservados para o treinamento e teste dos algoritmos de Aprendizado de Máquina após a remoção destes dados redundantes e duplicados.

É possível identificar ainda na Tabela 2, o desbalanceamento do número de amostras entre as classes. A classe U2R, por exemplo, possui somente 52 e 228 amostras nos conjuntos de dados reservados para o treinamento e testes, respectivamente. Dado a dimensão do conjunto de dados, a classe U2R representa cerca de 0.01% do total de amostras do conjunto de treinamento, dificultando a detecção dos ataques pertencentes a esta classe pelos modelos de Aprendizado de Máquina.

Tabela 2. Distribuição do KDD99 entre dados de treinamento e teste após a remoção de dados duplicados [Devi and Abualkibash 2019]

Classes	Dados de Treinamento	Percentual	Dados de Teste	Percentual
Normal	812.814	75,61%	60.593	19,48%
DoS	247.267	23,00%	29.853	73,90%
Probe	13.860	1,29%	4.166	1,34%
R2L	999	0,09%	16.189	5,21%
U2R	52	0,01%	228	0,07%
Total	1.074.992	100%	311.029	100%

5.1.2. NSL-KDD

Devido aos problemas relacionados à redundância e duplicidade dos dados no *dataset* KDD99 como por exemplo, alto custo computacional para o processamento dos dados e influência nas avaliações de desempenho, foi proposto o *dataset* NSL-KDD, conforme mencionado em [Revathi and Malathi 2013]. No novo *dataset*, foi removida a redundância e duplicidade dos dados, porém mantida a relação entre as quantidades de amostras por classe, subclasse e atributos. Na Tabela 4 são apresentadas as descrições dos atributos do *dataset* NSL-KDD.

O *dataset* NSL-KDD está disponível para fins acadêmicos no site da Universidade do Novo Brunswick (University of New Brunswick - Canada) [for Cybersecurity], de forma já particionada entre dados de treinamento e teste.

5.2. Pré-processamento

Um processo de extrema importância que precisa ocorrer antes do início do treinamento dos modelos é a realização de métodos de pré-processamento da base de dados. De acordo com o Centro de arquitetura do Google Cloud [de arquitetura do Cloud 2018], fazem parte dessa etapa os seguintes métodos:

- **Exclusão de dados:** essa ação consiste em remover registros da base de dados que estão duplicados, incompletos e/ou corrompidos;
- **Seleção de instâncias e particionamento:** esse método tem o objetivo de, realizar a repartição da base para obter amostras válidas para realizar o treinamento, teste e avaliação (quando necessário) do modelo;
- **Ajuste de atributos:** Nesta ação pode estar incluído algumas atividades como: normalização de valores numéricos; inserção de valores pré-definidos onde não há algum; redução ou remoção de *outliers* e, por fim, ajuste na distribuição dos valores. Neste caso, é comum a execução de uma padronização dos valores dentro

de uma faixa de 0 à 1 ou de -1 à 1, como pode ser visto de forma prática em [Vaz 2019].

- **Transformação de representação:** nesta etapa, é realizada a conversão de atributos do tipo texto em inteiros e também o inverso. Essa necessidade ocorre porque alguns modelos só são capazes de utilizar 1 dos tipos de variáveis, ou até mesmo quando o modelo é capaz de lidar com ambos, não realizar essa transformação pode causar certas tendências dentro do modelo, visto que os atributos não estão padronizados;
- **Extração de atributos:** essa ação consiste na redução da dimensão dos dados com o uso de técnicas como, PCA (*Principal Component Analysis*) [Keboola 2020], *Hashing*, entre outras formas de redução de dimensionalidade;
- **Seleção de atributos:** selecionar somente os atributos que façam sentido para o contexto do trabalho que precisa ser realizado. Este método requer uma análise e entendimento dos atributos da base. Ele pode ser realizado com o uso de filtros ou até mesmo a remoção direta de certos atributos;
- **Construção de atributos:** consiste em realizar a construção de novos atributos utilizando técnicas como a expansão polinomial e cruzamento de atributos.

Alguns dos métodos citados acima, como a Exclusão de dados e Ajuste de atributos, foram inclusive aplicados na base de dados KDD, a qual originou a base escolhida para este trabalho (NSL-KDD), conforme citado por [Choudhary and Kesswani 2020].

Neste trabalho, foram utilizados dois destes métodos, sendo estes: Ajuste de atributos e Transformação de representação.

5.3. Métricas de performance

Antes de retirar métricas a partir de cada modelo é necessária a criação de uma Matriz Confusão. Nela é apontado os acertos e erros do modelo dado o resultado esperado. A Tabela 4 apresenta um exemplo prático da Matriz Confusão.

Tabela 3. Matriz Confusão

	Previsto Positivo	**Previsto Negativo**
Real Positivo	Verdadeiro Positivo (VP)	Falso Negativo (FN)
Real Negativo	Falso Positivo (FP)	Verdadeiro Negativo (VN)

Os Verdadeiros Positivos são as classificações corretas do modelo; os Falsos Negativos são os casos em que o modelo deveria ter detectado como Sim mas detectou como um Não; os Falsos Positivos é quando algo é detectado como Sim mas deveria ser Não; e por último, os Verdadeiros Negativos são quando o modelo classifica corretamente algo que deveria ser Não.

Baseado nessas classificações, é possível tirar métricas de performance como precisão, *Recall*, acurácia e *f1-score*. Nos subtópicos 5.3.1, 5.3.2, 5.3.3 e 5.3.4, respectivamente, será descrito o que cada uma desses métricas representam e como são calculadas.

Tabela 4. Relação de todos os atributos contidos no NSL-KDD e KDD99 Cup [Staudemeyer and Omlin 2014]

Nome	**Descrição**
duration	Duração da conexão em segundos
protocol_type	Protocolo da conexão (tcp, udp, icmp)
service	Porta de destino mapeada para o serviço (ex: http, ftp...)
flag	Estado da conexão normal ou erro
src_bytes	Número de dados em bytes da origem para o destino
dst_bytes	Bytes do destino para origem
land	1 se a conexão é do/para o mesmo hospedeiro/porta, senão 0
wrong_fragment	Número de fragmentos errados (valores: 0, 1, 3)
urgent	Número de pacotes urgentes
hot	Número de indicadores "quentes"
num_failed_logins	Número de tentativas de *login* falhadas
logged_in	1 se o *login* foi realizado com sucesso, senão 0
num_compromised	Número de condições comprometidas
root_shell	1 se é obtido o "administrador"do Shell, senão 0
su_attempted	1 se o comando "su root"foi realizado com sucesso, senão 0
num_root	Número de acessos administrador
num_file_creations	Número de operações de criação de arquivos
num_shells	Número de comandos Shell
num_access_files	Número de operações no acesso de controle de arquivos
num_outbound_cmds	Número de comandos de saída em uma sessão ftp
is_hot_login	1 se a tentativa de login pertece à lista de "quentes"(ex: administrador), senão 0
is_guest_login	1 se o login é de convidado (ex: convidado, anônimo), senão 0
count	Número de conexões para o mesmo hospedeiro que o atual (conexão nos últimos dois segundos)
srv_count	Número de conexões para o mesmo serviço que o atual (conexão nos últimos dois segundos)
serror_rate	% de conexões que contenham erros do tipo "SYN"
srv_serror_rate	% de conexões que contenham erros do tipo "SYN"
rerror_rate	% de conexões que contenham erros do tipo "REJ"
srv_rerror_rate	% de conexões que contenham erros do tipo "REJ"
same_srv_rate	% de conexões para os mesmos serviços
diff_srv_rate	% de conexões para serviços diferentes
srv_diff_host_rate	% de conexões para diferentes hospedeiros
dst_host_count	Contagem de conexões com o mesmo hospedeiro de destino
dst_host_srv_count	Contagem de conexões com o mesmo hospedeiro de destino e usando o mesmo serviço
dst_host_same_srv_rate	% de conexões contendo mesma porta de destino e usando o mesmo serviço
dst_host_diff_srv_rate	% de serviços distintos no atual hospedeiro
dst_host_same_src_port_rate	% de conexões para o hospedeiro atual contendo mesma porta de origem
dst_host_src_diff_host_rate	% de conexões para o mesmo serviço vindo de diferentes hospedeiros
dst_host_serror_rate	% de conexões para o hospedeiro atual que contém erro S0
dst_host_srv_serror_rate	% de conexões para o hospedeiro atual e serviço específico que contém erro S0
dst_host_rerror_rate	% de conexões para o hospedeiro atual que contém erro RST
dst_host_srv_rerror_rate	% de conexões para o hospedeiro atual e serviço específico que contém erro RST

5.3.1. Precisão

A métrica de Precisão apresenta uma porcentagem de acertos em relação ao número total de amostras classificadas pelo modelo como Positivo. Essa métrica é calculada pela seguinte fórmula:

$$Precisao = \frac{VP}{VP + FP}$$

5.3.2. Recall

O *Recall* é a medida que irá dizer a porcentagem de acertos em relação ao total de amostras existentes de mesmo rótulo. Seu cálculo é realizado à partir da seguinte fórmula:

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

5.3.3. Acurácia

A métrica de acurácia equivale a porcentagem de acertos em relação ao total de amostras. O cálculo é realizado à partir da fórmula a seguir:

$$Acuracia = \frac{VP + VN}{Total}$$

5.3.4. F1-Score

É a média ponderada do valor da Precisão e do *Recall*, calculado à partir da seguinte fórmula:

$$f1score = 2 * \frac{Precisao * Recall}{Precisao + Recall}$$

6. Resultados e discussões

Nesta seção, serão apresentados os resultados das simulações que foram realizadas utilizando uma porção da base NSL-KDD, obtida no site oficial da Universidade do Novo Brunswick, a qual foi mencionada na subseção 5.1.2, em conjunto com os algoritmos de *Machine Learning*, como também os detalhes da configuração do ambiente em que foram executadas. Os números finais das amostras, os quais foram aplicados os modelos, estão apresentados na Tabela 5.

Tabela 5. Distribuição da porção de dados utilizados nas simulações

Classes	Dados de Treinamento	Percentual	Dados de Teste	Percentual
Normal	67.342	53,46%	9.711	51,67%
DoS	45.927	36,460%	5.740	30,54%
Probe	11.656	9,25%	1.106	5,89%
R2L	995	0,79%	2.199	11,70%
U2R	52	0,04%	37	0,20%
Total	125.972	100%	18.793	100%

Para avaliar o desempenho dos algoritmos *Naive Bayes* e *Decision Tree* foram utilizadas as 4 métricas citadas na seção anterior (foco na Precisão, Acurácia e *Recall*) e o custo computacional de cada modelo, sendo este o tempo de processamento e consumo de memória RAM.

6.1. Ambiente

As simulações foram realizadas em uma máquina com as seguintes características: Memória RAM: 16 GiB, Processador: Intel® Core™ i5-10310U CPU @ 1.70GHz × 8, Placa Gráfica Integrada: Mesa Intel® UHD Graphics (CML GT2), SSD: 256 GB, IDE: Visual Studio Code e S.O.: Linux com distribuição Ubuntu (20.04.3 LTS).

Para a preparação do ambiente, onde foram instaladas as dependências para a execução do código, foi utilizado o gerenciador de pacotes Conda, o qual foi necessário para a instalação das bibliotecas fundamentais para a reprodução deste trabalho, tais quais: *pandas* (1.2.4): utilizado para manipulação dos *datasets*; *sklearn* (0.24.2): utilizado para aplicação dos algoritmos; *scipy* (1.6.3): utilizado para transformação do formato dos dados do *dataset*; *matplotlib* (3.4.3): utilizado para geração de gráficos de barra e dispersão; *seaborn* (0.11.2): utilizado na geração do gráfico de *heatmap* da Matriz Confusão; *tracemalloc* (3.4.0): utilizado para obtenção dos números de consumo de memória RAM.

6.2. Resultados

A fim de possibilitar uma análise comparativa entre os algoritmos apresentados, optou-se por apresentar os resultados a partir de gráficos de barras e dispersão. A primeira parte compara as métricas de avaliação. Em seguida, a métrica *Recall* é utilizada para apresentar o desempenho dos algoritmos por classe e subclasse, visto que, a métrica *Recall* é amplamente utilizada na literatura em problemas denominados como detecção de anomalias, caracterizado por lidar com bases de dados desbalanceadas como a NSL-KDD utilizada nesse trabalho. Dessa forma, optou-se pela métrica *Recall* para apresentar parte dos resultados. Por fim, gráficos de dispersão apresentam a relação entre *Recall*, tempo de processamento e consumo de memória RAM.

6.2.1. Performance dos algoritmos

Para apoiar no entendimento de certos comportamentos identificados nos resultados obtidos das simulações, foi realizada uma análise acerca da quantidade de amostras disponíveis para cada subclasse, esses números podem ser consultados na tabela 6.

Conforme apresentado na Figura 1, o modelo *Decision Tree* apresentou melhores resultados nas três métricas. No entanto, os dois modelos mantiveram as mesmas proporções de acertos quando comparado às métricas avaliadas. Isso demonstra que ambos os modelos possuem a mesma capacidade de distinção entre as classes positiva e negativa. Ainda, o fato da precisão dos modelos terem menores taxas que as outras duas métricas indica maior dificuldade na classificação da classe negativa. Dessa forma, a taxa de falsos positivos tende a ser maior entre as taxas de erros.

O *Recall* para cada classe de ataque é apresentado na Figura 2. Na qual aponta a classe Probe como a classe com maior taxa de detecção para os dois algoritmos. Isso se deve, provavelmente, à quantidade de amostras da classe e à proporção do número de

amostras presentes nas bases de treino e teste, sendo a quantidade disposta na parte de teste equivalente a 10% das amostras de treinamento da mesma, o que se mostra uma boa proporção para treino e teste.

Tabela 6. Quantidade de amostras por subclasse e conjunto de dados do NSL-KDD

Subclasse	**Base de treino (qtd)**	**Base de teste (qtd)**
normal	67342	9711
neptune	41214	4656
satan	3633	735
ipsweep	3599	141
portsweep	2931	157
smurf	2646	665
nmap	1493	73
back	956	359
teardrop	892	12
warezclient	890	0
pod	201	41
guess_passwd	53	1231
buffer_overflow	30	20
warezmaster	20	944
land	18	7
imap	11	1
rootkit	10	13
loadmodule	9	2
ftp_write	8	3
multihop	7	18
phf	4	2
perl	3	2
spy	2	0

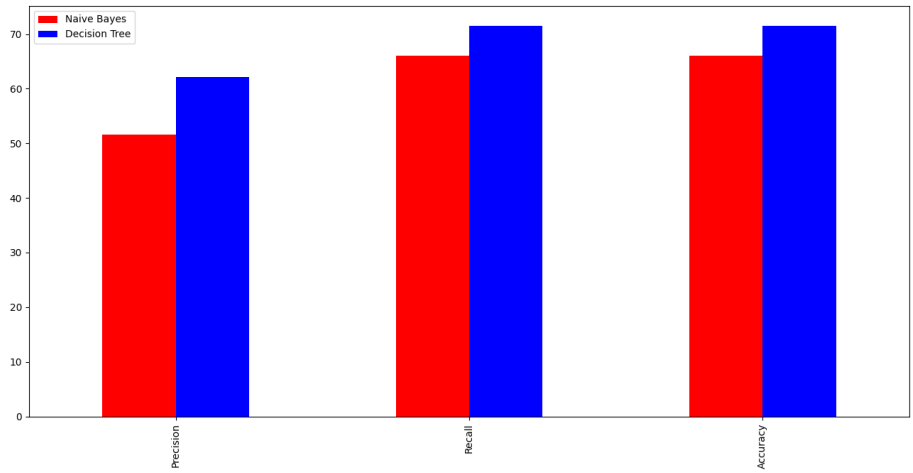

Figura 1. *Precision*, *Recall* e Acurácia geral dos modelos *Naive Bayes* e *Decision Tree*.

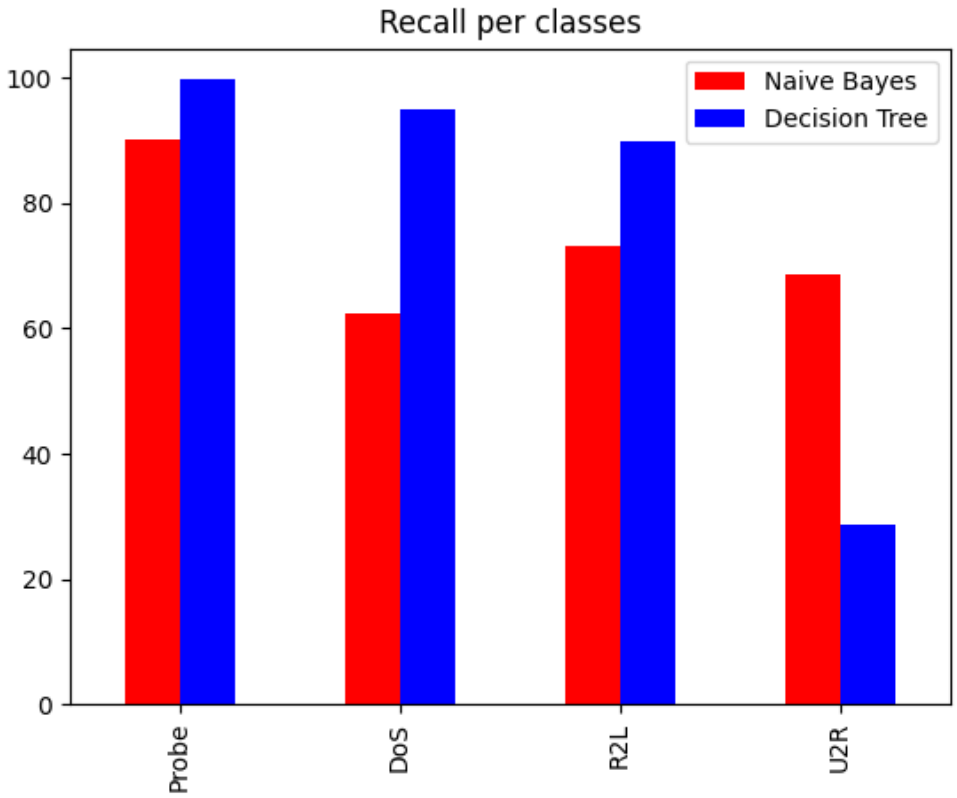

Figura 2. *Recall* por classes de ataques

Já para a classificação das amostras pertencentes a classe DoS, ainda com relação a Figura 2, o modelo *Naive Bayes* apresentou dificuldades em classificar de forma correta. Considerando que existem números suficientes de amostras na base de treinamento, esse resultado sugere uma deficiência do algoritmo *Naive Bayes* para aprendizagem dessa classe. Diferente da classe DoS, a classe U2R possui o menor número de amostras na base de treino. No entanto, foram suficientes para o bom desempenho do algoritmo *Naive Bayes*. Já o algoritmo *Decision Tree* obteve seu pior resultado ao classificar a classe U2R, demonstrando dificuldade quando há poucas amostras na base de treino. A classe R2L, apesar de apresentar um alto grau de desbalanceamento, o número de amostras presentes no treino forneceram informações suficientes para o bom desempenho dos modelos na fase de testes.

A Figura 3 apresenta o *Recall* das subclasses que tiveram suas características melhor aprendidas pelo algoritmo *Decision Tree* e, portanto, as melhores taxas de *Recall* apresentadas foram alcançadas por esse algoritmo.

Já as subclasses apresentadas na Figura 4 tiveram suas amostras detectadas na fase de testes 100% das vezes pelo algoritmo *Decision Tree*. Outras subclasses também foram 100% detectadas, no entanto para melhor apresentação do gráfico optou-se apenas por cinco delas.

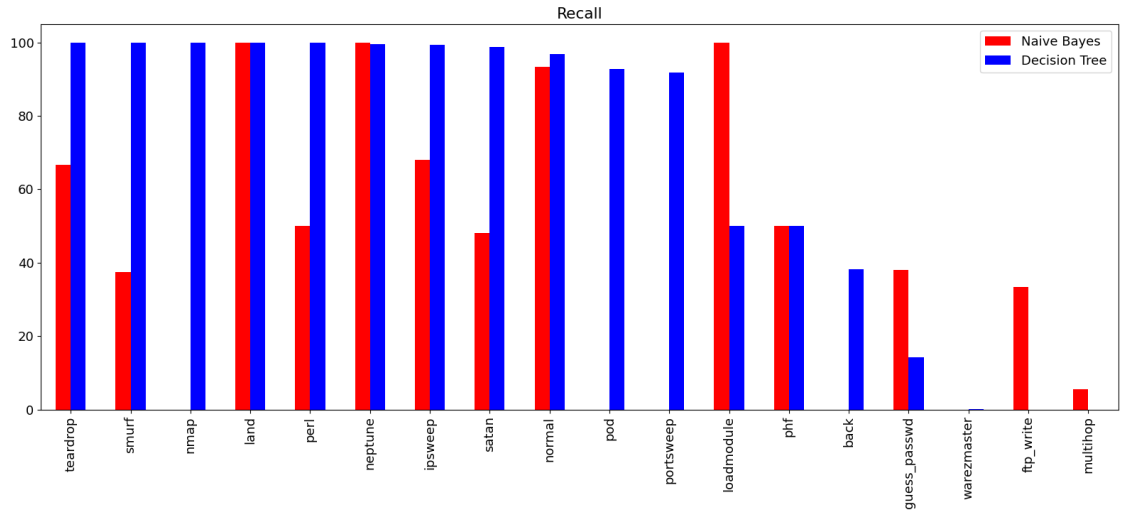

Figura 3. *Recall* por subclasses

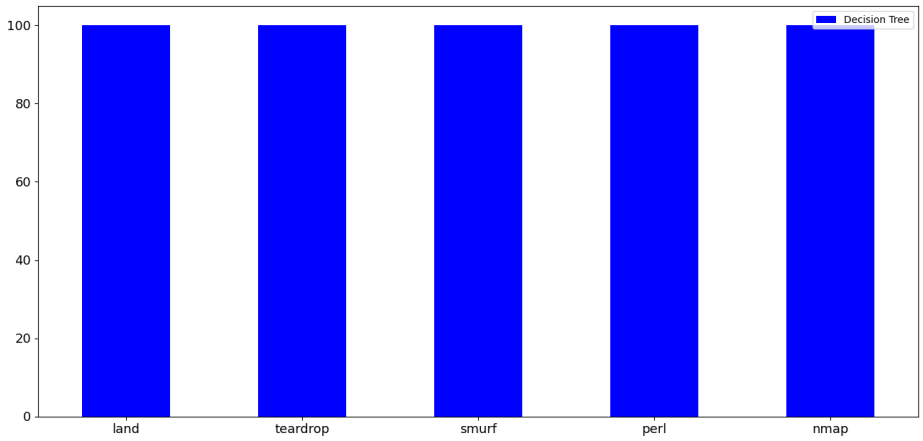

Figura 4. Subclasses com maior *Recall* alcançados pelo algoritmo Decision Tree

Na Figura 5 são apresentadas as subclasses com maiores taxas de *Recall* alcançadas pelo algoritmo *Naive Bayes*. Diferente do *Decision Tree*, apenas as subclasses *land*, *loadmodule* e *neptune* tiveram todas suas amostras detectadas. Já *ipsweep* e *teardrop* tiveram aproximadamente 65% das amostras detectadas na fase de teste.

Entre as subclasses presentes nas Figuras 4 e 5, somente a *land* aparece em ambos. Possivelmente a causa desta ocorrência seja o baixo número de amostras dessa subclasse em comparação às outras, sendo assim cada pequeno acerto gerou uma grande taxa de *Recall*. Entretanto, observa-se também a subclasse *neptune* com alta taxa de acerto e considerando o número de amostras dessa subclasse na base de treino, é possível assumir que com alto volume de dados de treino o algoritmo *Naive Bayes* é capaz de obter bons resultados durante a classificação.

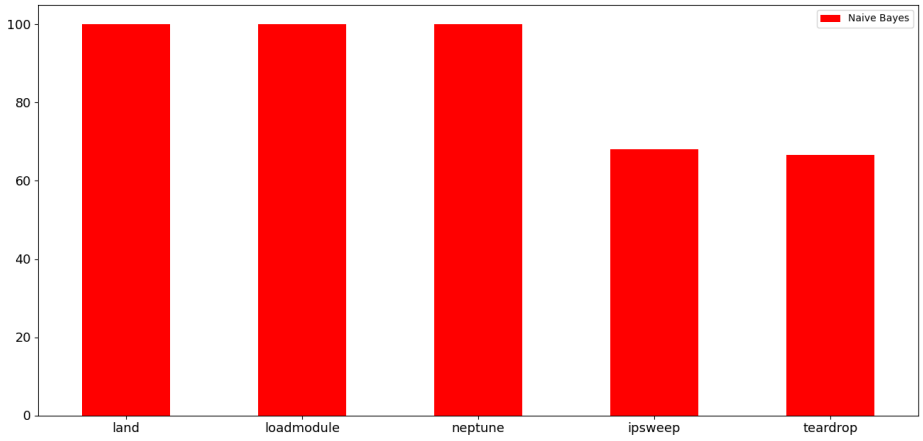

Figura 5. Subclasses com maior *Recall* alcançados pelo algoritmo Naive Bayes

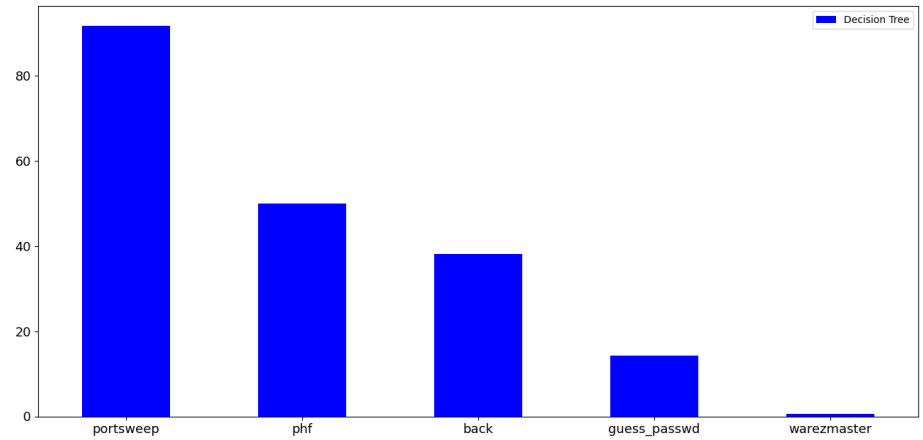

Figura 6. Subclasses com menor *Recall* alcançados pelo algoritmo *Decision Tree*

A Figura 6 apresenta as subclasses com menores taxas de *Recall* para o algoritmo *Decision Tree*. Dessas, somente a taxa de detecção da subclasse *warezmaster* ficou abaixo dos 10% e a *phf*, *back* e *guess_passwd* abaixo dos 60%. Já a *postsweep*, apesar de estar entre as cinco subclasses com menores taxas de detecção, teve cerca de 90% das suas amostras detectadas pelo algoritmo *Decision Tree*.

As cinco subclasses com menor taxa de *Recall* detectados pelo algoritmo *Naive Bayes* são apresentados na Figura 7. Diferente do *Decision Tree*, o algoritmo *Naive Bayes* não conseguiu classificar corretamente nenhuma amostra das subclasses *warezmaster*, *portsweep*, *pod* e *nmap*. Já a *multihop*, obteve apenas cerca de 5,5% de *Recall*.

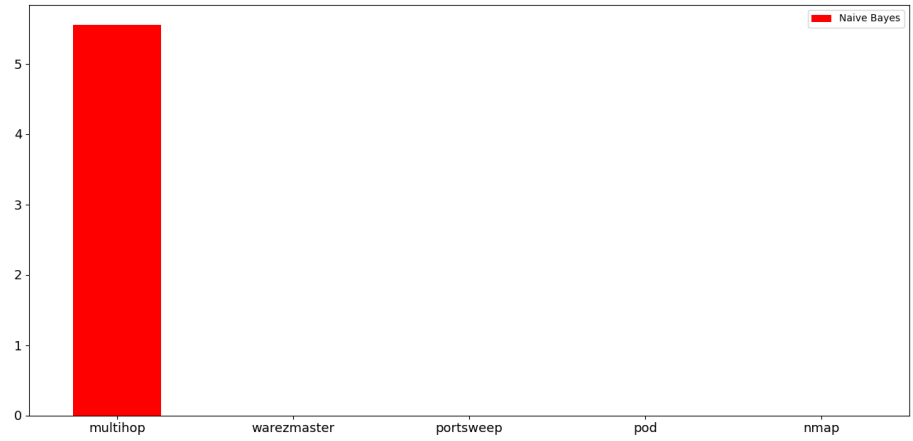

Figura 7. Subclasses com menor *Recall* alcançados pelo algoritmo *Naive Bayes*

Entre as subclasses presentes nas Figuras 6 e 7, somente a *warezmaster* aparece em ambas. A possível explicação para essa ocorrência, é o fato de que há um baixo volume de amostras dessa subclasse na base de treino, enquanto na base de teste existe um número elevado da mesma, sendo assim, é possível assumir que os modelos de *Machine Learning* não foram capazes de serem treinados o suficiente para saber identificar ocorrências da subclasse *warezmaster* com uma maior efetividade.

Algumas subclasses aparecem somente na base de testes, dificultando a classificação correta pelos algoritmos já que não tiveram acesso à amostras dessas subclasses na fase de treinamento. Dessa forma, não foram consideradas para compor os gráficos acima.

6.2.2. Custo computacional

A memória RAM consumida pelos algoritmos para o processamento das amostras não indica um possível problema para implementação em dispositivos com recursos limitados. Conforme o gráfico da Figura 8, o algoritmo *Naive Bayes* consumiu um pouco mais de 130 MB, enquanto que o *Decision Tree* precisou de menos de 80 MB. Esse último, além de retornar melhores resultados em precisão e *Recall*, teve uma menor necessidade de memória RAM disponível.

De acordo com o gráfico da Figura 9, o modelo *Decision Tree* levou cerca de 30 segundos a mais para o processamento das amostras em comparação com o modelo *Naive Bayes*. No entanto, considerando o número de amostras que compõem a base de dados de treinamento, o algoritmo *Decision Tree* processou a uma taxa de cerca de 1.574 amostras por segundo. Já o algoritmo *Naive Bayes* conseguiu processar cerca de 2.519 amostras por segundo. Apesar das altas taxas de processamento, o tempo de processamento pode ser um ponto de atenção para futuros testes, já que ataques DoS e DDoS tentam explorar a limitação de recursos dos equipamentos, entre eles o tempo de processamento.

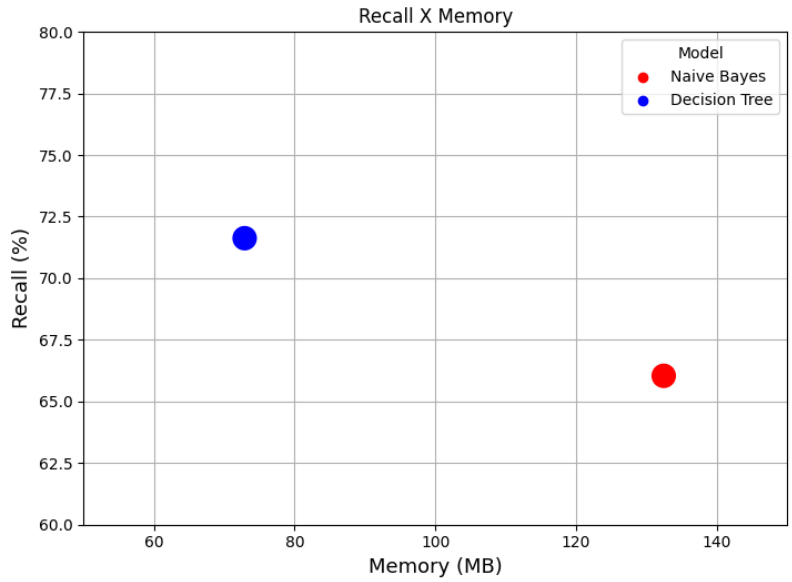

Figura 8. Consumo de memória RAM para o processamento das bases de treino e teste pelos algoritmos *Naive Bayes* e *Decision Tree*

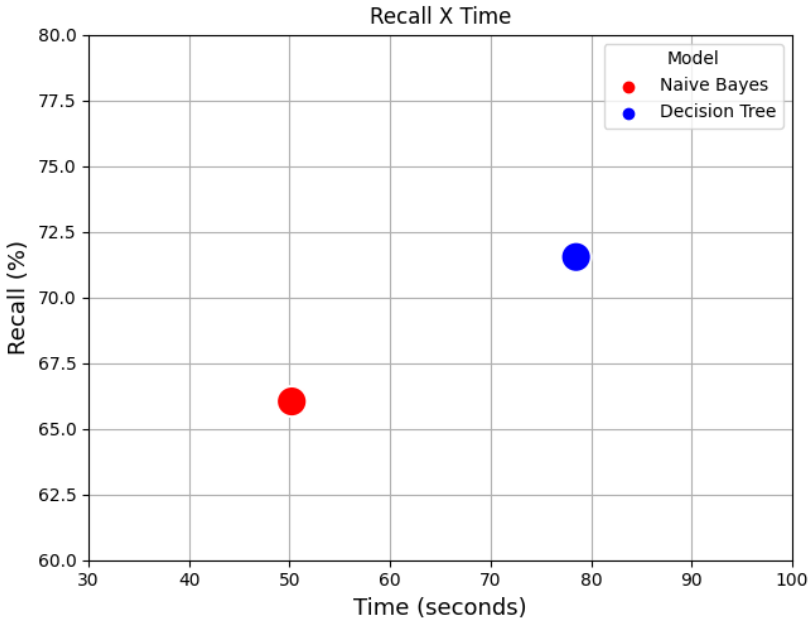

Figura 9. Tempo de execução total para processamento das bases de treino e teste pelos algoritmos *Naive Bayes* e *Decision Tree*

7. Conclusão

Com o crescimento das redes de computadores e também a evolução da tecnologia no geral, há também o aumento de ataques cibernéticos que se aproveitam de brechas nos sistemas e redes para atingir seu objetivo, seja este, indisponibilizar uma aplicação, como

até mesmo roubo de dados. Diante deste problema, é necessário ser capaz de identificar quando um ataque está sendo efetivado, por exemplo no caso dos ataques DDoSs.

Sendo assim, o objetivo deste trabalho foi, através de dois modelos de *Machine Learning* (*Naive Bayes* e *Decision Tree*) identificar qual que possui o menor custo computacional para ser executado, juntamente ao que tem a melhor assertividade em relação às 3 métricas principais de modelos de aprendizado de máquina utilizadas neste trabalho, sendo estas, Precisão, *Recall* e Acurácia. Através do *dataset* NSL-KDD, é possível obter uma quantidade razoável de dados para o treinamento e teste desses algoritmos. Junto ao uso de bibliotecas *Python* voltadas para o ramo da ciência de dados foram realizadas diversas simulações para atingir o resultado apresentado neste trabalho.

Portanto, através das análises realizadas nas métricas geradas pelos dois modelos e também acerca do custo computacional, como, consumo de memória RAM e tempo de execução, foi possível concluir que o modelo *Decision Tree* obteve os melhores resultados na maioria das métricas analisadas. Obtendo apenas resultado de *Recall* inferior ao *Naive Bayes* na classificação de poucas subclasses. Sua performance com relação a consumo de memória também se mostrou relativamente superior à do outro modelo, porém, se tratando de tempo de execução, o *Naive Bayes* obteve resultados mais satisfatório, o qual dependendo do cenário em que se estuda a aplicação desses modelos, pode ser um fator determinante.

8. Trabalhos Futuros

Apesar dos resultados obtidos neste trabalho, alguns estudos mais aprofundados e/ou alternativos poderiam trazer boas considerações e novos conhecimentos, como: realizar testes com uma base maior de dados, sendo assim, capaz de realmente mostrar a efetividade dos modelos em um cenário alternativo.

Outra sugestão seria realizar um estudo detalhado das características de cada classe, para identificar a complexidade da classificação de cada uma delas.

Também seria de grande valor realizar a aplicação dos modelos em uma base mais recente de dados, isto é, em um novo trabalho, realizar uma simulação de ataques DDoS em uma rede controlada, coletar essas amostras de pacotes e aplicar os modelos de *Machine Learning* já estudados nessa nova base.

E por fim, a aplicação dos modelos treinados para identificação de ataques DDoS em dispositivos IoT, assim validando qual modelo se aplica melhor em computadores que possuem, de certa forma, capacidade de processamento e armazenamento baixo.

Referências

AllEasy (2019). Entenda a efetividade do machine learning contra ataques cibernéticos. https://www.alleasy.com.br/2019/11/11/entenda-a-efetividade-do-machine-learning-contra-ataques-ciberneticos/.

Archive, T. U. K. (1999). Kdd cup 1999 data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

Cerri, R. and Carvalho, A. C. P. d. L. F. d. (2017). Aprendizado de máquina: breve introdução e aplicações. *Cadernos de Ciencia e Tecnologia*.

CERT.br (2017). Cartilha de segurança para internet. https://cartilha.cert.br/seguranca/.

Choudhary, S. and Kesswani, N. (2020). Analysis of kdd-cup'99, nsl-kdd and unsw-nb15 datasets using deep learning in iot. *Procedia Computer Science*, 167:1561–1573.

Dahlqvist, F., Patel, M., Rajko, A., and Shulman, J. (2019). Growing opportunities in the internet of things. https://www.mckinsey.com/industries/private-equity-and-principal-investors/our-insights/growing-opportunities-in-the-internet-of-things.

Das, S., Mahfouz, A. M., Venugopal, D., and Shiva, S. (2019). Ddos intrusion detection through machine learning ensemble. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 471–477.

de arquitetura do Cloud, C. (2018). Pré-processamento de dados para machine learning: opções e recomendações. https://cloud.google.com/architecture/data-preprocessing-for-ml-with-tf-transform-pt1?hl=pt-br.

Devi, R. and Abualkibash, M. (2019). Intrusion detection system classification using different machine learning algorithms on kdd-99 and nsl-kdd datasets - a review paper. *International Journal of Computer Science and Information Technology*, 11:65–80.

for Cybersecurity, C. I. Nsl-kdd dataset. https://www.unb.ca/cic/datasets/nsl.html.

Hoon, K. S., Yeo, K. C., Azam, S., Shunmugam, B., and De Boer, F. (2018). Critical review of machine learning approaches to apply big data analytics in ddos forensics. In *2018 International Conference on Computer Communication and Informatics (ICCCI)*, pages 1–5.

Hope, C. (2017). Syn. https://www.computerhope.com/jargon/s/syn.htm.

Idhammad, M., Afdel, K., and Belouch, M. (2018). Semi-supervised machine learning approach for ddos detection. *Applied Intelligence*, 48:3193–3208.

Jang-Jaccard, J. and Nepal, S. (2014). A survey of emerging threats in cybersecurity. *Journal of Computer and System Sciences*, 80(5):973–993. Special Issue on Dependable and Secure Computing.

Jijo, B. and Mohsin Abdulazeez, A. (2021). Classification based on decision tree algorithm for machine learning. *Journal of Applied Science and Technology Trends*, 2:20–28.

Keboola (2020). A guide to principal component analysis (pca) for machine learning. https://www.keboola.com/blog/pca-machine-learning.

Mitshashi, R. A. (2011). Segurança de redes. http://www.fatecsp.br/dti/tcc/tcc0017.pdf.

Oliveira, S. I. C. (2018). Estudo sobre o impacto de dispositivos iot em ataques ddos.

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., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Revathi, S. and Malathi, A. (2013). A detailed analysis on nsl-kdd dataset using various machine learning techniques for intrusion detection. *International journal of engineering research and technology*, 2.

Silva, L. N. M. (2021). Tipos de aprendizado de máquina e algumas aplicações. http://www2.decom.ufop.br/terralab/tipos-de-aprendizado-de-maquina-e-algumas-aplicacoes/.

Staudemeyer, R. and Omlin, C. (2014). Extracting salient features for network intrusion detection using machine learning methods. *South African Computer Journal*.

Vaz, A. L. (2019). Normalizar ou padronizar as variáveis? https://medium.com/data-hackers/normalizar-ou-padronizar-as-vari%C3%A1veis-3b619876ccc9.

Vieira, N. (2019). Ataques ddos geram preocupação no setor financeiro. https://canaltech.com.br/seguranca/ataques-ddos-geram-preocupacao-no-setor-financeiro-156861/.

WANKE, B. d. S. L., COSTA, V. O., PINA, A. C. d., and PINA FILHO, A. C. d. (2014). Aplicação do classificador naive bayes para identificação de falhas de um manipulador robótico. *ABCM Symposium Series in Mechatronics, Part II - National Congress, Section I - Aplicações de Inteligência Artificial.*, 6.

Özgür, A. and Erdem, H. (2016). A review of kdd99 dataset usage in intrusion detection and machine learning between 2010 and 2015.

Documento Digitalizado Público

Anexo I (Artigo) - Maria Eduarda de Oliveira Silva - HT1720376

Assunto: Anexo I (Artigo) - Maria Eduarda de Oliveira Silva - HT1720376
Assinado por: Carlos Junior
Tipo do Documento: Outro
Situação: Finalizado
Nível de Acesso: Público
Tipo do Conferência: Documento Digital

Documento assinado eletronicamente por:

- **Carlos Roberto dos Santos Junior**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 20/07/2022 22:02:01.

Este documento foi armazenado no SUAP em 20/07/2022. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1043249
Código de Autenticação: 6659640e7e