UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RENATO LEITE CAMILO

ANÁLISE DO DESEMPENHO DE HONEYPOTS E ALGORITMOS DE MACHINE LEARNING EM TAREFAS DE DETECÇÃO DE INTRUSÃO

BAURU
Dezembro/2023

RENATO LEITE CAMILO

ANÁLISE DO DESEMPENHO DE HONEYPOTS E ALGORITMOS DE MACHINE LEARNING EM TAREFAS DE DETECÇÃO DE INTRUSÃO

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
Dezembro/2023

Renato Leite Camilo Análise do desempenho de honeypots e algoritmos de machine learning em tarefas de detecção de intrusão/ Renato Leite Camilo. – Bauru, Dezembro/2023- 48 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, Dezembro/2023.
1. Tags 2. Para 3. A 4. Ficha 5. Catalográfica

Renato Leite Camilo

Análise do desempenho de honeypots e algoritmos de machine learning em tarefas de detecção de intrusão

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

Profa. Dra. Simone Domingues Prado
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Prof. Dr. Douglas Rodrigues
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Bauru, 16 de novembro de 2023.

Agradecimentos

Assim como na vida, o fim é inevitável. Ciclos eventualmente chegam ao seu limite e isso não deve ser motivo de dor ou de preocupação. Durante quatro anos convivi com pessoas que me transformaram e algumas delas definitivamente levarei para a minha vida. Hoje, me arrependo de algumas ações e comportamentos que tive durante esse período, mas não deixo isso ofuscar tudo de bom o que me proporcionaram e de todo os ensinamentos. No final das contas, são as experiências, boas ou ruins, que nos tornam melhores como pessoas.

Em especial, quero agradecer à minha família. Sem ela, nada disso seria possível e é por ela que eu encerro esse ciclo. Agradeço também todos os meus amigos que tornaram esses momentos mais alegres, e meu orientador que proporcionou essa oportunidade acadêmica. E por fim, falo diretamente com vocês: muito obrigado por tudo. Não existem palavras para descrever o quão importante vocês foram ao longo desses quatro anos. Alguns de vocês chegaram mais tarde e outros estão desde o início, mas o carinho que tenho por vocês é único. Por fim, encerro meus agradecimentos com uma pergunta especificamente para você: será que tudo seria o mesmo se você não fosse super organizado?

Não chore porque já terminou, sorria porque aconteceu.

Ludwig Jacobowski

Resumo

A cibersegurança é uma área de grande interesse na computação. A capacidade de detectar e lidar com invasores em ambientes digitais é uma preocupação para muitas pessoas e empresas. Em contrapartida, os mecanismos que proporcionam essa capacidade nem sempre são totalmente confiáveis, falhando em algumas ocasiões e causando prejuízos inimagináveis para seus usuários. Como resolução para esse problema, muito se propõe o uso de *machine learning*. Esse projeto emprega dois algoritmos de *clustering*: K-Means e DBSCAN. O objetivo é analisar seu impacto quando usados em conjuntos com outros classificadores baseados em três algoritmos de aprendizado de máquina: Support Vector Machines, Multilayer Perceptron e K-Nearest Neighbors. Além disso, é empregado o conceito de *honeypots* para analisar sua usabilidade como receptáculo de informações a respeito de ataques para providenciar dados aos modelos preditivos. Tudo que chega a um *honeypot* é considerando um ataque e esse tipo de informação pode ser aproveitado por modelos de aprendizado de máquina.

Palavras-chave: cibersegurança, ciberataques, aprendizado de máquina, clusterização, honeypot.

Abstract

Cybersecurity is an area of great interest in computing. The ability to detect and deal with attackers in digital environments is a concern for many people and companies. On the other hand, the mechanisms that provide this capacity are not always completely reliable, failing on some occasions and causing unimaginable losses for their users. As a solution to this problem, many people propose the use of *machine learning*. This project employs two *clustering* algorithms: K-Means and DBSCAN. The objective is to analyze their impact when used in conjunction with other classifiers based on three machine learning algorithms: Support Vector Machines, Multilayer Perceptron and K-Nearest Neighbors. Furthermore, the concept of *honeypots* is used to analyze its usability as a receptacle of information regarding attacks to provide data for predictive models. Everything that reaches a *honeypot* is considered an attack and this type of information can be used by machine learning models.

Keywords: cybersecurity, cyberattacks, machine learning, clustering, honeypot.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

DOS	Denial of Service
R2L	Remote to Local
U2R	User to Root
IDS	Intrusion Detection System
HIDS	Host Intrusion Detection System
NIDS	Network Intrusion Detection System
SVM	Support Vector Machine
MLP	Multilayer Perceptron
RELU	Rectified Linear Unit
KNN	K-Nearest Neighbors
VP	Verdadeiro Positivo
FP	Falso Positivo
VN	Verdadeiro Negativo
FN	Falso Negativo

Sumário

1 Introdução

A segurança é um direito de qualquer indivíduo. Embora existam leis, regras de convívio e meios de se proteger contra criminosos, nem sempre temos a total garantia de segurança. Em ambientes digitais, isso não é diferente. Assim como no mundo real, existem indivíduos que a todo custo querem causar danos aos alheios, prejudicando a segurança e o bem-estar, e também mecanismos e medidas de segurança que nos permitem se defender contra os malfeitores.

No mundo virtual, "um ataque cibernético é qualquer esforço intencional para roubar, expor, alterar, desabilitar ou destruir dados, aplicativos ou outros ativos por meio de acesso não autorizado a uma rede, sistema de computador ou dispositivo digital"(IBM, 2023). Pesquisas recentes demonstram um aumento significativo no número de ataques cibernéticos. Como aponta um levantamento produzido pela Fortinet, o Brasil registrou 31,5 bilhões de tentativas de ataques cibernéticos a empresas no primeiro semestre de 2022, representando um aumento de aproximadamente 94% em relação ao ano anterior (CNN, 2022).

Para nos protegermos de ataques digitais, adotamos medidas de cibersegurança. A "cibersegurança é uma prática que protege computadores e servidores, dispositivos móveis, sistemas eletrônicos, redes e dados contra ataques maliciosos"(KASPERSKY, 2023). Em linhas gerais, essa prática envolve qualquer esforço para identificar e eliminar invasores. Mecanismos de defesa, como sistemas de detecção e prevenção de intrusões, bem como firewalls, operam com esse propósito.

Entretanto, ainda existem vulnerabilidades em mecanismos de defesa que são exploradas por invasores, comprometendo a segurança nos ambientes digitais.

1.1 Problema

Levando como exemplo os sistemas de detecção de intrusão, um dos seus maiores problemas está em sua detecção com um número relativamente alto de falsos positivos, ou seja, há um alto número de comportamentos normais que são detectados como ataque (ASSUNCAO., 2009). Esses mecanismos são competentes o suficiente para identificar a maioria dos ataques, mas constantemente identificam comportamentos normais como comportamentos maliciosos, o que gera muita informação inútil em arquivos de registro destinados para análise de novas tendências em ciberataques.

Seguindo para uma abordagem de detecção de ataques por assinatura presentes em alguns sistemas de detecção de intrusão, notamos um indesejável problema: eles necessitam ser alimentados constantemente com informações de novos ataques. Em linhas gerais, esses sistemas são projetados para identificar assinaturas de ataques conhecidas, ou seja, padrões

conhecidos. Esses mecanismos de defesa identificarão um comportamento estranho apenas se o mesmo coincidir com uma das assinaturas cadastradas no seu banco de dados. Por conta disso, ligeiras alterações na assinatura de um ataque são suficientes para superar essa barreira.

Não se sabe exatamente quando uma nova técnica de ataque irá surgir, mas é muito importante que os mecanismos de defesa estejam preparados para esse momento. Em outras palavras, ele devem ser mais robustos, flexíveis e inteligentes.

1.2 Justificativa

Para resolver esse problema, empregamos uma abordagem de detecção com *machine learning*. Há muita pesquisa em torno da empregabilidade de *machine learning* em tarefas de detecção de intrusão. Algumas delas, inclusive, citam o uso de técnicas de *clustering* como possibilidade para gerar taxas de detecção maiores do que classificadores únicos (MISHRA et al., 2018). Com o uso de aprendizado de máquina, os sistemas tornam-se dinâmicos e mais eficientes.

Para entender as novas tendências de ciberataques, usamos honeypots. Eles são sistemas projetados exclusivamente para serem atacados, oferecendo vulnerabidades aos invasores. Devido à suas propriedades e segundo a teoria da obscuridade, qualquer comunicação com um honeypot é, por definição, um ataque (NAWROCKI et al., 2016). Em outras palavras, um honeypot não possui registros de falsos positivos, isto é, comportamentos normais que foram tratados como maliciosos. No entanto, mesmo com suas enormes vantagens, são sistemas muito pouco aproveitados em relação à pesquisas de detecção de intrusão (ASSUNCAO., 2009). Em suma, honeypots são excelentes acumuladores de informação que podem ser usados com cautela na descoberta de novos ataques cibernéticos.

1.3 Objetivos

O objetivo geral desse projeto é analisar a eficácia de técnicas de *clustering* para tarefas de detecção de intrusão e a viabilidade do uso de dados coletados por honeypots para criação de sistemas de detecção mais robustos e flexíveis.

1.3.1 Objetivos específicos

O presente trabalho possui os seguintes objetivos específicos:

- Revisar a literatura para identificar o estado da arte sobre detecção de intrusão usando *machine learning*
- Estudar e compreender o que são *honeypots* e como eles podem ser usados em tarefas de detecção de intrusão

- Estudar e entender como aplicar os possíveis métodos de *clustering* em conjunto com algoritmos de aprendizado de máquina
- Reunir dados coletados por *honeypots* para o treinamento de modelos de *machine learning* em conjunto com técnicas de *clustering*
- Analisar a eficácia dos métodos de *clustering* e comparar os resultados com trabalhos correlatos

1.4 Trabalhos relacionados

Em (SONG et al., 2011), os autores propõem um novo dataset para pesquisadores na área de sistemas de detecção de intrusão com machine learning. A explicação é que os datasets normalmente usados para treinamento de modelos de aprendizado de máquina não refletem cenários atuais de ataque e a maioria deles são frutos de dados coletados por simulações em redes virtuais. O dataset em questão foi construído usando uma rede de honeypots instalada em uma universidade no Japão.

Em (MISHRA et al., 2018), os autores reúnem e analisam trabalhos na área de detecção de intrusão relacionados a técnicas de aprendizado de máquina. Para cada trabalho foram analisados os algoritmos e os conjuntos de dados empregados, o conjunto de features escolhidas para o treinamento dos modelos, seus resultados e as métricas de avaliação usadas, servindo como um ótimo norte para novas pesquisas na área.

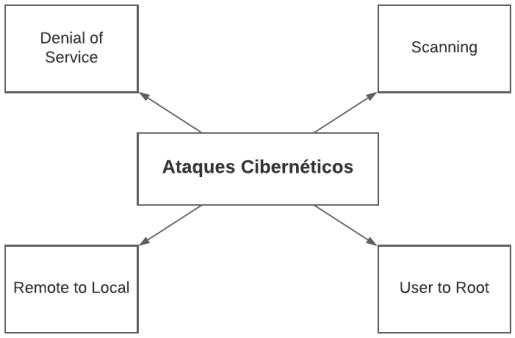
Em (GRÉGIO; SANTOS; MONTES, 2007), os autores aplicam três técnicas de aprendizado de máquina em pequenas amostras de dados reais de tráfego de internet e honeypots. As técnicas usadas foram: K- Nearest Neighbors, Artificial Neural Networks e Decision Trees. Os dados eram classificados como normais ou anormais, possibilitando uma análise mais curta e eficiente dos arquivos de registro gerados durante o monitoramento da rede.

2 Revisão teórica

2.1 Ataques cibernéticos

Ataques cibernéticos são tentativas maliciosas de explorar vulnerabilidades em sistemas de computadores, redes e ativos de tecnologia da informação. Segundo (MISHRA et al., 2018), podemos classificar ataques em quatro categorias: *Denial of Service (DOS)*, *Scanning*, *Remote to Local (R2L)* e *User to Root (U2R)*. Podemos observar essa classificação conforme figura 1.

Figura 1 – Uma classificação para ataques cibernéticos



Fonte: Elaborada pelo autor.

Quando lidamos com detecção de ameaças, os ataques do tipo *R2L* e *U2R* são mais discretos do que os ataques do tipo *DOS* e *Scanning*. Por conta disso, são categorias que exigem uma análise detalhada para sua efetiva identificação. Em contrapartida, ataques do tipo *DOS* e *Scanning* normalmente são reconhecidos apenas com análise de tráfego de rede (MISHRA et al., 2018).

De acordo com (MISHRA et al., 2018), cada uma das categorias de ataques cibernéticos mencionadas podem ser descritas como:

- **DOS**: técnica usada para sobrecarregar um sistema, serviço ou rede, tornando-o inacessível para usuários legítimos.
- **Scanning**: técnica usada para identificar vulnerabilidades em sistemas, redes ou aplicativos. As informações coletadas durante a varredura são frequentemente usadas para planejar e executar ataques mais sofisticados, como invasões ou exploração de vulnerabilidades.
- **R2L**: técnica usada para obter acesso à uma máquina local por meio de conexão remota por meio da exploração de vulnerabilidades.

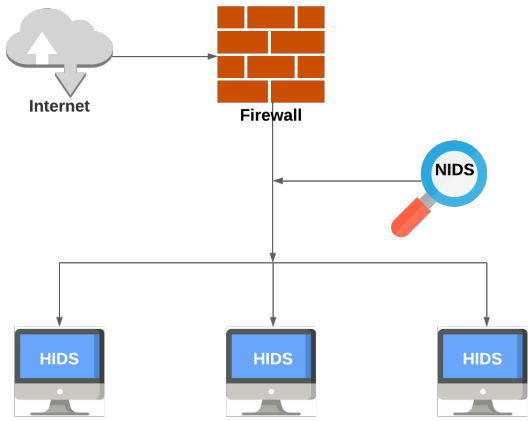
- **U2R**: técnica usada para obter privilégios de superusuário em um sistema de computador ou rede por meio da exploração de vulnerabilidades, normalmente após ter conseguido acesso como usuário comum.

2.2 Sistema de detecção de instrução

Sistemas de detecção de intrusão (*Intrusion Detection System*, IDS) são ferramentas programadas para monitorar e identificar atividades suspeitas ou maliciosas em uma rede ou sistema de computador. Um IDS pode ser classificado como NIDS (*Network Intrustion Detection System*) ou HIDS (*Host Intrusion Detection System*) de acordo com seu posicionamento na rede e seu modo de operação (ALMANSOR; GAN, 2018). Respectivamente, são sistemas cuja detecção é baseada em rede ou baseada em *host*. Ambas as abordagens podem ser observadas conforme figura 2 e descritas em linhas gerais como:

- **Baseado em rede**: monitoram o tráfego de rede inspecionando pacotes em busca de padrões anômalos ou assinaturas de ataques conhecidos e geram alertas.
- **Baseado em host**: são instalados em *hosts* individuais e monitoram atividades no nível do sistema operacional e do aplicativo, identificando alterações de arquivos e uso irregular, por exemplo.

Figura 2 – Sistemas de detecção de intrusão em uma rede



Fonte: Elaborada pelo autor.

Existem duas principais abordagens para detectar ameaças: reconhecimento por assinatura ou por anomalia (ALMANSOR; GAN, 2018).

- **Assinatura**: A abordagem por assinatura visa identificar ataques conhecidos com base em padrões de comportamento malicioso. Em outras palavras, o sistema procura por

assinaturas de ataques em pacotes de dados, e uma ameaça é reconhecida somente se uma assinatura correspondente é encontrada.

- **Anomalia**: A abordagem por anomalia detecta ameaças por meio de variações inesperadas em relação ao tráfego de rede padrão. Um perfil típico de tráfego de rede é definido, e qualquer desvio em relação a esse padrão é tratado como uma possível ameaça.

De acordo com (ASSUNCAO., 2009), ambas as abordagens possuem defeitos. De um lado, o reconhecimento de ameças por assinatura não detecta variações de um mesmo ataque. Em outras palavras, pequenas alterações na assinatura de um ataque podem enganar um sistema detector de intrusão. Por outro lado, o reconhecimento de ameaças por anomalia é traiçoeiro, pois o perfil de tráfego de uma rede pode mudar frequentemente ao longo do tempo, afetando o desempenho desse tipo de sistema. Para resolver problemas como esses, existem muitas pesquisas na implementação de algoritmos de aprendizado de máquina (MISHRA et al., 2018).

2.3 Honeypots

Honeypots são sistemas ou dispositivos projetados para atrair e enganar invasores, simulando vulnerabilidades e oferecendo uma armadilha com o propósito de monitorar e estudar suas táticas. Esses sistemas devem ser configurados com o propóstio de atrair a atenção dos invasores, equipando-os com serviços de transferência de arquivos ou de correio, por exemplo. De acordo com (ASSUNCAO., 2009), um *honeypot* que não instigue interesse nos invasores é ineficaz para o estudo de ataques e invasões. Portanto, é essencial oferecer serviços, sejam eles reais ou simulados. Em relação à fidelidade desses serviços, os *honeypots* podem ser divididos em duas categorias: *honeypots* de alta interação e *honeypots* de baixa interação. Ambas as abordagens podem ser descritas como segue:

- ***Honeypots* de alta interação**: geralmente usados em contextos de pesquisa, onde o objetivo é analisar e compreender o comportamento dos invasores. Nesse tipo de *honeypot*, serviços reais são disponibilizados para os invasores interagirem, possibilitando uma coleta abrangente de informações sobre o ataque, o que resulta em análises mais detalhadas e precisas. No entanto, é importante ressaltar que, nesses casos, a segurança do sistema pode estar comprometida, uma vez que serviços reais estão acessíveis aos invasores (ASSUNCAO., 2009).
- ***Honeypots* de baixa interação**: geralmente usados em ambientes de produção, com foco na detecção e expulsão imediata de invasores da rede. Os serviços disponibilizados aos invasores são simulados, evitando qualquer risco real de comprometimento do sistema.

2.4 Aprendizado de máquina

A área do aprendizado de máquina, também conhecida como *machine learning*, constitui uma das subáreas essenciais da inteligência artificial. Em resumo, os algoritmos de *machine learning* se baseiam em dados para fornecer soluções a uma variedade de problemas.

Considere, por exemplo, uma tarefa de classificação de imagens, em que você recebe imagens de animais e precisa identificá-los. Nesse cenário, você possui um conjunto de referência que associa cada imagem a um animal específico. Você divide esse conjunto em duas partes: uma delas é usada para treinar o algoritmo. Com esses dados em mãos, o modelo aprende a classificar as imagens de acordo com os rótulos. A outra parte do conjunto de referência é usada para testar o modelo, e você verifica a precisão das classificações. Esse processo se repete até que o algoritmo possa, com base nas imagens, classificar de forma satisfatória. Esse exemplo ilustra o aprendizado de máquina supervisionado, no qual o sistema conhece a resposta correta para cada entrada.

Algoritmos de aprendizado de máquina se dividem principalmente em três categorias: supervisionado, como mencionado anteriormente, não supervisionado e por reforço. Em resumo, os algoritmos em cada categoria têm os seguintes objetivos: mapear entradas e saídas, descobrir padrões nos dados agrupando-os e aprender por meio de interações e *feedback*. Durante o desenvolvimento deste projeto, nossa ênfase estará principalmente nos algoritmos de aprendizado de máquina supervisionado, como *Support Vector Machine*, *Multilayer Perceptron* e *K-Nearest Neighbours*, e nos algoritmos de aprendizado de máquina não supervisionado, como *K-Means* e *DBSCAN*.

2.4.1 Support Vector Machine

Um *Support Vector Machine* é, de maneira resumida, um algoritmo de aprendizado de máquina que tem por objetivo descobrir o melhor hiperplano que separa um determinado conjunto de dados. "Em essência, um SVM é uma entidade matemática, um algoritmo para maximizar uma função matemática específica em relação a um dada coleção de dados"(NOBLE, 2006). É um modelo muito usado para tarefas de classificação binária. Para entender a essência de um *Support Vector Machine*, é necessário compreender quatro conceitos básicos: o hiperplano separador, o hiperplano de margem máxima, margem suave e a função *kernel*.

- **Hiperplano separador**: na geometria, um hiperplano é um espaço onde a dimensão é um grau menor que a dimensão do problema. Se a dimensão do problema se encontra em um espaço de três dimensões, um espaço de duas dimensões é considerado como hiperplano do mesmo. Em um algoritmo de SVM, usamos um hiperplano separador como método para segmentar um conjunto de amostras em dois subconjuntos.

- **Hiperplano de margem máxima**: hiperplano separador que maximiza a distância do mesmo aos pontos vizinhos mais próximos.
- **Margem suave**: permite que uma quantia de amostras sejam classificadas incorretamente
- **Função *kernel***: operação matemática que projeta dados de um espaço de baixa dimensão para um espaço de dimensão superior

2.4.2 Multilayer Perceptron

Um *Multilayer Perceptron* é uma rede neural: é um algoritmo implementado de forma a imitar o comportamento do cérebro humano. Inclusive, o próprio modelo possui um nome intuitivo: *perceptron* de múltiplas camadas. Cada camada é formada por *perceptrons*, unidades matemáticas que simulam o comportamento dos neurônios. Essas redes neurais não possuem tamanho fixo, podendo variar desde as mais simples até as mais complexas, para os mais diversos fins (GARDNER; DORLING, 1998).

Um MLP é composto por uma arquitetura em camadas, que inclui uma camada de entrada, uma ou mais camadas ocultas (hidden layers) e uma camada de saída. A camada de entrada recebe os dados brutos e consiste em neurônios correspondentes às características de entrada. As camadas ocultas são responsáveis por aprender representações intermediárias dos dados, enquanto a camada de saída gera a saída final do modelo.

Cada neurônio em um MLP aplica uma função de ativação à sua entrada ponderada. As funções de ativação introduzem não linearidades nas camadas ocultas, permitindo que o MLP aprenda representações complexas e não lineares dos dados. Algumas funções de ativação comuns incluem a função sigmoide, a função tangente hiperbólica (tanh) e funções de ativação mais modernas, como a ReLU (Rectified Linear Unit).

O treinamento de MLPs envolve a otimização dos pesos e tendências das conexões neurais para minimizar uma função de perda, que calcula a diferença entre as previsões do modelo e os valores reais. O algoritmo de retropropagação (backpropagation) é amplamente utilizado para esse fim. Durante o treinamento, os pesos são atualizados iterativamente usando gradientes calculados a partir da função de perda.

2.4.3 K-Nearest Neighbors

O algoritmo K-Nearest Neighbors (KNN) é um método de aprendizado de máquina supervisionado e baseado em instância. A abordagem do KNN é intuitiva e simples, baseada no princípio de que pontos de dados semelhantes tendem a pertencer à mesma classe ou ter valores de saída semelhantes (ZHANG et al., 2017). O KNN opera calculando a distância entre um novo exemplo e os pontos de dados de treinamento no espaço de características, com o parâmetro K especificando quantos vizinhos mais próximos devem ser considerados na

decisão. A escolha da métrica de distância e do valor de K é essencial e afeta a sensibilidade do algoritmo. O KNN é aplicado tanto em tarefas de classificação quanto em regressão, onde a classe majoritária ou a média dos valores alvo dos vizinhos mais próximos determinam a previsão (ZHANG et al., 2017). No entanto, o KNN possui limitações em relação à dimensionalidade e ao custo computacional em conjuntos de dados grandes. Em resumo, o KNN é uma técnica versátil, porém sensível aos parâmetros, que se baseia na proximidade entre pontos de dados no espaço de características para realizar classificação e regressão.

2.5 Clustering

Clustering é um conjunto de técnicas de *machine learning* não supervisionado que têm por objetivo gerar grupos de amostras que possuem relações entre si. Em linhas gerais, a técnica de *clustering* tem por objetivo agrupar amostras por semelhança. Os algoritmos de *clustering* poder ser separados em quatro abordagens: baseados em centroides, baseados em densidade, baseado em distribuição e hierárquica (KODINARIYA; MAKWANA et al., 2013).

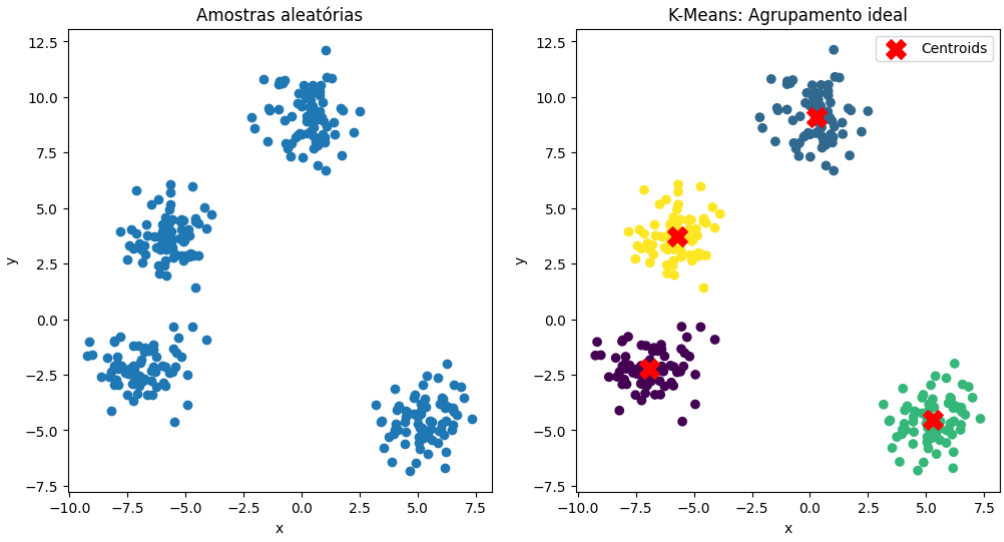
- **Baseados em Centroides**: procuram agrupar as amostras em torno de pontos centrais chamados centroides. Os *clusters* são formados com base na proximidade das amostras em relação aos centroides.
- **Baseados em Densidade**: identificam *clusters* com base na densidade local de pontos. Eles procuram áreas de alta densidade de pontos e consideram regiões menos densas como fronteiras entre *clusters*.
- **Baseados em Distribuição**: modelam as distribuições probabilísticas dos dados e atribuem amostras aos *clusters* com base nas probabilidades.
- **Hierárquica**: constroem uma estrutura de árvore que representa a relação de agrupamento entre as amostras. Podem ser aglomerativos, onde cada amostra começa como seu próprio *cluster* e é mesclada progressivamente, ou divisivos, onde todas as amostras começam em um único *cluster* e são divididas em subgrupos.

2.5.1 K-Means

O *K-Means* é um algoritmo de *clustering* baseado em centroides que tenta "separar as amostras em grupos de variância igual, minimizando um critério conhecido como inércia ou soma dos quadrados dentro do *cluster*"(PEDREGOSA et al., 2011). Em resumo, o algoritmo *K-Means* busca definir a melhor posição dos centroides em um conjunto de dados. Os centros são iniciados aleatoriamente e um determinado dado pertencerá a um determinado *cluster* de acordo com a menor distância entre centro e ponto. Após todos os pontos serem classificados, o centro do grupo é reajustado calculando a média da posição de todos os pontos pertencentes

ao mesmo *cluster*. Essas etapas são repetidas até que o centro convirja para um valor ideal. Um exemplo dessa técnica pode ser observado conforme Figura 3, onde três centroides são inicializados aleatoriamente e encontram seu local ideal formando três agrupamentos de pontos.

Figura 3 – K-Means: um exemplo de *clustering*



Fonte: Elaborada pelo autor.

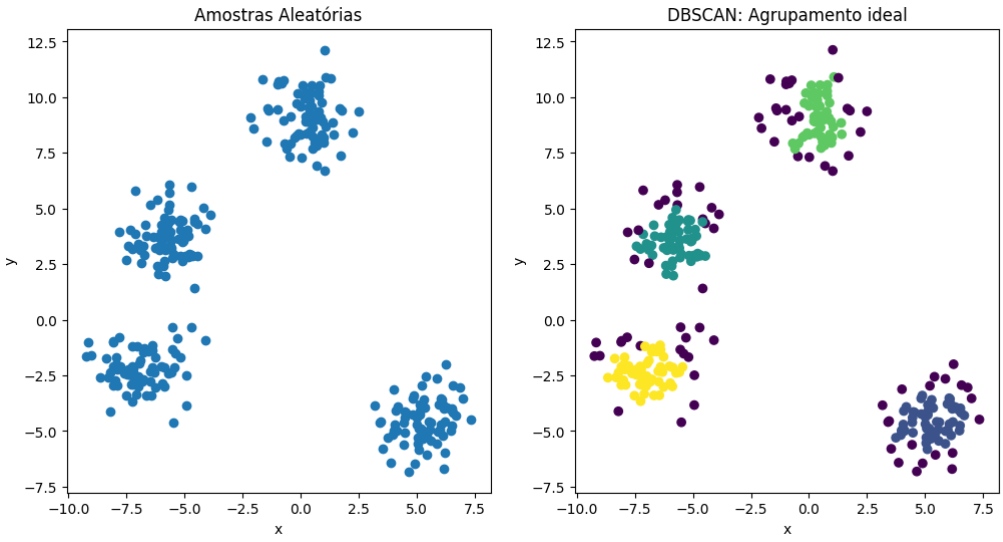
Nessa técnica, o número de *clusters* formados deve ser previamente determinado pelo usuário. Por conta disso, existem métodos que ajudam a escolher o número ideal de *clusters* para um conjunto de dados, como: *By rule of thumb*, *Elbow method*, *Information Criterion Approach*, *An Information Theoretic Approach*, *Using the Silhouette* e *Cross-validation* (KODINARIYA; MAKWANA et al., 2013).

Esse algoritmo lida bem com grandes conjuntos de dados e é um dos mais aplicados em pesquisas na área de *machine learning* (PEDREGOSA et al., 2011).

2.5.2 DBSCAN

"*DBSCAN* é o primeiro algoritmo de clustering baseado em densidade. Ele foi proposto por Ester et al. em 1996, e foi projetado para agrupar dados de formas arbitrárias na presença de ruído em bancos de dados espaciais e não espaciais de alta dimensão"(KHAN et al., 2014). Em suma, a quantidade de pontos em uma determinada região ocasiona a formação de *clusters*. Um exemplo dessa técnica pode ser observado conforme Figura 4. Nesse exemplo, amostras foram geradas aleatoriamente. Cada cor representa um agrupamento. As amostras em roxo, porém, são pontos que não pertencem a nenhum grupo.

Figura 4 – DBSCAN: um exemplo de *clustering*



Fonte: Elaborada pelo autor.

O algoritmo *DBSCAN* é ideal para conjuntos de dados com um número muito alto de amostras, lidando bem com dados de múltiplas dimensões, agrupamentos de diferentes tamanhos e dados discrepantes (PEDREGOSA et al., 2011). Seus *clusters* são calculados considerando principalmente dois parâmetros: *Eps* e *MinPts*. O primeiro parâmetro indica a distância limite em que dois pontos devem estar para serem considerados do mesmo *cluster*, enquanto o segundo parâmetro indica o valor mínimo de pontos para a formação de um agrupamento (KHAN et al., 2014)

2.6 Métricas de avaliação

Métricas de avaliação são expressões matemáticas responsáveis por calcular a eficácia de um modelo preditivo. Existem diferentes métricas de avaliação para diferentes tipos de problema em aprendizado de máquina, sejam eles de classificação, regressão ou agrupamento. A seguir, especificaremos as seguintes métricas de avaliação para tarefas de classificação: acurácia, recall, precisão e F1-score. Em problemas de classificação binária, essas métricas são calculadas de acordo com o número de classes corretamente classificadas (verdadeiros positivos e verdadeiros negativos) e de classes incorretamente classificadas (falsos positivos e falsos negativos). Uma matriz para visualizar esse tipo de tarefa pode ser observada abaixo.

Tabela 1 – Matriz de Confusão para tarefas de classificação binária

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

A acurácia é a métrica de avaliação de modelos preditivos para classificação binária ou de múltiplas classes mais usada na prática, possuindo uma menor complexidade de cálculo,

porém entregando valores menos distintivos e discrimináveis (HOSSIN; SULAIMAN, 2015). De maneira geral, a acurácia calcula o número de classes corretamente classificadas sobre o total de amostras analisadas, como exemplificado pela função matemática abaixo:

$$\frac{VP + VN}{VP + FP + VN + FN} \tag{2.1}$$

Analisando a equação acima, quanto mais próxima essa métrica estiver de 1, menor o número de classificações incorretas feitas pelo modelo.

A sensibilidade, conhecida como *recall*, é usada para medir a relação das classes verdadeiras. Resumidamente, essa métrica divide o total de classes corretamente classificadas como positivas pela soma de todos os valores de classes que devem ser classificadas como positivas (verdadeiras positivas e falsas negativas) (HOSSIN; SULAIMAN, 2015). A equação pode ser entendida da seguinte forma:

$$\frac{VP}{VP + FN} \tag{2.2}$$

Analisando a equação acima, quanto mais próxima essa métrica estiver de 1, menor o número de falsos negativos previstos pelo modelo.

A precisão é usada para medir a relação entre as classes positivas. Em outras palavras, essa métrica divide o total de classes corretamente classificadas como positivas pela soma de todos os valores previstos como positivos pelo algoritmo (verdadeiro positivo e falso positivo) (HOSSIN; SULAIMAN, 2015). A equação pode ser entendida da seguinte forma:

$$\frac{VP}{VP + FP} \tag{2.3}$$

Analisando a equação acima, quanto mais próxima essa métrica estiver de 1, menor o número de falsos positivos previstos pelo modelo.

A *F1-Score* é uma métrica que busca o equilíbrio entre as medidas de *Precision* e *Recall* (HOSSIN; SULAIMAN, 2015). A equação pode ser entendida da seguinte forma:

$$\frac{2 * p * r}{p + r} \tag{2.4}$$

2.7 Python

Python é uma linguagem de programação de alto nível interpretada empregada em diversas aplicações, como desenvolvimento web, ciência de dados, análise de dados e inteligência artificial. A enorme disponibilidade de recursos e bibliotecas para essa linguagem a proporciona um alto nível de adaptabilidade para o desenvolvimento dos mais diversos tipos de projetos.

Para o desenvolvimento desse projeto relacionado com a área de *Machine Learning*, as seguintes bibliotecas foram empregadas: *Pandas*, *Scikit-Learn*, *Tensorflow* e *Matplotlib*. A seguir, serão descritas em detalhes as ferramentas mencionadas.

2.7.1 Pandas

O desenvolvimento do presente projeto exige a manipulação de uma enorme quantia de dados, muito por conta dos modelos de aprendizado de máquina que necessitam deles durante a fase de treinamento. Na maioria das ocasiões, os conjuntos de dados sobre algum tema específico estão em formato *.csv* (*comma-separated values*), arquivos que se assemelham muito à uma tabela. Usando a biblioteca pandas, arquivos *.csv* são facilmente lidos e transformados em *DataFrames*, estrutura de dados que os armazenam em formato tabular, facilitando a manipulação. Além disso, muitos pacotes usados para tarefas de *machine learning*, como o *scikit-learn*, usam a biblioteca *pandas* em seu código por conta das vantagens que um *DataFrame* possui em relação à outras estruturas de dados.

2.7.2 Scikit-Learn

O *scikit-learn* providencia inúmeros modelos de aprendizado de máquina prontos para serem treinados e avaliados, acelerando o desenvolvimento de projetos. Todavia, os modelos não são imutáveis e o pacote nos permite ajustá-los de acordo com a particularidade de cada problema, oferecendo flexibilidade para mudanças.

As funções que o *scikit-learn* oferece são essenciais para tarefa de *machine learning*, desde a fase de pré-processamento de dados, com funções que nos permitem lidar com dados que não estão em um formato adequado, normalização de valores e balanceamento dos dados, até a avaliação de modelos, com funções para calcular métricas de avaliação, por exemplo.

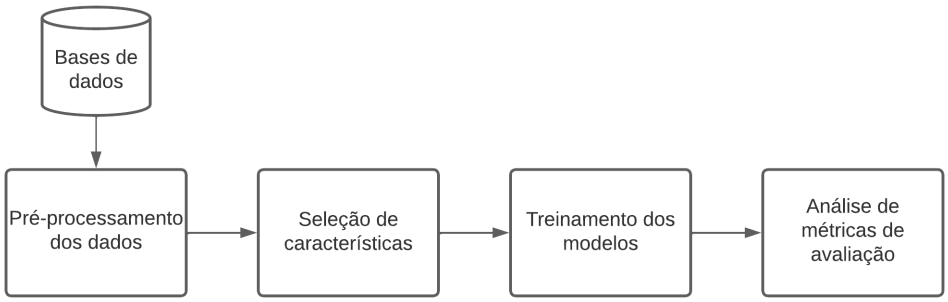
2.7.3 Matplotlib

A todo o momento, durante o desenvolvimento do projeto, dados estão sendo manipulados. Para observar o comportamento desses dados, esse pacote é usado. Com ele, conseguimos gerar desde figuras até gráficos para entender as informações com que estamos lidando, facilitando análises e tomadas de decisão.

3 Metodologia

Para o desenvolvimento desse projeto, foram consideradas quatro principais etapas conforme figura 5: pré-processamento dos dados, seleção de características, treinamento dos modelos e análise de desempenho.

Figura 5 – Metodologia geral proposta para o desenvolvimento do trabalho



Fonte: Elaborada pelo autor.

As etapas observadas na imagem podem ser descritas como segue:

1. **Pré-processamento de dados**: ambos os conjuntos de dados passarão por um cuidadoso tratamento e balanceamento, garantindo que estejam em condições ideais para análise.
2. **Seleção de características**: identificar os atributos mais relevantes para a tarefa de detecção de intrusão empregando o algoritmo *Random Forest*.
3. **Treinamento dos modelos**: ajustar os parâmetros de cada um dos algoritmos sugeridos procurando otimizá-los para que nossos modelos atinjam o melhor desempenho possível.
4. **Análise de métricas de avaliação**: examinar a eficácia de cada modelo em dois conjuntos de dados e avaliar o impacto dos algoritmos de *clustering*.

3.1 Coleta de dados

3.1.1 Kyoto Dataset +2006

Para embasar o desenvolvimento deste projeto, optamos pelo uso do conjunto de dados Kyoto de 2006. O Kyoto Dataset 2006 é uma extensa base de dados composta por informações extraídas a partir de tráfego real, coletado com a ajuda de honeypots da Universidade de Kyoto, no Japão (SONG et al., 2011).

É um conjunto de dados grande e robusto, possuindo amostras coletadas desde 2006 até 2015, cada uma com 24 atributos para análises. Todas as amostras presentes nesse conjunto de dados estão devidamente rotuladas, o que permite o emprego de modelos de aprendizado de máquina supervisionados para tarefas de classificação.

Tabela 2 – Descrição dos atributos do Kyoto Dataset +2006

Atributo	**Tipo**
Duration	float64
Service	object
Source bytes	int64
Destination bytes	int64
Count	int64
Same srv rate	float64
Serror rate	float64
Srv serror rate	float64
Dst host count	int64
Dst host srv count	int64
Dst host same src port rate	float64
Dst host serror rate	float64
Dst host srv serror rate	float64
Flag	object
IDS detection	object
Malware detection	object
Ashula detection	object
Label	int64
Source IP Address	object
Source Port Number	int64
Destination IP Address	object
Destination Port Number	int64
Start Time	object
Protocol	object

Fonte: Elaborada pelo autor.

Dentre os 24 atributos disponíveis para estudo conforme Tabela 2, os 14 primeiros são classificados como atributos convencionais, enquanto os outros 10 são classificados como atributos adicionais. Os atributos convencionais são baseados em um outro conjunto de dados, o KDD Cup 1999, que foi especificamente desenvolvido para detecção de intrusões.

Devido ao seu enorme tamanho, é inviável operar com todas as unidades que o conjunto de dados tem à disposição. Por motivos de simplificação, um número menor de amostras foi usado. Foram usadas amostras coletadas no período de um mês, resultado em aproximadamente 10 milhões de amostras. A escolha do mês foi determinada pela data de coleta mais recente do conjunto de dados, o que resultou nas amostras do mês de dezembro de 2015.

3.1.2 KDD Cup 1999 Data

Como dados que serão usados para testes, o conjunto de dados KDD Cup 1999 Data foi escolhido. Esse conjunto de dados foi elaborado para uma competição cuja tarefa era implementar um detector de intrusão de rede.

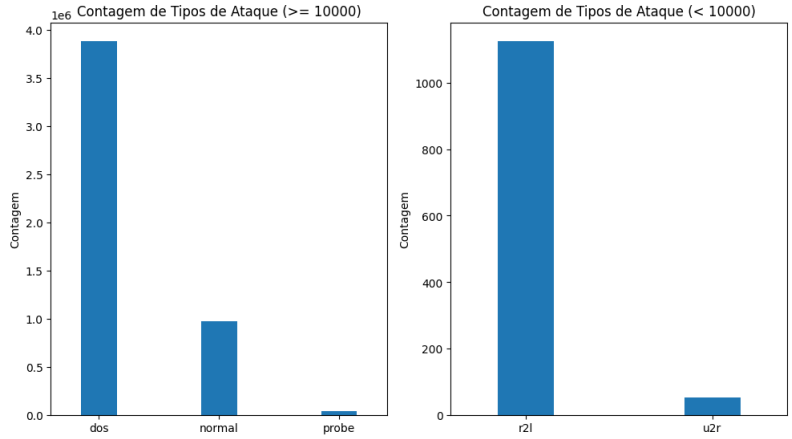
Tabela 3 – Descrição dos atributos do KDD Cup 1999

Atributo	**Tipo**
duration	int64
Protocol	int64
Service	int64
flag	object
src_bytes	int64
Destination bytes	int64
land	int64
wrong_fragment	int64
urgent	int64
hot	int64
num_failed_logins	int64
logged_in	int64
num_compromised	int64
root_shell	int64
su_attempted	int64
num_root	int64
num_file_creations	int64
num_shells	int64
num_access_files	int64
num_outbound_cmds	int64
is_host_login	int64

Atributo	**Tipo**
is_guest_login	int64
Count	int64
srv_count	int64
serror_rate	float64
srv_serror_rate	float64
rerror_rate	float64
srv_rerror_rate	float64
same_srv_rate	float64
diff_srv_rate	float64
srv_diff_host_rate	float64
Dst host count	int64
dst_host_srv_count	int64
dst_host_same_srv_rate	float64
dst_host_diff_srv_rate	float64
dst_host_same_src_port_rate	float64
dst_host_srv_diff_host_rate	float64
dst_host_serror_rate	float64
dst_host_srv_serror_rate	float64
dst_host_rerror_rate	float64
dst_host_srv_rerror_rate	float64
Label	int64

Os dados desse conjunto de dados representam invasões simuladas em um ambiente de rede controlado. As amostras possuem 42 atributos para análise, conforme observado na Tabela 3. Uma diferença do conjunto de dados KDD Cup 1999 Data para o conjunto de dados Kyoto é que cada uma das amostras é rotulada de acordo com o ataque que foi simulado. Ou seja, cada unidade desse conjunto de dados pode ser rotulada como sessão normal, ataque *Denial of Service*, ataque *Scanning*, ataque R2L ou ataque U2R, conforme observado no gráfico da figura 6.

Figura 6 – Distribuição de amostras do conjunto de dados KDD Cup 1999



Fonte: Elaborada pelo autor.

3.2 Pré-processamento de dados

3.2.1 Tratamento

O conjunto de dados é dividido em arquivos .txt, distribuídos por dia, mês e ano. Para cada arquivo, toda linha representa uma coleta. Para cada coleta, os atributos são separados por espaços. Em razão disso, para iniciar o tratamento dos dados apropriadamente, foram coletados todos os arquivos do mês de dezembro de 2015, convertendo-os para um único arquivo .csv, facilitando futuras manipulações. Dessa forma, os espaços entre as features foram substituídos por vírgulas e um cabeçalho para o conjunto de dados foi incluído. Após essa etapa, temos um arquivo final com aproximadamente 10 milhões de amostras para análise.

A próxima etapa envolveu a transformação dos atributos com diversos tipos de dados. Em um contexto de modelagem de aprendizado de máquina, é essencial que os dados estejam representados em formato numérico, uma vez que os modelos de máquina interpretam e processam apenas números. Nesse sentido, foram identificadas duas colunas que não estavam nesse formato: *Service* e *Protocol*. Essas colunas precisaram ser convertidas para uma representação numérica adequada. Vale ressaltar que outras colunas também continham dados em formato não numérico, porém, não foram alvo de modificação nessa etapa, uma vez que seriam posteriormente removidas do conjunto de dados.

Por fim, a coluna de etiqueta também foi modificada. Para essa coluna, cada amostra poderia apresentar um dos seguintes números: 1, para uma amostra que não identifica qualquer ataque; -1, para uma amostra que identifica um ataque conhecido; -2, para uma amostra que identifica um ataque desconhecido. Como a ideia é desenvolver um classificador binário, ataques conhecidos foram tratados da mesma maneira que ataques desconhecidos. Dessa forma, a coluna foi modificada de forma que as sessões normais passaram a ser representadas pelo

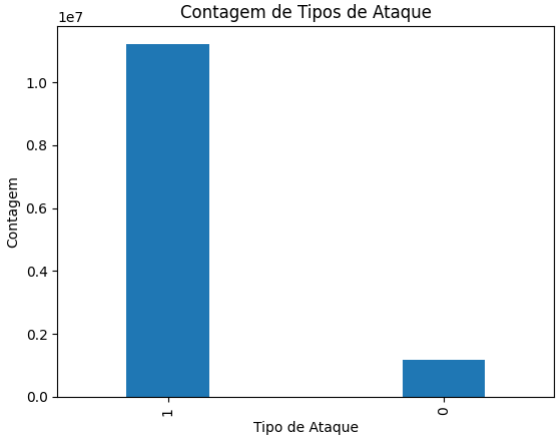
número 0, enquanto sessões de ataque, tanto conhecidos como desconhecidos, representadas pelo número 1.

Após a conclusão da etapa de conversão de todas as características para formatos numéricos, os dados passaram por um processo de normalização, realizado por meio da função *Normalizer*, disponível na biblioteca Scikit-Learn. Esse procedimento tem como resultado a padronização dos valores dos atributos, garantindo que eles estejam dentro de uma escala uniforme e assegurando a consistência e comparabilidade dos dados ao longo do processo de análise e modelagem.

3.2.2 Balanceamento

Para extraírmos os melhores resultados de modelos de aprendizado de máquina, é fundamental que os dados não estejam enviesados. Em outras palavras, os dados devem estar apropriadamente balanceados. Após as etapas de pré-processamento, os valores para a coluna *label* estão conforme o gráfico da figura 7.

Figura 7 – Comparação das amostras desbalanceadas de sessão normal e ataque



Fonte: Elaborada pelo autor.

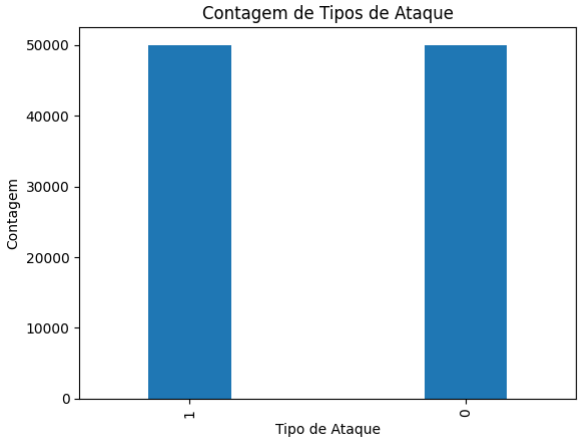
O conjunto de dados apresenta um desequilíbrio notável no número de sessões com rótulo igual a 1 (indicando ataques) em comparação com o número de sessões com rótulo igual a 0 (indicando sessões normais). Esta disparidade seria problemática para o treinamento de modelos preditivos, pois a maioria dos dados com os quais o modelo entraria em contato seriam sessões anormais. Isso levaria a um viés em direção a uma alta precisão de detecção de ataques, devido ao desequilíbrio dos dados.

Para superar esse desafio, foi implementada uma técnica de balanceamento conhecida como *Downsampling*. Essa técnica tem como objetivo reduzir o número de amostras da classe majoritária para igualá-lo ao número de amostras da classe minoritária. No contexto deste

estudo, o objetivo era igualar o número de amostras com rótulo 1 ao número de amostras com rótulo 0.

Como resultado desse procedimento de balanceamento, o conjunto de dados foi reconfigurado de modo que as duas classes tivessem aproximadamente o mesmo número de amostras. Isso permitiu que o modelo fosse treinado de forma mais equilibrada e justa, evitando o viés devido ao desequilíbrio original dos dados. O impacto desse balanceamento pode ser observado no gráfico de barras apresentado na Figura 8.

Figura 8 – Comparação das amostras balanceadas de sessão normal e ataque



Fonte: Elaborada pelo autor.

Durante o processo de balanceamento, aproveitamos a oportunidade para reduzir significativamente o tamanho total do conjunto de dados. Originalmente composto por 10 milhões de amostras, o conjunto foi ajustado para conter aproximadamente 100 mil amostras, divididas igualmente em 50 mil amostras benignas e 50 mil amostras malignas. Essa quantidade reduzida de amostras será utilizada nas etapas de seleção de características e de treinamento dos modelos. Essa redução no tamanho do conjunto de dados contribuirá para uma análise mais eficiente e um processo de treinamento de modelos mais ágil.

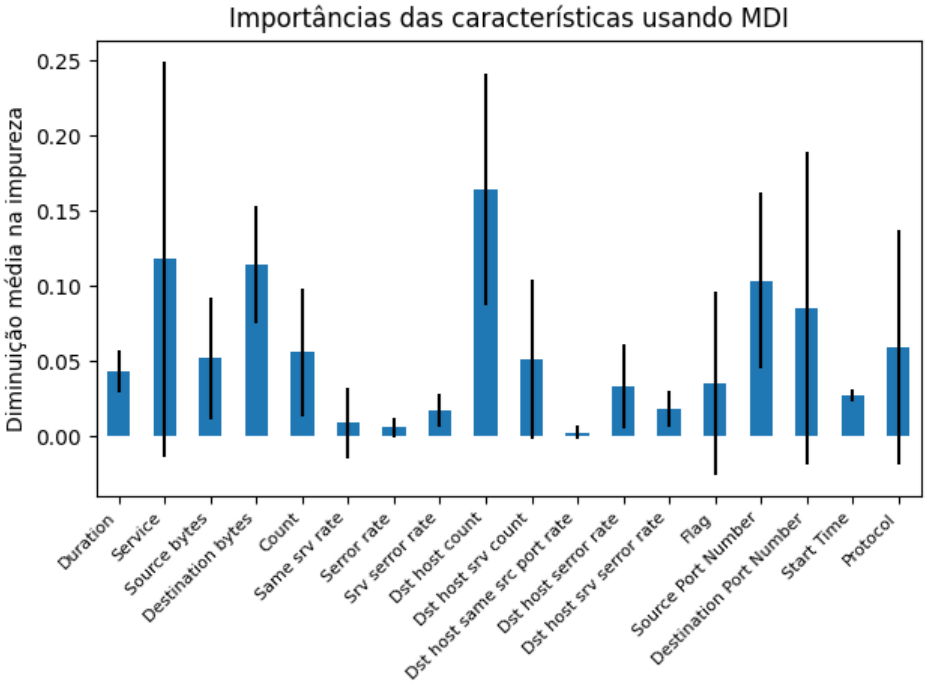
3.3 Seleção de características

A seleção das características exerce uma influência direta na capacidade de detecção de ataques cibernéticos e é uma etapa fundamental no desenvolvimento de modelos preditivos. Como exemplo, considere a detecção de ataques do tipo *smurf*, que pode ser realizada analisando o tipo de serviço nos pacotes. No caso de um ataque *smurf*, um indicador observável é a súbita inundação de *echo ICMP replies* direcionados à máquina vítima, a qual não enviou nenhum *echo ICMP* em primeiro lugar (MISHRA et al., 2018).

Para a extração das características mais significativas do conjunto de dados, empregou-se o algoritmo de aprendizado de máquina denominado *Random Forest*. Esse algoritmo se baseia na combinação de várias árvores de decisão, utilizando a técnica de *Ensemble Learning*, o que resulta em um modelo notavelmente preciso. O modelo foi treinado com todas as características disponíveis no conjunto de dados e alcançou uma acurácia de aproximadamente 99%. A partir desse resultado, é possível identificar as características que mais influenciaram o treinamento desse modelo.

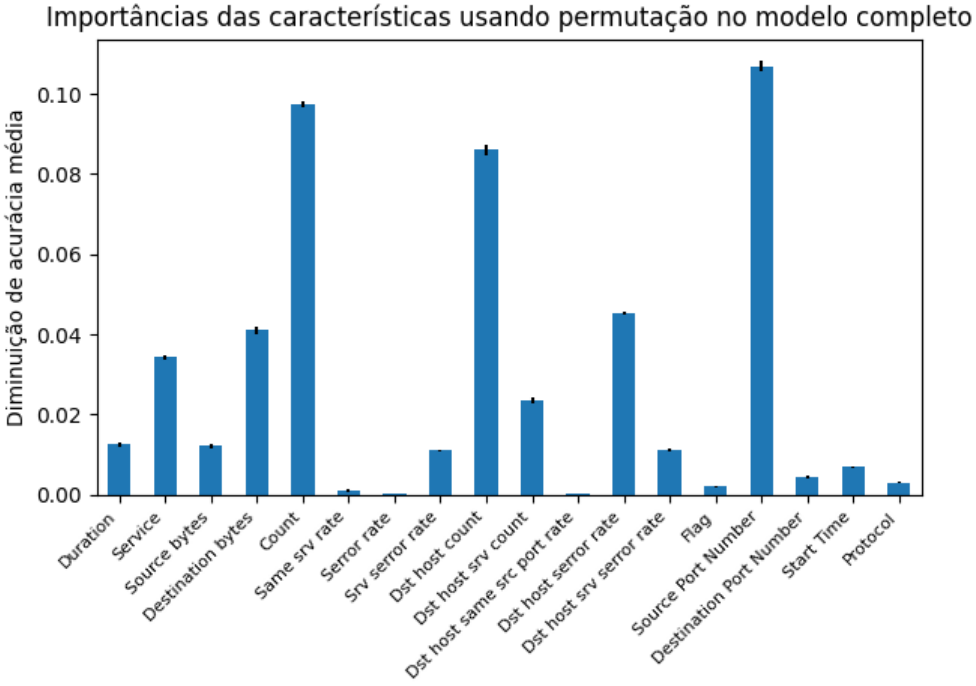
A implementação do modelo *Random Forest*, o treinamento e a extração das características relevantes foram realizados com o auxílio da biblioteca *Scikit-Learn*. Duas abordagens distintas foram aplicadas para a extração dessas características: uma utilizando o método *Mean Decrease in Impurity* e a outra empregando a técnica de *Feature Permutation*. Os resultados obtidos por ambas as abordagens podem ser visualizados conforme os gráficos das figuras 9 e 10:

Figura 9 – Importância das características usando *Mean Decrease in Impurity*



Fonte: Elaborada pelo autor.

Figura 10 – Importância das características usando *Feature Permutation*



Fonte: Elaborada pelo autor.

Analisando ambos os gráficos, observa-se que algumas características se destacam em relação às outras. Para o desenvolvimento do atual projeto, os seguintes atributos foram selecionados: *Service*, *Destination bytes*, *Count*, *Destination Host Count*, *Protocol* e *Label*.

3.4 Treinamento dos modelos

Ao longo dessa seção, serão descritos os treinamentos de três algoritmos de aprendizado de máquina: Support Vector Machine, Multilayer Perceptron e K-Nearest Neighbours. Todos os modelos foram treinados usando o conjunto de dados Kyoto +2006, conjunto de dados que oferece informações de tráfego real coletado por meio de uma rede de *honeypots*. Essa etapa possui como objetivo encontrar os melhores parâmetros para cada modelo de aprendizado de máquina.

3.4.1 Support Vector Machine

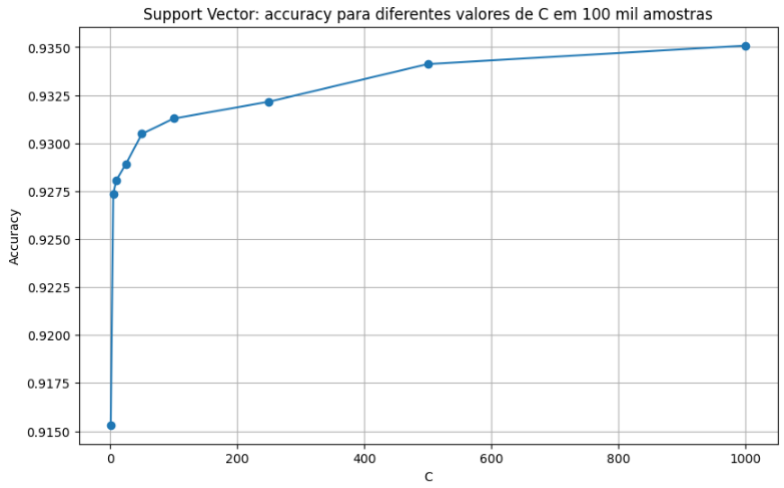
Para trabalhar com o conceito de SVM, o classificador Support Vector foi selecionado, disponível no pacote *scikit-learn*. É importante destacar que Support Vector Machines são muito usados em tarefas de detecção de intrusão, sendo a técnica de aprendizado de máquina mais eficaz quando usada em conjunto com outros classificadores (MISHRA et al., 2018). Em

resumo, um Support Vector Machine procura a melhor margem que separa um determinado conjunto de dados, independente de dimensões.

Em relação ao modelo SVC, uma das possibilidades que ele nos oferece é a mudança da função *kernel*. A biblioteca *scikit-learn* nos permite alternar entre quatro tipos: *linear*, *polinomial*, *sigmoidal* e *RBF*. No entanto, usaremos a função *RBF* definitivamente, visto que apresentou os melhores resultado quando comparada às outras funções em tarefas de detecção de intrusão (SCHERER et al., 2011).

Além disso, também podemos alterar o parâmetro C, valor responsável pelo controle de flexibilidade da margem entre as classes. Resumidamente, a penalização de erros é proporcional ao valor desse parâmetro. Nesse caso, os erros são as classes que estão do lado de "fora"da sua margem. Para projetar qual o melhor modelo SVC, foram testados diferentes valores para o parâmetro mencionado, respeitando a restrição de que o valor deve ser estritamente positivo: 1, 5, 10, 25, 50, 100, 250, 500 e 1000. Dessa forma, pode-se analisar o comportamento das métricas de avaliação em relação a cada parâmetro para determinar um suposto valor ideal.

Figura 11 – Support Vector: *accuracy* para diferentes valores de *C* em 100 mil amostras



Fonte: Elaborada pelo autor.

Analisando o gráfico em relação à *accuracy* conforme figura 11, observa-se que o modelo SVC que melhor performou sobre o conjunto de dados foi o mesmo quando o parâmetro *C* é igual a 1000. No entanto, observa-se que a performance do modelo melhora a medida que esse valor aumenta.

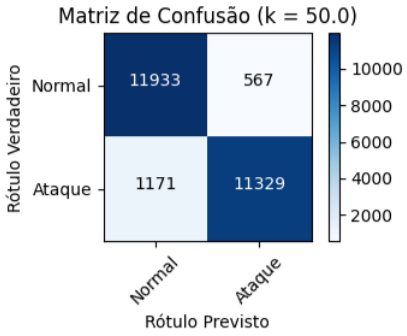
Tabela 4 – Support Vector: avaliação de modelos para diferentes valores de *C*

C	Acurácia	Precisão	Recall	F1-Score
n = 1	92%	95%	88%	91%
n = 5	93%	95%	90%	93%
n = 10	93%	95%	90%	93%
n = 25	93%	95%	90%	93%
n = 50	93%	95%	91%	93%
n = 100	93%	95%	91%	93%
n = 250	93%	95%	91%	93%
n = 500	93%	95%	91%	93%
n = 1000	94%	95%	91%	93%

Fonte: Elaborada pelo autor.

Em relação às métricas de avaliação conforme tabela 4, pode-se confirmar que o melhor valor para *C* dentre os analisados é 50. Embora todos os modelos tenham performado bem, nota-se que, a partir de um momento, aumentar o valor de *C* proporciona pouquíssimos resultados nas métricas de avaliação, além de desacelerar o processo de treinamento e classificação do modelo.

Figura 12 – Support Vector: matriz de confusão para valor de *C* igual a 50



Fonte: Elaborada pelo autor.

Observando a matriz de confusão para o melhor modelo conforme figura 12, nota-se que, em um conjunto de 25 mil amostras, aproximadamente 1800 amostras foram classificadas incorretamente. Considerando trabalhos correlatos que empregam o uso de Support Vector Machines para tarefas de detecção de intrusão, uma *accuracy* geral de 93% é um bom resultado para um classificador único. Em outras pesquisas, nota-se que alguns modelos atingiram uma precisão de aproximadamente 98% usando um conjunto de dados reduzido do KDD Cup 1999 e apenas o SVM como classificador (LI et al., 2012), enquanto outros atingiram uma precisão de aproximadamente 99% usando SVM em conjunto com outros modelos (MISHRA et al., 2018).

3.4.2 Multilayer perceptron

Um MLP (*multilayer perceptron*) é uma rede neural artificial composta por unidades que simulam os neurônios do sistema nervoso humano. Possivelmente, é um dos modelos mais conhecidos quando falamos sobre *machine learning*. Esse modelo pode ser implementado usando a biblioteca *scikit-learn* ou *Tensorflow*.

Um dos pontos fundamentais para construir um MLP é identificar a característica do problema que o algoritmo deverá resolver. No caso do atual projeto, a MLP deve resolver uma tarefa de classificação binária, ou seja, o algoritmo deve ser capaz de distinguir o conjunto de dados em duas classes distintas. Para isso, a última camada desse modelo possui uma função de ativação *sigmoid*, que é responsável por gerar valores de saída na rede entre 0 e 1. A função *sigmoid* é formulada como segue:

$$f(x) = \frac{1}{(1 + exp(-x))} \tag{3.1}$$

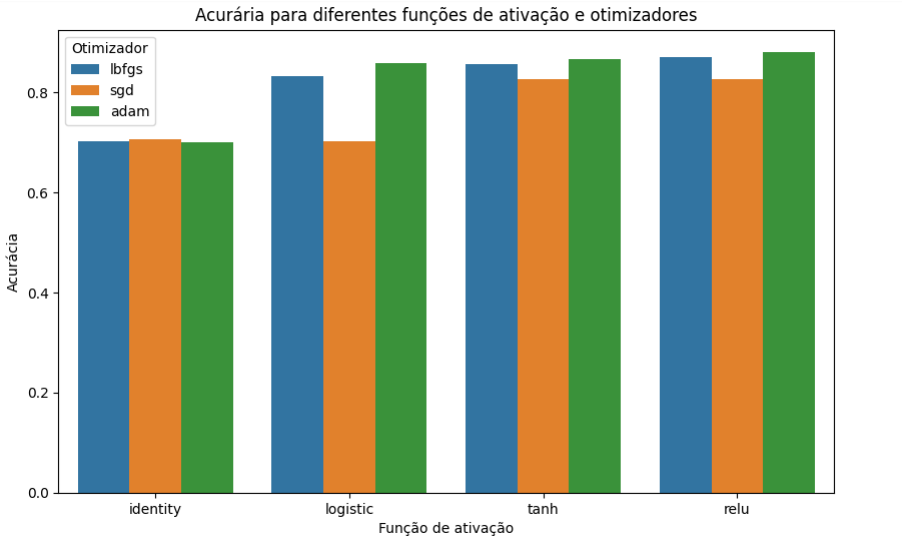
Além disso, a MLP pensada para a tarefa de detecção de intrusão tem duas camadas intermediárias de neurônios, uma com 128 e outra com 64 respectivamente. Para selecionar o melhor modelo de acordo com essa estrutura, foram testadas diversas combinações de funções de ativação e funções de otimização.

Tabela 5 – Multilayer Perceptron: avaliação dos modelos para diferentes combinações de ativação e otimização

Ativação	**Otimizador**	**Acurácia**	**Precisão**	**Recall**	**F1-Score**
identity	lbfgs	68%	73%	59%	65%
identity	sgd	68%	72%	61%	66%
identity	adam	68%	73%	59%	65%
logistic	lbfgs	83%	86%	79%	82%
logistic	sgd	68%	72%	61%	66%
logistic	adam	82%	87%	76%	81%
tanh	lbfgs	86%	88%	84%	86%
tanh	sgd	69%	73%	61%	66%
tanh	adam	85%	89%	81%	85%
relu	lbfgs	88%	88%	87%	88%
relu	sgd	80%	85%	73%	78%
relu	adam	86%	90%	82%	86%

Fonte: Elaborada pelo autor.

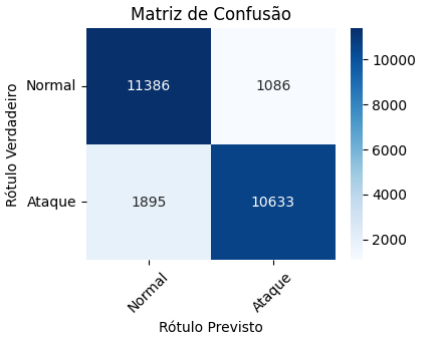
Figura 13 – Multilayer Perceptron: comparação dos modelos para diferentes combinações de ativação e otimização



Fonte: Elaborada pelo autor.

Dentre os modelos treinados, observa-se que o modelo que obteve o melhor desempenho geral conforme tabela 5 e gráfico da figura 13 foi o mesmo que possuiu como parâmetro a função de ativação *ReLU* e os otimizadores *adam* e *lbfgs*. A menor taxa de falsos positivos ocorre quando usamos o otimizador *adam*, atigindo uma *precision* de aproximadamente 90%, enquanto o melhor resultado geral ocorre quando usamos o otimizador *lbfgs*.

Figura 14 – Multilayer Perceptron: matriz de confusão para ativador *ReLU* e otimizador *Adam*



Observando a matriz de confusão conforme figura 14, nota-se que, em um conjunto de 25 mil amostras, aproximadamente 3000 amostras foram classificadas incorretamente. Considerando outros modelos de *machine learning* usados para tarefas de detecção, o MLP obteve uma performance um pouco abaixo, atingindo uma *accuracy* de aproximadamente 86% quando empregamos o otimizador *adam* e a função de ativação *ReLU*

3.4.3 Nearest Neighbours

O classificador K-Nearest Neighbours, disponível no pacote *scikit-learn*, se baseia na seguinte abordagem: um ponto é classificado de acordo com os pontos adjacentes à ele. Em outras palavras, um ponto em análise será classificado de acordo com a classe com o maior número de representantes vizinhos ao ponto. Esse modelo é muito comum na detecção de intrusão por anomalia, podendo ser abordado de duas maneiras distintas: de acordo com a distância entre os pontos ou a densidade de cada ponto (MISHRA et al., 2018). Embora seja um modelo que possua uma fase de treinamento rápida, sua classificação é um pouco mais demorada.

Um parâmetro que influencia diretamente nos resultados de classificação desse modelo é o número de pontos vizinhos ou adjacentes. "A escolha ideal do valor é altamente dependente de dados: em geral, um valor maior suprime os efeitos de ruído, mas torna os limites de classificação menos distintos"(PEDREGOSA et al., 2011). Sumariamente, a classificação é mais precisa quando o número de vizinhos é menor, enquanto valores discrepantes influenciam menos nos resultados quando esse número é maior. A biblioteca *scikit-learn* permite a manipulação desse valor e por isso, para fins de análise e seleção, o modelo foi treinado assumindo alguns valores para esse parâmetro: 5, 10, 25, 50, 100, 250, 500 e 1000.

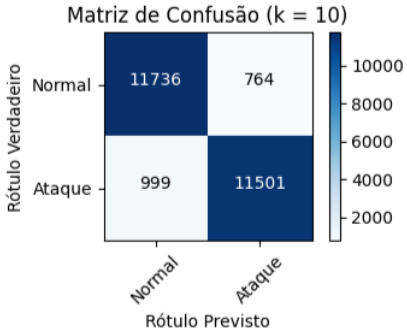
Tabela 6 – K-Nearest Neighbours: avaliação de modelos para diferentes números de vizinhos

Vizinhos	**Acurácia**	**Precisão**	**Recall**	**F1-Score**
n = 5	93%	93%	92%	93%
n = 10	93%	94%	92%	93%
n = 25	93%	94%	91%	93%
n = 50	92%	94%	90%	92%
n = 100	92%	94%	89%	92%
n = 250	91%	94%	88%	90%
n = 500	90%	94%	87%	90%
n = 1000	90%	94%	86%	89%

Fonte: Elaborada pelo autor.

Dentre os modelos treinados, o modelo que obteve o melhor desempenho geral confore tabela 6 foi o mesmo que possuiu como parâmetro o valor 10. Todos os modelos performaram relativamente bem, apresentando valores por volta de 90% em métricas de avaliação. No entanto, a menor taxa de falsos positivos e também falsos negativos ocorre quando o valor 10 é atribuído ao parâmetro de vizinhos do modelo, uma vez que os valores de *precision* e *recall* são os maiores dentre todos. Observa-se também que, como mencionado anteriormente, a distinção de classes diminui quando o número de vizinhos aumenta, refletindo diretamente nos valores de *accuracy*. A abordagem usada para o treinamento do modelo foi a de distância, situação onde pontos distantes influenciam menos do que pontos próximos para a classificação de um ponto de consulta.

Figura 15 – K-Nearest Neighbours: matriz de confusão para número de vizinhos igual a 10



Fonte: Elaborada pelo autor.

Analisando a matriz de confusão do modelo conforme figura 15 que obteve melhores resultados, observa-se que, dentro de 25 mil amostras, menos de 2 mil amostras foram classificadas incorretamente. Para um classificador único, uma *accuracy* geral de aproximadamente 93% mostra-se um bom resultado quando comparado ao estado da arte.

4 Experimentos e resultados

Todos os algoritmos foram previamente treinados de acordo com o *dataset* Kyoto +2006. Para avaliar os modelos e também a técnica de *ensemble learning*, o *dataset* KDD Cup 1999 também foi considerado. O primeiro conjunto de dados foi construído a partir de tráfego real com o auxílio de *honeypots*, enquanto o segundo foi construído a partir de um ambiente simulado. A ideia dos testes é avaliar a eficiência dos algoritmos e das técnicas de *clustering* usadas e analisar os resultados obtidos empregando dados de *honeypots* para o treinamento dos modelos preditivos.

Para os testes, usaremos subconjuntos menores de ambos os *datasets*. Cada subconjunto possui 20 mil amostras, divididas em 10 mil amostras benignas e 10 mil malignas. Vale lembrar que o subconjunto para treinamento é dividido na proporção 8 para 2 para poder avaliar o modelo durante a fase de treinamento. Além disso, ao longo dos testes, serão analisadas quatro métricas de avaliação: *Accuracy*, *Precision*, *Recall* e *F1-Score*.

4.1 Classificadores

O teste foi elaborado no intuito de avaliar a eficiência dos algoritmos de maneira geral.

Tabela 7 – Comparação dos modelos testando com o dataset Kyoto +2006

Algoritmo	**Acurácia**	**Precisão**	**Recall**	**F1-Score**
SVM	86%	90%	81%	85%
MLP	88%	86%	91%	88%
KNN	90%	89%	92%	90%

Fonte: Elaborada pelo autor.

Tabela 8 – Comparação dos modelos testando com o dataset KDD Cup 1999

Algoritmo	**Acurácia**	**Precisão**	**Recall**	**F1-Score**
SVM	80%	76%	88%	90%
MLP	75%	67%	99%	90%
KNN	72%	68%	82%	90%

Fonte: Elaborada pelo autor.

Como observado na tabela 7, todos os algoritmos implementados obtiveram uma ótima performance quando usado o dataset Kyoto +2006. Esse desempenho era esperado, visto que a escolha para os algoritmos foi baseada em resultados de outras pesquisas na área de detecção

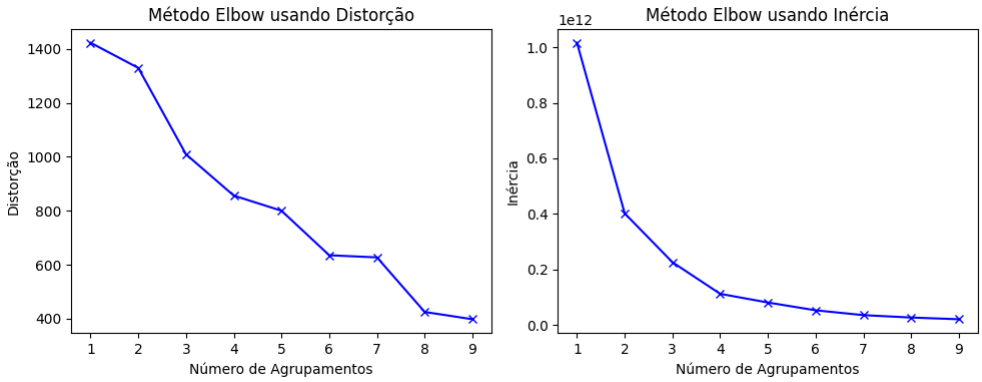
de intrusão com aprendizado de máquina que demonstram que essas técnicas atingiram ótimas performances (MISHRA et al., 2018).

Quando usado o dataset KDD Cup 1999, o resultado é um pouco diferente, caindo o rendimento dos modelos conforme tabela 8. Uma possível explicação para isso é a natureza de cada um dos conjuntos de dados: o conjunto de dados Kyoto +2006, como explicado anteriormente, foi gerado a partir de tráfego real, enquanto o conjunto de dados KDD Cup 1999 foi gerado a partir de tráfego simulado especificamente para uma competição de detecção de intrusão. Por conta disso, esse último conjunto de dados possui informações muito mais abrangentes do que aquelas coletadas diretamente de tráfego real. Como consequência, os algoritmos não conseguem generalizar o suficiente a partir dos dados do conjunto de dados Kyoto +2006 para manter a mesma precisão quando testadas com o outro conjunto de dados. Nesse primero teste, observamos que algoritmo mais prejudicado foi o KNN, enquanto o que melhor manteve sua performance foi o SVM.

4.2 Clustering K-Means

O teste foi elaborado no intuito de avaliar a eficiência da técnica de *clustering K-Means*. Usando essa técnica, *clusters* foram formados e uma nova coluna em cada conjunto de dados foi criada. Dessa forma, os modelos preditivos possuem uma nova *feature* para levar em consideração em suas predições. No entanto, é necessário informar ao algoritmo *K-Means* o número de *clusters* que devem ser formados. Para isso, usamos o *Elbow method* para identificar a quantidade ideal de *clusters* para o problema.

Figura 16 – *Elbow method*: encontrando o número ideal de *clusters*



Fonte: Elaborada pelo autor.

De acordo com o resultado da técnica conforme figura 16, observamos que o número ideal de *clusters* para o problema é 2, 5 ou 8. Eles podem ser notados pelos "cotovelos"gerados no gráfico. Os "cotovelos"são pontos que, a partir dele, a distorção ou a inércia começa a diminuir de forma linear.

Tabela 9 – Comparação dos modelos testando com o dataset Kyoto +2006 e *K-Means* (k=2)

Algoritmo	Acurácia	Precisão	Recall	F1-Score
SVM	86%	90%	81%	85%
MLP	89%	91%	87%	89%
KNN	90%	89%	92%	90%

Fonte: Elaborada pelo autor.

Tabela 10 – Comparação dos modelos testando com o dataset KDD Cup 1999 e *K-Means* (k=2)

Algoritmo	Acurácia	Precisão	Recall	F1-Score
SVM	80%	76%	88%	90%
MLP	75%	67%	99%	90%
KNN	72%	68%	82%	90%

Fonte: Elaborada pelo autor.

Quando observamos os resultados da técnica *K-Means* com 2 *clusters* conforme tabela 9, não há melhora quando usado o dataset Kyoto +2006. Isso indica que os grupos formados pelo algoritmos não agregam informações relevantes para as predições.

No entanto, quando usado o dataset KDD Cup 1999, observamos uma notável melhora no algoritmo MLP, aumentando 3% em sua taxa de *accuracy* conforme tabela 10.

Tabela 11 – Comparação dos modelos testando com o dataset Kyoto +2006 e *K-Means* (k=5)

Algoritmo	Acurácia	Precisão	Recall	F1-Score
SVM	86%	90%	81%	85%
MLP	89%	88%	90%	89%
KNN	90%	89%	92%	90%

Fonte: Elaborada pelo autor.

Tabela 12 – Comparação dos modelos testando com o dataset KDD Cup 1999 e *K-Means* (k=5)

Algoritmo	Acurácia	Precisão	Recall	F1-Score
SVM	80%	76%	88%	90%
MLP	74%	66%	99%	90%
KNN	72%	68%	82%	90%

Fonte: Elaborada pelo autor.

Não é possível observar melhoras nos resultados da técnica *K-Means* com 5 *clusters*, ocorrendo até pior desempenho em algumas métricas, como a *accuracy* do MLP caindo de 75% para 74% conforme tabelas 9 e 11.

Tabela 13 – Comparação dos modelos testando com o dataset Kyoto +2006 e *K-Means* (k=8)

Algoritmo	**Acurácia**	**Precisão**	**Recall**	**F1-Score**
SVM	86%	90%	81%	85%
MLP	86%	83%	92%	87%
KNN	90%	89%	92%	90%

Fonte: Elaborada pelo autor.

Tabela 14 – Comparação dos modelos testando com o dataset KDD Cup 1999 e *K-Means* (k=8)

Algoritmo	**Acurácia**	**Precisão**	**Recall**	**F1-Score**
SVM	80%	76%	88%	90%
MLP	72%	64%	99%	90%
KNN	72%	68%	82%	90%

Fonte: Elaborada pelo autor.

Não é possível observar melhoras nos resultados da técnica *K-Means* com 8 *clusters*, sendo a pior situação dentre as três analisadas conforme tabelas 13 e 14.

4.3 Clustering DBSCAN

O teste foi elaborado no intuito de avaliar a eficiência da técnica de *clustering DBSCAN*. Assim como no teste com a técnica *K-Means*, *clusters* foram formados e uma nova coluna em cada conjunto de dados foi criada. Lembrando que é necessário informar ao algoritmo *DBSCAN* o valor de *Eps* e *MinPts* para a formação de grupos.

Tabela 15 – Comparação dos modelos testando com o dataset Kyoto +2006 e *DBSCAN* (Eps=10, MinPts=10)

Algoritmo	**Acurácia**	**Precisão**	**Recall**	**F1-Score**
SVM	90%	90%	91%	90%
MLP	89%	92%	85%	88%
KNN	91%	90%	93%	91%

Fonte: Elaborada pelo autor.

Tabela 16 – Comparação dos modelos testando com o dataset KDD Cup 1999 e *DBSCAN* (Eps=10, MinPts=10)

Algoritmo	**Acurácia**	**Precisão**	**Recall**	**F1-Score**
SVM	62%	58%	88%	91%
MLP	70%	63%	99%	91%
KNN	65%	62%	81%	91%

Fonte: Elaborada pelo autor.

Para os valores de *Eps* 10 e *MinPts* 10, observa-se uma melhora no desempenho quando comparamos os modelos testando com o dataset Kyoto +2006 conforme tabela 15. O algoritmo mais beneficiado nesse teste foi o SVM, ganhando quatro pontos de porcentagem em *accuracy*. No entanto, o algoritmo que obteve a menor taxa de falsos positivos foi o MLP, atingindo uma *precision* de 92%.

Por outro lado, quando observamos os testes com o dataset KDD Cup 1999 conforme tabela 16, o desempenho cai consideravelmente. Isso indica que os *clusters* formados pelo algoritmo estão gerando informações específicas apenas ao conjunto de dados em que foi treinado. Uma das possíveis causas é o valor de *Eps* muito baixo, o que ocasiona a geração de muitos agrupamentos. Mesmo nesse caso, é válido notar que o algoritmo MLP obteve um *recall* de aproximadamente 100%, indicando que o número de falsos negativos classificados pelo algoritmo é próximo de 0.

Tabela 17 – Comparação dos modelos testando com o dataset Kyoto +2006 e *DBSCAN* (Eps=100, MinPts=10)

Algoritmo	**Acurácia**	**Precisão**	**Recall**	**F1-Score**
SVM	86%	90%	81%	85%
MLP	88%	91%	84%	88%
KNN	90%	89%	92%	90%

Fonte: Elaborada pelo autor.

Tabela 18 – Comparação dos modelos testando com o dataset KDD Cup 1999 e *DBSCAN* (Eps=100, MinPts=10)

Algoritmo	**Acurácia**	**Precisão**	**Recall**	**F1-Score**
SVM	80%	76%	88%	90%
MLP	79%	70%	99%	90%
KNN	72%	68%	82%	90%

Fonte: Elaborada pelo autor.

Por um outro lado, quando aumentamos o valor de *Eps* para 100, observa-se uma performance conforme as tabelas 17 e 18 muito semelhante ao teste realizado com a técnica

de *clustering K-Means*. Isso indica que para situações mais abrangentes, um número menor de *clusters* torna o algoritmo mais eficiente.

5 Conclusão

Diante das análises executadas ao longo do desenvolvimento desse projeto, confirma-se que o emprego de algoritmos de *machine learning* para tarefas de detecção de intrusão é uma área promissora e que ainda há muito a ser explorado. Nesse projeto, foram analisadas apenas duas técnicas de *clustering* para análise de melhoria de desempenho: K-Means e DBSCAN. Essas técnicas cobrem apenas duas de quatro abordagens dentre os algoritmos de *clustering*, abrindo margem para projetos futuros.

Não menos importante, a aplicação do conceito de *honeypots* em tarefas de detecção de intrusão também apresentou bons resultados. Embora não tenha sido implementado uma rede com *honeypots*, foram usado dados de terceiros coletados com o auxílio dessas próprias ferramentas. Mesmo diante dessas circunstâncias, é possível observar a usabilidade de *honeypots* como meio de obtenção de dados para treinamento de modelos preditivos. As características de um *honeypot* permitem construir amostras de ataque rotuladas que servem como combustível para a implementação de algoritmos de aprendizado de máquina.

Referências

ALMANSOR, M.; GAN, K. B. Intrusion detection systems: Principles and perspectives. p. 2458–925, 11 2018. Disponível em: <https://www.researchgate.net/publication/332524050_Intrusion_Detection_Systems_Principles_And_Perspectives> . Acesso em: 25 de abril de 2023.

ASSUNCAO., M. F. *Honeypots E honeynets: Aprenda a detectar e enganar os invasores.* -. [S.l.]: Visual Books, 2009. Disponível em: <https://arxiv.org/abs/2010.11929> . Acesso em: 07 de abril de 2023.

CNN. *Levantamento mostra que ataques cibernéticos no Brasil cresceram 94Acesso em 30 de Agosto de 2023. Disponível em: <https://www.cnnbrasil.com.br/tecnologia/levantamento-mostra-que-ataques-ciberneticos-no-brasil-cresceram-94/>.*

GARDNER, M. W.; DORLING, S. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, Elsevier, v. 32, n. 14-15, p. 2627–2636, 1998. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S1352231097004470> . Acesso em: 18 de maio de 2023.

GRÉGIO, A.; SANTOS, R.; MONTES, A. Evaluation of data mining techniques for suspicious network activity classification using honeypots data. In: SPIE. *Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security 2007*. [S.l.], 2007. v. 6570, p. 55–64. Disponível em: <https://www.researchgate.net/publication/253261410_Evaluation_of_data_mining_techniques_for_suspicious_network_activity_classification_using_honeypots_data> . Acesso em: 23 de agosto de 2023.

HOSSIN, M.; SULAIMAN, M. N. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, Academy & Industry Research Collaboration Center (AIRCC), v. 5, n. 2, p. 1, 2015. Disponível em: <https://www.researchgate.net/publication/275224157_A_Review_on_Evaluation_Metrics_for_Data_Classification_Evaluations> . Acesso em: 14 de agosto de 2023.

IBM. *What is a cyberattack?* 2023. Disponível em: <https://www.ibm.com/topics/cyber-attack>. Acesso em 30 de Agosto de 2023.

KASPERSKY. O que é cibersegurança? *www.kaspersky.com.br*, 2023. Disponível em: <https://www.kaspersky.com.br/resource-center/definitions/what-is-cyber-security>. Acesso em 30 de Agosto de 2023.

KHAN, K.; REHMAN, S. U.; AZIZ, K.; FONG, S.; SARASVADY, S. Dbscan: Past, present and future. In: *The Fifth International Conference on the Applications of Digital Information and Web Technologies (ICADIWT 2014)*. [S.l.: s.n.], 2014. p. 232–238. Disponível em: <https://ieeexplore.ieee.org/document/6814687> . Acesso em: 28 de setembro de 2023.

KODINARIYA, T. M.; MAKWANA, P. R. et al. Review on determining number of cluster in k-means clustering. *International Journal*, v. 1, n. 6, p. 90–95, 2013. Disponível em: <https://www.researchgate.net/publication/313554124_Review_on_Determining_of_Cluster_in_K-means_Clustering> . Acesso em: 27 de setembro de 2023.

LI, Y.; XIA, J.; ZHANG, S.; YAN, J.; AI, X.; DAI, K. An efficient intrusion detection system based on support vector machines and gradually feature removal method. *Expert systems with applications*, Elsevier, v. 39, n. 1, p. 424–430, 2012. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S0957417411009948> . Acesso em: 11 de setembro de 2023.

MISHRA, P.; VARADHARAJAN, V.; TUPAKULA, U.; PILLI, E. S. A detailed investigation and analysis of using machine learning techniques for intrusion detection. *IEEE communications surveys & tutorials*, IEEE, v. 21, n. 1, p. 686–728, 2018. Disponível em: <https://ieeexplore.ieee.org/document/8386762> . Acesso em: 02 de junho de 2023.

NAWROCKI, M.; WäHLISCH, M.; SCHMIDT, T. C.; KEIL, C.; SCHöNFELDER, J. *A Survey on Honeypot Software and Data Analysis*. 2016. Disponível em: <https://arxiv.org/abs/1608.06249> . Acesso em: 24 de abril de 2023.

NOBLE, W. S. What is a support vector machine? *Nature biotechnology*, Nature Publishing Group UK London, v. 24, n. 12, p. 1565–1567, 2006. Disponível em: <https://www.researchgate.net/publication/6639744_What_is_a_Support_Vector_Machine> . Acesso em: 17 de maio de 2023.

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. Disponível em: <https://dl.acm.org/doi/10.5555/1953048.2078195> . Acesso em: 29 de junho de 2023.

SCHERER, P.; VICHER, M.; DRÁZDILOVÁ, P.; MARTINOVIC, J.; DVORSKY, J.; SNASEL, V. Using svm and clustering algorithms in ids systems. In: *Proc. Int Conf. Dateso 2011, 2011*. [S.l.: s.n.], 2011. Disponível em: <https://ceur-ws.org/Vol-706/paper25.pdf> . Acesso em: 17 de maio de 2023.

SONG, J.; TAKAKURA, H.; OKABE, Y.; ETO, M.; INOUE, D.; NAKAO, K. Statistical analysis of honeypot data and building of kyoto 2006+ dataset for nids evaluation. In: *Proceedings of the first workshop on building analysis datasets and gathering experience returns for security*. [S.l.: s.n.], 2011. p. 29–36. Disponível em: <https://dl.acm.org/doi/10.1145/1978672.1978676> . Acesso em: 15 de junho de 2023.

ZHANG, S.; LI, X.; ZONG, M.; ZHU, X.; CHENG, D. Learning k for knn classification. *ACM Trans. Intell. Syst. Technol.*, Association for Computing Machinery, New York, NY, USA, v. 8, n. 3, jan 2017. ISSN 2157-6904. Disponível em: <https://dl.acm.org/doi/abs/10.1145/2990508> . Acesso em: 19 de maior de 2023. Disponível em: <https://doi.org/10.1145/2990508>.