



UNIVERSIDADE AUTÓNOMA DE LISBOA
LUÍS DE CAMÕES

DEPARTAMENTO DE ENGENHARIAS E DE CIÊNCIAS DA
COMPUTAÇÃO
LICENCIATURA EM ENGENHARIA INFORMÁTICA

IDS POR MACHINE LEARNING

Unidade Curricular Laboratório de Projeto

Autores: Afonso Figueiredo Frasquilho; Guilherme Lopes Fernandes; Tomás Miguel
Rodrigues Viana

Orientador: Professor Doutor Mário Marques da Silva

Números: 30010929; 30010398; 30010623

Julho de 2025

Lisboa

(Página em Branco)

Resumo

O presente projeto consistiu no desenvolvimento de um Sistema de Detecção de Intrusões com recurso a técnicas de Machine Learning, destinado a identificar tráfego normal e tráfego anómalo em ambiente controlado com recurso a máquinas virtuais. Foram implementados e avaliados modelos supervisionados, como a Rede Neuronal Multi-Layer Perceptron, e não supervisionados, como o Autoencoder. Cada um destes elementos foi desenvolvido, avaliado com base nos resultados obtidos e discutido em detalhe, contribuindo para o desenvolvimento e implementação da versão final do Sistema de Detecção de Intrusões.

Palavras-chave: Sistema de Detecção de Intrusões; Machine Learning; Tráfego de rede; Máquinas Virtuais.

Abstract

The present project consisted of developing an Intrusion Detection System using Machine Learning techniques, designed to identify normal and anomalous traffic in a controlled environment using virtual machines. Supervised models, such as the Multi-Layer Perceptron Neural Network, and unsupervised models, such as the Autoencoder, were implemented and evaluated. Each of these elements was developed, evaluated based on the obtained results, and discussed in detail, contributing to the development and implementation of the final version of the Intrusion Detection System.

Keywords: Intrusion Detection System; Machine Learning; Network Traffic; Virtual Machines.

Índice

| | |
|----------------------------------------------------|-----------|
| Resumo | 3 |
| Abstract | 3 |
| Índice | 4 |
| Lista de Quadros/Gráficos..... | 10 |
| Lista de Fotografias/Ilustrações | 10 |
| Lista de Siglas e Acrónimos..... | 11 |
| 1 Introdução..... | 12 |
| 2 Desenvolvimento..... | 14 |
| 2.1 Conjunto de Dados | 17 |
| 2.1.1 Preparação do Dataset | 17 |
| 2.1.2 Análise Exploratória | 18 |
| 2.2 Modelos de Machine Learning..... | 19 |
| 2.2.1 Modelos Supervisionados Desenvolvidos | 19 |
| 2.2.1.1 Rede Neuronal Multi-Layer Perceptron | 20 |
| 2.2.1.1.1 Primeiro Teste | 20 |
| 2.2.1.1.2 Segundo Teste | 21 |
| 2.2.1.1.3 Terceiro Teste | 21 |
| 2.2.1.1.4 Quarto Teste | 22 |
| 2.2.1.1.5 Modelo Final | 22 |
| 2.2.1.2 Árvore de Decisão Random Forest..... | 23 |
| 2.2.1.2.1 Primeiro Teste | 23 |
| 2.2.1.2.2 Segundo Teste | 24 |
| 2.2.1.2.3 Terceiro Teste | 25 |
| 2.2.1.2.4 Quarto Teste | 25 |
| 2.2.1.2.5 Modelo Final | 26 |

| | | |
|--------------|-------------------------------------------------------------|-----------|
| 2.2.1.3 | Máquina de Vetores de Suporte | 27 |
| 2.2.1.3.1 | Primeiro Teste | 27 |
| 2.2.1.3.2 | Segundo Teste | 28 |
| 2.2.1.3.3 | Terceiro Teste | 28 |
| 2.2.1.3.4 | Quarto Teste | 28 |
| 2.2.1.3.5 | Modelo Final | 29 |
| 2.2.2 | Modelos Não Supervisionados Desenvolvidos | 30 |
| 2.2.2.1 | K-Means Clustering..... | 30 |
| 2.2.2.1.1 | Primeiro Teste | 30 |
| 2.2.2.1.2 | Segundo Teste | 31 |
| 2.2.2.1.3 | Terceiro Teste | 32 |
| 2.2.2.1.4 | Quarto Teste | 32 |
| 2.2.2.1.5 | Modelo Final | 33 |
| 2.2.2.2 | Isolation Forest | 33 |
| 2.2.2.2.1 | Primeiro Teste | 34 |
| 2.2.2.2.2 | Segundo Teste | 34 |
| 2.2.2.2.3 | Terceiro Teste | 35 |
| 2.2.2.2.4 | Quarto Teste | 35 |
| 2.2.2.2.5 | Modelo Final | 36 |
| 2.2.2.3 | Autoencoder | 36 |
| 2.2.2.3.1 | Primeiro Teste | 37 |
| 2.2.2.3.2 | Segundo Teste | 38 |
| 2.2.2.3.3 | Terceiro Teste | 38 |
| 2.2.2.3.4 | Quarto Teste | 39 |
| 2.2.2.3.5 | Modelo Final | 39 |
| 2.2.2.3.6 | Teste Threshold | 40 |
| 2.3 | Testes Preliminares com os Modelos de Machine Learning..... | 41 |

| | | |
|-----------|------------------------------------------------------------|----|
| 2.3.1 | Testes Preliminares nos Modelos Supervisionados | 41 |
| 2.3.2 | Testes Preliminares nos Modelos Não Supervisionados | 42 |
| 2.4 | Arquitetura Final do IDS..... | 43 |
| 2.5 | Configuração do Ambiente de Teste | 45 |
| 2.5.1 | Preparação das Máquinas Virtuais | 45 |
| 2.5.2 | Configuração da Rede Virtual | 49 |
| 2.5.3 | Configuração de Hardware das Máquinas Virtuais | 50 |
| 2.5.4 | Configuração do IDS em Modo Promíscoo | 52 |
| 2.5.5 | Integração do CICFlowMeter no Ambiente de Testes | 53 |
| 2.5.6 | Simulação de Ataques | 54 |
| 2.5.7 | Execução Automatizada do IDS..... | 60 |
| 3 | Avaliações e Resultados | 61 |
| 3.1 | Resultados Obtidos com o Dataset..... | 62 |
| 3.1.1 | Dimensão e Estrutura do Dataset | 62 |
| 3.1.2 | Estatísticas Descritivas dos Atributos | 62 |
| 3.1.3 | Distribuição das Classes..... | 63 |
| 3.1.4 | Correlação Entre Atributos..... | 64 |
| 3.2 | Métricas de Avaliação Utilizadas..... | 64 |
| 3.3 | Resultados Obtidos com os Modelos de Machine Learning..... | 68 |
| 3.3.1 | Modelos Supervisionados | 69 |
| 3.3.1.1 | Rede Neuronal Multi-Layer Perceptron | 69 |
| 3.3.1.1.1 | Primeiro Teste | 69 |
| 3.3.1.1.2 | Segundo Teste | 70 |
| 3.3.1.1.3 | Terceiro Teste | 70 |
| 3.3.1.1.4 | Quarto Teste | 71 |
| 3.3.1.1.5 | Modelo Final | 72 |
| 3.3.1.2 | Árvore de Decisão Random Forest..... | 73 |

| | | |
|--------------|-----------------------------------------|-----------|
| 3.3.1.2.1 | Primeiro Teste | 73 |
| 3.3.1.2.2 | Segundo Teste | 74 |
| 3.3.1.2.3 | Terceiro Teste | 75 |
| 3.3.1.2.4 | Quarto Teste | 76 |
| 3.3.1.2.5 | Modelo Final | 77 |
| 3.3.1.3 | Máquina de Vetores de Suporte | 78 |
| 3.3.1.3.1 | Primeiro Teste | 78 |
| 3.3.1.3.2 | Segundo Teste | 79 |
| 3.3.1.3.3 | Terceiro Teste | 80 |
| 3.3.1.3.4 | Quarto Teste | 81 |
| 3.3.1.3.5 | Modelo Final | 82 |
| 3.3.2 | Modelos Não Supervisionados..... | 84 |
| 3.3.2.1 | K-Means Clustering..... | 84 |
| 3.3.2.1.1 | Primeiro Teste | 84 |
| 3.3.2.1.2 | Segundo Teste | 84 |
| 3.3.2.1.3 | Terceiro Teste | 85 |
| 3.3.2.1.4 | Quarto Teste | 86 |
| 3.3.2.1.5 | Modelo Final | 87 |
| 3.3.2.2 | Isolation Forest | 87 |
| 3.3.2.2.1 | Primeiro Teste | 87 |
| 3.3.2.2.2 | Segundo Teste | 88 |
| 3.3.2.2.3 | Terceiro Teste | 89 |
| 3.3.2.2.4 | Quarto Teste | 89 |
| 3.3.2.2.5 | Modelo Final | 90 |
| 3.3.2.3 | Autoencoder | 91 |
| 3.3.2.3.1 | Primeiro Teste | 91 |
| 3.3.2.3.2 | Segundo Teste | 92 |

| | | |
|-----------|------------------------------------------------------------------------------------|-----|
| 3.3.2.3.3 | Terceiro Teste | 92 |
| 3.3.2.3.4 | Quarto Teste | 93 |
| 3.3.2.3.5 | Modelo Final | 94 |
| 3.3.2.3.6 | Teste Threshold | 94 |
| 3.4 | Resultados Obtidos com os Testes Preliminares aos Modelos de Machine Learning | 95 |
| 3.4.1 | Resultados Obtidos com os Testes Preliminares aos Modelos Supervisionados..... | 95 |
| 3.4.2 | Resultados Obtidos com os Testes Preliminares aos Modelos Não Supervisionados..... | 97 |
| 3.5 | Resultados Obtidos do Ambiente de Testes | 100 |
| 3.5.1 | Análise de Tráfego com Elevado Número de Fluxos | 100 |
| 3.5.2 | Análise de Tráfego com Fluxo Normal..... | 101 |
| 3.5.3 | Análise de Tráfego com Fluxo Anômalo | 102 |
| 4 | Discussão | 102 |
| 4.1 | Dataset..... | 102 |
| 4.2 | Modelos de Machine Learning..... | 104 |
| 4.2.1 | Modelos Supervisionados | 104 |
| 4.2.1.1 | Rede Neuronal Multi-Layer Perceptron | 104 |
| 4.2.1.2 | Árvore de Decisão Random Forest..... | 104 |
| 4.2.1.3 | Máquina de Vetores de Suporte | 105 |
| 4.2.2 | Modelos Não Supervisionados..... | 106 |
| 4.2.2.1 | K-Means Clustering..... | 106 |
| 4.2.2.2 | Isolation Forest | 106 |
| 4.2.2.3 | Autoencoder | 107 |
| 4.2.3 | Escolha dos Modelos para o IDS..... | 108 |
| 4.3 | Escolha da Arquitetura do IDS | 109 |
| 4.4 | Ambiente de Testes | 110 |

| | | |
|----------|---------------------------------------------------------------|------------|
| 5 | Conclusões..... | 111 |
| 6 | Trabalho futuro..... | 113 |
| | Bibliografia..... | 114 |
| | Anexo 01 – Manual de Instruções/Instalação | 115 |
| 1 | Instalação do Python..... | 115 |
| 2 | Instalação das Bibliotecas..... | 115 |
| 3 | Dataset..... | 115 |
| 3.1 | Instalação do Visual Studio Code | 115 |
| 3.2 | Abrir Pasta com os Ficheiros do Dataset | 115 |
| 3.3 | Gerar Ficheiro do Dataset | 116 |
| 3.4 | Análise Exploratória do Dataset..... | 116 |
| 4 | Modelos de Machine Learning..... | 117 |
| 5 | IDS | 118 |
| 5.1 | Execução do IDS por tipo de Modelo | 118 |
| 5.2 | Executar o IDS Utilizando a Arquitetura Definida | 119 |
| 6 | Ambiente de Testes | 119 |
| 6.1 | Instalação do VMware Workstation Pro | 119 |
| 6.2 | Obtenção da Imagem do Kali Linux | 120 |
| 6.3 | Importação da Máquina Virtual Kali Linux..... | 120 |
| 6.4 | Clonagem da Máquina Virtual | 121 |
| 6.5 | Criação da Rede Virtual VMnet2..... | 121 |
| 6.6 | Configuração do Adaptador de Rede | 122 |
| 6.7 | Acesso Temporário à Internet para Instalações..... | 122 |
| 6.8 | Instalação do CICFlowMeter | 123 |
| 6.9 | Adicionar a Pasta IDS e Configurar alias | 124 |
| 6.10 | Reconfigurar para a Rede VMnet2 | 124 |
| 6.11 | Ativar o Modo Promísceo na Interface eth0 da Máquina IDS..... | 124 |

| | | |
|------|---------------------------------|-----|
| 6.12 | Execução Final do Sistema | 125 |
|------|---------------------------------|-----|

Lista de Quadros/Gráficos

| | |
|----------------------------------------------------------|----|
| Tabela 1 - Configuração das Três Máquinas Virtuais | 50 |
|----------------------------------------------------------|----|

Lista de Fotografias/Ilustrações

| | |
|---------------------------------------------------------------------------------------------|----|
| Figura 1 - Project com o Planeamento Inicial | 14 |
| Figura 2 - Project com Alterações no Planeamento | 16 |
| Figura 3 - Versão Final do Project | 16 |
| Figura 4 - Exemplo de Relatório de Resultados de Testes Preliminares | 42 |
| Figura 5 - Fluxo de Funcionamento da Arquitetura do IDS | 44 |
| Figura 6 - Importação do Sistema Operativo Kali | 45 |
| Figura 7 - Clone Wizard para Cópias de Máquinas Virtuais Kali | 46 |
| Figura 8 - Estado do Clone..... | 47 |
| Figura 9 - Tipo de Clone | 48 |
| Figura 10 - Nome da Máquina e Localização da Máquina Física..... | 48 |
| Figura 11 - Criação da VMnet2..... | 50 |
| Figura 12 - Configuração das Máquinas Virtuais | 52 |
| Figura 13 - Configuração do IDS em Modo Promíscoo | 52 |
| Figura 14 - Definição da Interface Eth0 em Modo Promíscoo e Inicialização do script.sh..... | 54 |
| Figura 15 - Execução de Ataque TCP SYN Flood..... | 55 |
| Figura 16 - Termina do script.sh..... | 56 |
| Figura 17 - Pasta onde os Ficheiros .csv são Acedidos pelo IDS | 57 |
| Figura 18 - Estrutura de Diretórios Associados ao CICFlowMeter e ao IDS | 58 |
| Figura 19 - Geração de um Ficheiro Binário na Máquina Vítima | 58 |
| Figura 20 - Preparação da Máquina Atacante para Receção do Ficheiro via Netcat | 59 |
| Figura 21 - Envio do Ficheiro da Vítima para o Atacante via Netcat..... | 60 |
| Figura 22 - Comparação do Tamanho dos Ficheiros | 60 |
| Figura 23 - Distribuição de Classes do Dataset..... | 63 |
| Figura 24 - Matriz de Correlação do Dataset | 64 |
| Figura 25 - Relatório de Classificação e Matriz de Confusão do Teste 1 MLP..... | 69 |
| Figura 26 - Relatório de Classificação e Matriz de Confusão do Teste 2 MLP..... | 70 |
| Figura 27 - Relatório de Classificação e Matriz de Confusão do Teste 3 MLP..... | 71 |
| Figura 28 - Relatório de Classificação e Matriz de Confusão do Teste 4 MLP..... | 72 |
| Figura 29 - Matriz de Confusão do Modelo Final MLP | 73 |
| Figura 30 - F1-score em cada Fold na Validação Cruzada do Modelo Final MLP | 73 |
| Figura 31 - Relatório de Classificação e Matriz de Confusão do Teste 1 RF | 74 |
| Figura 32 - Relatório de Classificação e Matriz de Confusão do Teste 2 RF | 75 |
| Figura 33 - Relatório de Classificação e Matriz de Confusão do Teste 3 RF | 76 |
| Figura 34 - Relatório de Classificação e Matriz de Confusão do Teste 4 RF | 77 |
| Figura 35 - Matriz de Confusão do Modelo Final RF | 78 |

| | |
|----------------------------------------------------------------------------------------|-----|
| Figura 36 - F1-score em cada Fold na Validação Cruzada do Modelo Final RF | 78 |
| Figura 37 - Relatório de Classificação e Matriz de Confusão do Teste 1 SVM | 79 |
| Figura 38 - Relatório de Classificação e Matriz de Confusão do Teste 2 SVM | 80 |
| Figura 39 - Relatório de Classificação e Matriz de Confusão do Teste 3 SVM | 81 |
| Figura 40 - Relatório de Classificação e Matriz de Confusão do Teste 4 SVM | 82 |
| Figura 41 - Matriz de Confusão do Modelo Final SVM | 83 |
| Figura 42 - F1-score em cada Fold na Validação Cruzada do Modelo Final SVM | 83 |
| Figura 43 - Relatório de Classificação e Matriz de Confusão do Teste 1 KM | 84 |
| Figura 44 - Relatório de Classificação e Matriz de Confusão do Teste 2 KM | 85 |
| Figura 45 - Relatório de Classificação e Matriz de Confusão do Teste 3 KM | 86 |
| Figura 46 - Relatório de Classificação e Matriz de Confusão do Teste 4 KM | 86 |
| Figura 47 - Matriz de Confusão do Modelo Final KM | 87 |
| Figura 48 - Relatório de Classificação e Matriz de Confusão do Teste 1 IF | 88 |
| Figura 49 - Relatório de Classificação e Matriz de Confusão do Teste 2 IF | 88 |
| Figura 50 - Relatório de Classificação e Matriz de Confusão do Teste 3 IF | 89 |
| Figura 51 - Relatório de Classificação e Matriz de Confusão do Teste 4 IF | 90 |
| Figura 52 - Matriz de Confusão do Modelo Final IF | 91 |
| Figura 53 - Relatório de Classificação e Matriz de Confusão do Teste 1 AE | 91 |
| Figura 54 - Relatório de Classificação e Matriz de Confusão do Teste 2 AE | 92 |
| Figura 55 - Relatório de Classificação e Matriz de Confusão do Teste 3 AE | 93 |
| Figura 56 - Relatório de Classificação e Matriz de Confusão do Teste 4 AE | 93 |
| Figura 57 - Matriz de Confusão do Modelo Final AE | 94 |
| Figura 58 - Gráfico do Terceiro Teste Supervisionado | 96 |
| Figura 59 - Gráficos do Quarto Teste Supervisionado | 97 |
| Figura 60 - Gráficos do Primeiro Teste Não Supervisionado | 97 |
| Figura 61 - Gráficos do Segundo Teste Não Supervisionado | 98 |
| Figura 62 - Gráficos do Terceiro Teste Não Supervisionado | 99 |
| Figura 63 - Gráficos do Quarto Teste Não Supervisionado | 99 |
| Figura 64 - Resultado do IDS em Tráfego com Elevado Número de Fluxos | 100 |
| Figura 65 - Resultado do IDS em Tráfego com Ataque DDoS Presente | 101 |
| Figura 66 - Resultado do IDS em Tráfego Normal | 101 |
| Figura 67 - Resultado do IDS em Tráfego Anômalo | 102 |
| Figura 68 - Exemplificação da Estrutura e Organização das Pastas e Ficheiros (1) | 117 |
| Figura 69 - Exemplificação da Estrutura e Organização das Pastas e Ficheiros (2) | 118 |
| Figura 70 - Website do VMware | 120 |
| Figura 71 - Website do Kali Linux | 120 |
| Figura 72 - Importação do Ficheiro .vmx | 121 |
| Figura 73 - Configuração da VMnet2 | 122 |
| Figura 74 - Site do Visual Studio Code | 123 |

Lista de Siglas e Acrónimos

| | |
|-----|----------------------------|
| IDS | Intrusion Detection System |
| ML | Machine Learning |
| RN | Redes Neurais |

| | |
|-------|-------------------------------------|
| IF | Isolation Forest |
| VM | Virtual Machines |
| IAGEN | Inteligência Artificial Generativa |
| MLP | Multi-Layer Perceptron |
| RF | Random Forest |
| SVM | Support Vector Machines |
| OOB | Out-Of-Bag |
| SVC | Support Vector Classifier |
| KM | K-means Clustering |
| AE | Autoencoder |
| PCA | Principal Component Analysis |
| MSE | Mean Squared Error |
| IP | Internet Protocol |
| DHCP | Dynamic Host Configuration Protocol |
| RAM | Random Access Memory |
| CPU | Central Processing Unit |
| NAT | Networking Address Translation |
| VSC | Visual Studio Code |
| TCP | Transmission Control Protocol |
| TP | True Positives |
| FP | False Positives |
| FN | False Negatives |
| TN | True Negatives |
| DMZ | Demilitarized Zone |

1 Introdução

Com a entrada na quarta revolução industrial e o crescimento contínuo de dispositivos interligados, têm surgido novas tecnologias e uma variedade de oportunidades a explorar. No entanto, este avanço traz também novos desafios, sobretudo ao nível da segurança. O desenvolvimento de novas soluções tecnológicas vem acompanhado de ameaças mais sofisticadas e complexas.

O uso de firewalls continua a ser uma medida importante para proteger sistemas e serviços online. Contudo, a sua utilização isolada não garante um nível de segurança suficiente, sendo necessário adotar abordagens complementares.

A Detecção de Intrusões, implementada através de soluções de software ou hardware, permite identificar e prevenir comportamentos suspeitos, resultantes de tentativas de intrusão ou ataque, vindas tanto do exterior como de dentro da rede. Estes sistemas monitorizam e analisam as interações entre dispositivos, como a troca de pacotes de rede, detetando padrões invulgares. Ainda assim, à medida que os atacantes se tornam mais experientes, é necessário também evoluir os mecanismos de defesa, reforçando-os com técnicas mais adaptativas e inteligentes.

Neste projeto, propomos o desenvolvimento de um Sistema de Detecção de Intrusões (Intrusion Detection System) ([IDS](#)) com recurso a Aprendizagem Máquina (Machine Learning) ([ML](#)), com o intuito de detetar tanto ataques já conhecidos como novas formas de atividade mal-intencionada. Os modelos de [ML](#) podem aumentar a eficácia dos [IDS](#) na identificação de anomalias.

Este trabalho centra-se na construção de diferentes modelos capazes de identificar ou agrupar anomalias com base na informação recolhida em tempo real.

O dataset (CSE-CIC-IDS2018 [1]) utilizado no projeto baseia-se no tráfego de rede recolhido em ambientes reais simulados, contendo exemplos de comunicações legítimas e de diferentes formas de ataques. Este dataset inclui um conjunto alargado de atributos que descrevem as características dos fluxos de rede, como a duração das ligações, o número de pacotes enviados e recebidos, e diferentes estatísticas sobre os dados trocados. Os dados foram previamente preparados e limpos, garantindo a remoção de valores inconsistentes e a normalização necessária para os modelos de [ML](#).

Foram consideradas duas abordagens distintas: na primeira, é feita a análise do tráfego para identificar e classificar situações de intrusão ou ataque; na segunda, o tráfego é analisado com o objetivo de o separar entre normal e potencialmente malicioso, sem uma classificação detalhada.

Ambas as abordagens utilizam o mesmo conjunto de dados, mas diferem na forma como a informação é tratada e nas técnicas aplicadas. A primeira abordagem segue uma lógica supervisionada, que exige dados previamente rotulados, e onde é utilizado um modelo mais complexo, como as Redes Neurais ([RN](#)), para classificar os padrões. Na segunda abordagem, como o objetivo é apenas agrupar comportamentos semelhantes sem rótulos definidos, opta-se por um modelo Não Supervisionado, como a Isolation Forest ([IF](#)).

A infraestrutura de testes foi realizada com recurso a máquinas virtuais (Virtual Machines) ([VM](#)) interligadas numa rede privada, permitindo a simulação de cenários reais de comunicação e ataque de forma isolada e segura. O ambiente inclui componentes para a geração de tráfego, captura e análise dos fluxos, bem como para a execução dos modelos de deteção implementados no [IDS](#).

No Capítulo 2 são descritas as etapas de desenvolvimento do projeto, desde o planeamento inicial e as alterações ocorridas ao longo do processo, passando pela criação do dataset, pelo desenvolvimento dos diferentes modelos de [ML](#) e pela arquitetura final do [IDS](#), até à implementação do ambiente de testes. No Capítulo 3 é feita a avaliação do desempenho de tudo o que foi desenvolvido ao longo do projeto, analisando os resultados obtidos com base nas métricas definidas. No Capítulo 4, é feita a discussão dos resultados obtidos e apresentados no capítulo anterior. No Capítulo 5 apresentamos as conclusões finais do trabalho, refletindo sobre os objetivos atingidos e os principais pontos a destacar. Por fim, no Capítulo 6 é descrito o possível trabalho futuro, com sugestões de melhorias e continuação do projeto.

2 Desenvolvimento

O desenvolvimento do projeto teve início após a atribuição do tema, com a definição das tarefas mais adequadas, bem como a sua divisão e distribuição ao longo do semestre e pelos elementos do grupo. Para isso, iniciou-se a elaboração do ficheiro Project, demonstrado na Figura 1, no qual foi registado o planeamento inicial, contendo a sequência das tarefas, as respetivas datas de início e conclusão, e o elemento do grupo responsável por cada uma delas.

| Nome da Tarefa | Início | Conclusão | Predecessoras | % Concluída | Normes de Recursos |
|-----------------------------------------------------------------|--------------|--------------|----------------|-------------|-----------------------------------------------------|
| • 1 LP_2425 | Dom 16/03/25 | Sex 18/07/25 | | | 12% Afonso Frasilho;Guilherme Fernandes;Tomás Viana |
| • 1.1 Desenvolvimento do Ficheiro Project | Dom 16/03/25 | Qua 19/03/25 | | | 100% Tomás Viana |
| • 1.2 Pesquisa e Definição | Qui 20/03/25 | Dom 30/03/25 | | | 81% Afonso Frasilho;Guilherme Fernandes;Tomás Viana |
| 1.2.1 Pesquisa sobre IAGEN e geração de datasets sintéticos | Qui 20/03/25 | Sex 28/03/25 | | | 84% Afonso Frasilho |
| 1.2.2 Pesquisa sobre ataques cibernéticos e ambientes de testes | Qui 20/03/25 | Sex 28/03/25 | | | 84% Guilherme Fernandes |
| 1.2.3 Pesquisa sobre modelos de machine learning | Qui 20/03/25 | Sex 28/03/25 | | | 84% Tomás Viana |
| 1.2.4 Escolha das ferramentas e frameworks | Sáb 29/03/25 | Dom 30/03/25 | 4;5;6 | | 0% Afonso Frasilho;Guilherme Fernandes;Tomás Viana |
| • 1.3 Geração e Preparação do Dataset | Dom 30/03/25 | Qua 30/04/25 | 3 | | 0% Afonso Frasilho |
| 1.3.1 Definição das variáveis e categorias do dataset | Dom 30/03/25 | Qua 02/04/25 | | | 0% Afonso Frasilho |
| 1.3.2 Implementação IAGEN para geração de dados sintéticos | Qui 03/04/25 | Qua 16/04/25 | 9 | | 0% Afonso Frasilho |
| 1.3.3 Aplicação de normalização e balanceamento de dados | Qui 17/04/25 | Seg 21/04/25 | 9;10 | | 0% Afonso Frasilho |
| 1.3.4 Comparação com datasets reais disponíveis | Ter 22/04/25 | Sáb 26/04/25 | 9;10 | | 0% Afonso Frasilho |
| 1.3.5 Validação inicial dos dados e ajustes necessários | Dom 27/04/25 | Qua 30/04/25 | 9;10;11 | | 0% Afonso Frasilho |
| • 1.4 Balanço do projeto | Dom 04/05/25 | Sáb 10/05/25 | | | 0% Afonso Frasilho;Guilherme Fernandes;Tomás Viana |
| • 1.5 Análise e Seleção dos Algoritmos | Qui 01/05/25 | Sáb 31/05/25 | 7 | | 0% Tomás Viana |
| 1.5.1 Treino dos modelos | Qui 01/05/25 | Sáb 10/05/25 | 7 | | 0% Tomás Viana |
| 1.5.2 Avaliação dos modelos | Dom 11/05/25 | Sáb 17/05/25 | 16 | | 0% Tomás Viana |
| 1.5.3 Aplicação de hiperparâmetros | Dom 18/05/25 | Sáb 24/05/25 | 16 | | 0% Tomás Viana |
| 1.5.4 Seleção do melhor modelo | Dom 25/05/25 | Sáb 31/05/25 | 17 | | 0% Tomás Viana |
| • 1.6 Criação e Implementação do Ambiente de Testes | Dom 30/03/25 | Sáb 14/06/25 | 7;15 | | 0% Guilherme Fernandes |
| 1.6.1 Preparação da infraestrutura do ambiente | Dom 30/03/25 | Dom 06/04/25 | 7 | | 0% Guilherme Fernandes |
| 1.6.2 Configuração das máquinas virtuais | Seg 07/04/25 | Dom 13/04/25 | 21 | | 0% Guilherme Fernandes |
| 1.6.3 Simulação de tráfego legítimo e ataques cibernéticos | Qui 01/05/25 | Sáb 10/05/25 | 21;8;22 | | 0% Guilherme Fernandes |
| 1.6.4 Integração do modelo de Machine Learning no IDS | Dom 01/06/25 | Ter 10/06/25 | 21;15;22 | | 0% Guilherme Fernandes |
| 1.6.5 Teste do sistema e ajustes necessários | Qua 11/06/25 | Sáb 14/06/25 | 21;22;23 | | 0% Guilherme Fernandes |
| • 1.7 Relatório Final | Qui 01/05/25 | Dom 22/06/25 | 2;3;14;8;15 | | 0% Afonso Frasilho;Guilherme Fernandes;Tomás Viana |
| • 1.8 Discussão e Defesa do Projeto | Ter 01/07/25 | Sex 18/07/25 | 2;3;14;8;15;20 | | 0% Afonso Frasilho;Guilherme Fernandes;Tomás Viana |
| 1.8.1 Preparação da Discussão e Defesa do Projeto | Ter 01/07/25 | Dom 13/07/25 | | | 0% Afonso Frasilho;Guilherme Fernandes;Tomás Viana |
| 1.8.2 Discussão e Defesa do Projeto | Qui 17/07/25 | Sex 18/07/25 | 28 | | 0% Afonso Frasilho;Guilherme Fernandes;Tomás Viana |

Figura 1 - Project com o Planeamento Inicial

Fonte desta figura: Captura de Ecrã tirada do Ficheiro Project.

O projeto iniciou-se com uma pesquisa aprofundada sobre os diferentes temas e áreas de estudo aplicáveis ao tema atribuído. Foram explorados e analisados os vários tipos de datasets disponíveis (realistas e sintéticos), a forma como a Inteligência Artificial Generativa ([IAGEN](#)) pode ser aplicada e constituir uma mais-valia, o funcionamento dos diversos [IDS](#) existentes, os ataques que podem ser executados e ainda os ambientes de teste adequados ao contexto do projeto. Após a conclusão da fase de pesquisa, realizou-se um ponto de situação sobre os resultados obtidos, tendo o grupo acordado quais as ferramentas e frameworks que seriam utilizadas no desenvolvimento do projeto.

Durante o desenvolvimento, o planeamento inicial sofreu alterações significativas, principalmente devido à participação do grupo na elaboração de um artigo científico, em colaboração com os docentes da faculdade. Esta participação exigiu uma adaptação dos prazos e uma reorganização das tarefas, de modo a assegurar que os resultados obtidos estivessem prontos para serem integrados no artigo.

Estas alterações traduziram-se em ajustes nas datas de início e conclusão de diversas tarefas, na reorganização de subtarefas e na redefinição de prioridades. Algumas tarefas foram antecipadas, como a implementação dos modelos de Machine Learning, enquanto outras foram reestruturadas para garantir uma integração mais eficiente dos resultados. A tarefa relativa ao ambiente de testes foi igualmente reconfigurada, passando a incluir a integração da [IAGEN](#) para geração de tráfego sintético e simulação de cenários de rede.

O planeamento atualizado, apresentado na Figura 2, reflete esta reorganização, garantindo que o desenvolvimento do projeto segue uma lógica ajustada aos novos objetivos estabelecidos e que todas as tarefas estão devidamente organizadas.

| Nome da Tarefa | Início | Conclusão | Predecessoras | % Concluída | Nomes de Recursos |
|-----------------------------------------------------------------|--------------|--------------|----------------|-------------|--------------------------------------------------------|
| 1 LP_2425 | Dom 16/03/25 | Sex 18/07/25 | | | 52% Afonso Frاسquillo;Guilherme Fernandes;Tomás Viana |
| 1.1 Desenvolvimento do Ficheiro Project | Dom 16/03/25 | Qua 19/03/25 | | | 100% Tomás Viana |
| 1.2 Pesquisa e Definição | Qui 20/03/25 | Dom 30/03/25 | | | 100% Afonso Frاسquillo;Guilherme Fernandes;Tomás Viana |
| 1.2.1 Pesquisa sobre IAGEN e datasets | Qui 20/03/25 | Sex 28/03/25 | | | 100% Afonso Frاسquillo |
| 1.2.2 Pesquisa sobre ataques cibernéticos e ambientes de testes | Qui 20/03/25 | Sex 28/03/25 | | | 100% Guilherme Fernandes |
| 1.2.3 Pesquisa sobre modelos de machine learning | Qui 20/03/25 | Sex 28/03/25 | | | 100% Tomás Viana |
| 1.2.4 Escolha das ferramentas e frameworks | Sáb 29/03/25 | Dom 30/03/25 | 4;5;6 | | 100% Afonso Frاسquillo;Guilherme Fernandes;Tomás Viana |
| 1.3 Geração e Preparação do Dataset | Ter 01/04/25 | Dom 20/04/25 | 3 | | 100% Afonso Frاسquillo |
| 1.3.1 Definição das variáveis e categorias do dataset | Qua 02/04/25 | Seg 07/04/25 | 3 | | 100% Afonso Frاسquillo |
| 1.3.2 Limpeza e preparação do dataset | Seg 07/04/25 | Sex 11/04/25 | 9 | | 100% Afonso Frاسquillo |
| 1.3.3 Comparação com datasets reais disponíveis | Sáb 12/04/25 | Ter 15/04/25 | 9;10 | | 100% Afonso Frاسquillo |
| 1.3.4 Validação inicial dos dados e ajustes necessários | Qua 16/04/25 | Dom 20/04/25 | 9;10;11 | | 100% Afonso Frاسquillo |
| 1.4 Análise e Seleção dos Algoritmos | Seg 21/04/25 | Qua 30/04/25 | 8 | | 100% Tomás Viana |
| 1.4.1 Treino dos modelos | Seg 21/04/25 | Sex 25/04/25 | 8 | | 100% Tomás Viana |
| 1.4.2 Avaliação dos modelos | Qui 24/04/25 | Sex 25/04/25 | 14 | | 100% Tomás Viana |
| 1.4.3 Ajustes nos Modelos | Seg 28/04/25 | Ter 29/04/25 | 14;15 | | 100% Tomás Viana |
| 1.4.4 Seleção dos melhores modelos | Qua 30/04/25 | Qua 30/04/25 | 15 | | 100% Tomás Viana |
| 1.5 Balanço do projeto | Qui 08/05/25 | Sáb 10/05/25 | | | 0% Tomás Viana;Afonso Frاسquillo;Guilherme Fernandes |
| 1.6 Criação e Implementação do Ambiente de Testes | Seg 31/03/25 | Sex 23/05/25 | 3;8;13 | | 23% Guilherme Fernandes |
| 1.6.1 Preparação da infraestrutura do ambiente | Seg 31/03/25 | Ter 01/04/25 | | | 100% Guilherme Fernandes |
| 1.6.2 Configuração das máquinas virtuais | Qua 02/04/25 | Sex 04/04/25 | | | 100% Guilherme Fernandes |
| 1.6.3 Implementação IAGEN para geração de dados sintéticos | Sex 09/05/25 | Ter 13/05/25 | 8;13 | | 0% Afonso Frاسquillo |
| 1.6.4 Integração do modelo de Machine Learning no IDS | Qua 14/05/25 | Qui 15/05/25 | 13;20;21 | | 0% Guilherme Fernandes |
| 1.6.5 Simulação de tráfego legítimo e ataques cibernéticos | Sex 16/05/25 | Sáb 17/05/25 | 20;21;22;23 | | 0% Guilherme Fernandes |
| 1.6.6 Teste do sistema e ajustes necessários | Seg 19/05/25 | Sex 23/05/25 | 20;21;22;23;24 | | 0% Guilherme Fernandes |
| 1.7 Relatório Final | Ter 01/04/25 | Seg 30/06/25 | | | 41% Afonso Frاسquillo;Guilherme Fernandes;Tomás Viana |
| 1.8 Discussão e Defesa do Projeto | Ter 01/07/25 | Sex 18/07/25 | | | 0% Guilherme Fernandes;Tomás Viana;Afonso Frاسquillo |
| 1.8.1 Preparação da Discussão e Defesa do Projeto | Ter 01/07/25 | Sex 07/07/25 | | | 0% Afonso Frاسquillo;Guilherme Fernandes;Tomás Viana |
| 1.8.2 Discussão e Defesa do Projeto | Qui 17/07/25 | Sex 18/07/25 | | | 0% Tomás Viana;Afonso Frاسquillo;Guilherme Fernandes |

Figura 2 - Project com Alterações no Planeamento

Fonte desta figura: Captura de Ecrã tirada do Ficheiro Project.

Na fase de implementação do ambiente de testes com o [IDS](#), foi novamente necessário proceder a ajustes no planeamento no ficheiro Project, conforme demonstrado na Figura 3. Neste contexto, algumas datas foram reajustadas e a tarefa referente à implementação da [IAGEN](#) para geração de dados sintéticos foi removida. Esta decisão deveu-se à ausência de necessidade em adicionar novos tipos de ataques ao conjunto de dados previamente utilizado, uma vez que este já continha um número considerável e diversificado de ataques. Além disso, não houve necessidade de recorrer à geração sintética para as previsões práticas dos modelos, pois as ferramentas disponíveis no ambiente de testes já permitiam executar diversos ataques, incluindo ataques novos e desconhecidos pelos modelos.

| Nome da Tarefa | Início | Conclusão | Predecessoras | % Concluída | Nomes de Recursos |
|-----------------------------------------------------------------|--------------|--------------|---------------|-------------|--------------------------------------------------------|
| 1 LP_2425 | Dom 16/03/25 | Sex 18/07/25 | | | 96% Afonso Frاسquillo;Guilherme Fernandes;Tomás Viana |
| 1.1 Desenvolvimento do Ficheiro Project | Dom 16/03/25 | Qua 19/03/25 | | | 100% Tomás Viana |
| 1.2 Pesquisa e Definição | Qui 20/03/25 | Dom 30/03/25 | | | 100% Afonso Frاسquillo;Guilherme Fernandes;Tomás Viana |
| 1.2.1 Pesquisa sobre IAGEN e datasets | Qui 20/03/25 | Sex 28/03/25 | | | 100% Afonso Frاسquillo |
| 1.2.2 Pesquisa sobre ataques cibernéticos e ambientes de testes | Qui 20/03/25 | Sex 28/03/25 | | | 100% Guilherme Fernandes |
| 1.2.3 Pesquisa sobre modelos de machine learning | Qui 20/03/25 | Sex 28/03/25 | | | 100% Tomás Viana |
| 1.2.4 Escolha das ferramentas e frameworks | Sáb 29/03/25 | Dom 30/03/25 | 4;5;6 | | 100% Afonso Frاسquillo;Guilherme Fernandes;Tomás Viana |
| 1.3 Geração e Preparação do Dataset | Ter 01/04/25 | Sex 18/04/25 | 3 | | 100% Afonso Frاسquillo |
| 1.3.1 Definição das variáveis e categorias do dataset | Ter 01/04/25 | Seg 07/04/25 | 3 | | 100% Afonso Frاسquillo |
| 1.3.2 Limpeza e preparação do dataset | Ter 08/04/25 | Sex 11/04/25 | 9 | | 100% Afonso Frاسquillo |
| 1.3.3 Validação inicial dos dados e ajustes necessários | Qua 16/04/25 | Sex 18/04/25 | 9;10 | | 100% Afonso Frاسquillo |
| 1.4 Análise e Seleção dos Algoritmos | Seg 21/04/25 | Qua 30/04/25 | 8 | | 100% Tomás Viana |
| 1.4.1 Treino dos modelos | Seg 21/04/25 | Sex 25/04/25 | 8 | | 100% Tomás Viana |
| 1.4.2 Avaliação dos modelos | Sáb 26/04/25 | Dom 27/04/25 | 13 | | 100% Tomás Viana |
| 1.4.3 Ajustes nos Modelos | Seg 28/04/25 | Seg 23/06/25 | 13;14 | | 100% Tomás Viana |
| 1.4.4 Seleção dos melhores modelos | Qua 30/04/25 | Qua 30/04/25 | 14 | | 100% Tomás Viana |
| 1.5 Balanço do projeto | Qui 08/05/25 | Sáb 10/05/25 | | | 100% Tomás Viana;Afonso Frاسquillo;Guilherme Fernandes |
| 1.6 Criação e Implementação do Ambiente de Testes | Seg 31/03/25 | Sex 23/05/25 | 3;8;12 | | 100% Guilherme Fernandes |
| 1.6.1 Preparação da infraestrutura do ambiente | Seg 31/03/25 | Ter 01/04/25 | | | 100% Guilherme Fernandes |
| 1.6.2 Configuração das máquinas virtuais | Qua 02/04/25 | Sex 04/04/25 | | | 100% Guilherme Fernandes |
| 1.6.3 Integração do modelo de Machine Learning no IDS | Seg 16/06/25 | Ter 24/06/25 | 12;19;20 | | 100% Guilherme Fernandes;Tomás Viana |
| 1.6.4 Simulação de tráfego legítimo e ataques cibernéticos | Qua 25/06/25 | Qua 25/06/25 | 19;20;21 | | 100% Guilherme Fernandes |
| 1.6.5 Teste do sistema e ajustes necessários | Qua 26/03/25 | Qua 26/03/25 | 19;20;21;22 | | 100% Guilherme Fernandes |
| 1.7 Relatório Final | Ter 01/04/25 | Seg 30/06/25 | | | 97% Afonso Frاسquillo;Guilherme Fernandes;Tomás Viana |
| 1.8 Discussão e Defesa do Projeto | Ter 01/07/25 | Ter 08/07/25 | | | 0% Guilherme Fernandes;Tomás Viana;Afonso Frاسquillo |
| 1.8.1 Preparação da Discussão e Defesa do Projeto | Ter 01/07/25 | Seg 07/07/25 | | | 0% Afonso Frاسquillo;Guilherme Fernandes;Tomás Viana |
| 1.8.2 Discussão e Defesa do Projeto | Ter 08/07/25 | Ter 08/07/25 | 26 | | 0% Tomás Viana;Afonso Frاسquillo;Guilherme Fernandes |

Figura 3 - Versão Final do Project

Fonte desta figura: Captura de Ecrã tirada do Ficheiro Project.

Até à data prevista para a conclusão do projeto, não se verificou a necessidade de efetuar novas alterações ou ajustes ao planeamento, pelo que este permaneceu inalterado.

2.1 Conjunto de Dados

Nesta secção, é apresentada a seleção e preparação do conjunto de dados utilizado no desenvolvimento dos modelos de [ML](#) aplicados ao [IDS](#). Não foi desenvolvido um conjunto de dados próprio; em vez disso, optou-se pela utilização do CSE-CIC-IDS2018 [1], um dataset de referência na área, escolhido pelas suas características adequadas ao contexto do projeto.

A escolha deste dataset deveu-se ao facto de oferecer tráfego de rede realista, recolhido em ambiente controlado, incluindo uma ampla variedade de ataques e comunicações benignas devidamente rotuladas, e por já se encontrar, em grande parte, limpo e preparado, restando apenas algumas personalizações para o tornar mais aplicável ao nosso projeto. Esta combinação permite treinar e avaliar modelos de deteção de intrusões em cenários próximos da realidade, com dados consistentes e representativos.

A seguir, são descritos os processos aplicados para a preparação do dataset, incluindo a junção dos diferentes ficheiros num único, os ajustes realizados para uma aplicação mais apropriada ao projeto e a respetiva análise exploratória.

2.1.1 Preparação do Dataset

Para a preparação do dataset, foi realizada uma junção e limpeza dos diversos ficheiros do dataset original CSE-CIC-IDS2018, bem como a normalização dos rótulos e uma amostragem balanceada das classes.

Inicialmente, foram selecionados os ficheiros relativos aos dias 14, 15, 21, 22 e 23 de fevereiro e 2 de março de 2018. Estes ficheiros foram carregados com tipos de dados otimizados por coluna, definidos para reduzir o consumo de memória e garantir uma leitura eficiente. Apenas as colunas relevantes foram consideradas no carregamento.

Após a junção dos diferentes ficheiros num único dataframe, procedeu-se à substituição de eventuais valores infinitos e à eliminação de linhas com valores nulos, assegurando a integridade dos dados.

Seguidamente, os rótulos dos ataques foram generalizados para reduzir a complexidade das classes e facilitar o treino dos modelos. Esta generalização agrupou, por exemplo, os

ataques FTP-BruteForce, SSH-Bruteforce, Brute Force -Web e Brute Force -XSS na classe BruteForce, e os ataques DDoS attack-HOIC e DDoS attack-LOIC-UDP na classe DDoS.

Os rótulos foram então codificados numericamente, com o tráfego benigno representado pelo valor 1 e os diferentes tipos de ataques por valores entre 2 e 6.

Por fim, foi realizada uma amostragem balanceada para a criação do ficheiro final. Selecionaram-se 100.000 amostras de tráfego benigno e até 10.000 amostras de cada tipo de ataque, de modo a obter um dataset equilibrado e adequado ao treino dos modelos.

O conjunto resultante foi exportado para um ficheiro `cleaned_ids2018.csv`, pronto a ser utilizado no desenvolvimento e avaliação do [IDS](#).

2.1.2 Análise Exploratória

Foi realizada uma análise exploratória ao ficheiro final `cleaned_ids2018.csv` com o objetivo de caracterizar o conjunto de dados e identificar propriedades relevantes para o treino e avaliação dos modelos de [ML](#).

Inicialmente, verificou-se a dimensão do dataset e a sua estrutura, confirmando a inexistência de valores nulos nas colunas consideradas. Os tipos de dados atribuídos a cada atributo foram revistos de forma a garantir a consistência e a otimização da utilização de memória.

Procedeu-se ao cálculo das estatísticas descritivas dos atributos numéricos, incluindo média, desvio padrão, valores mínimos e máximos, quartis, mediana, assimetria (skewness) e curtose (kurtosis). Estes indicadores permitiram obter uma visão detalhada da distribuição dos dados e da presença de possíveis desvios ou outliers. Os resultados foram guardados no ficheiro `estatisticas_descritivas.csv` para futura referência.

Analisou-se ainda a distribuição das classes no dataset, através de um gráfico de barras que apresenta o número de registos por classe e as respetivas percentagens. Esta análise confirmou o balanceamento aplicado na fase de preparação dos dados, sendo visível uma distribuição proporcional entre tráfego benigno e as diferentes categorias de ataque. O gráfico gerado foi exportado como `distribuicao_classes.png`.

Por fim, foi calculada a matriz de correlação entre os 15 atributos com maior variância, por forma a identificar relações lineares mais significativas no conjunto de dados. Esta matriz foi representada graficamente sob a forma de heatmap, facilitando a interpretação visual das correlações mais fortes ou mais fracas entre atributos. O resultado foi exportado como `matriz_correlacao.png`.

Esta análise permitiu reforçar o conhecimento sobre o dataset e identificar aspetos importantes para a configuração dos modelos de deteção de intrusões.

2.2 Modelos de Machine Learning

Nesta secção, são apresentados os seis modelos de [ML](#) desenvolvidos para o [IDS](#). Foram consideradas abordagens supervisionadas e não supervisionadas, permitindo explorar diferentes estratégias para a deteção de comportamentos maliciosos.

Os modelos supervisionados foram treinados com base em dados rotulados, onde cada amostra contém a indicação da sua classe (tráfego benigna ou algum tipo de ataque). Estes modelos foram configurados para aprender a partir dos padrões presentes nos dados e, posteriormente, aplicá-los para classificar novas amostras de forma automática.

Por outro lado, os modelos não supervisionados foram configurados para analisar o tráfego sem qualquer conhecimento prévio das classes, identificando padrões e anomalias com base nas características dos dados. Esta abordagem é particularmente útil para a deteção de novos tipos de ataques ou comportamentos desconhecidos.

A seguir, são descritos em detalhe os modelos utilizados, incluindo a sua configuração, arquitetura e justificações para as escolhas realizadas.

2.2.1 Modelos Supervisionados Desenvolvidos

Os modelos supervisionados utilizados neste projeto foram selecionados com base na sua eficácia em problemas de classificação. Estes modelos foram configurados para aprender padrões a partir dos dados rotulados, permitindo distinguir entre tráfego benigno e tráfego com algum tipo de ataque.

Foram implementados os seguintes modelos supervisionados:

- Rede Neuronal Multi-Layer Perceptron ([MLP](#)): Um modelo flexível e poderoso, capaz de capturar padrões complexos nos dados de tráfego de rede.
- Árvore de Decisão Random Forest ([RF](#)): Uma abordagem robusta e eficiente, que combina múltiplas árvores de decisão para obter previsões estáveis.
- Máquina de Vetores de Suporte ([SVM](#)): Um modelo eficaz para classificação, especialmente em contextos onde os dados não são linearmente separáveis.

Nos subcapítulos seguintes, são descritos detalhadamente cada um destes modelos, incluindo os diversos testes realizados até à seleção do modelo mais adequado, bem como o seu pré-processamento, arquitetura e configuração final.

2.2.1.1 Rede Neuronal Multi-Layer Perceptron

Iniciou-se o desenvolvimento dos modelos supervisionados com uma Rede Neuronal do tipo [MLP](#). Para tal, foram criados quatro ficheiros de teste, sendo que no primeiro ficheiro foi colocada uma versão baseline do modelo, enquanto no quarto ficheiro foi implementada aquela que se considerou ser a versão mais complexa e trabalhada. O objetivo destes testes foi determinar, entre as diferentes configurações possíveis do modelo, qual a melhor opção a utilizar como modelo final.

2.2.1.1.1 Primeiro Teste

No primeiro teste realizado, implementou-se uma versão simples e inicial do modelo, que serviu como baseline para comparação com versões posteriores. Todo o processo foi desenvolvido em Python, recorrendo-se a bibliotecas da `scikit-learn`, `pandas` e `numpy`. A biblioteca `pandas` foi utilizada para o carregamento e manipulação do conjunto de dados (`cleaned_ids2018.csv`), enquanto `numpy` foi usada para o tratamento eficiente de arrays numéricos. A biblioteca `scikit-learn` foi fundamental em todo o processo, fornecendo ferramentas para divisão dos dados (`train_test_split`), normalização (`StandardScaler`), construção e treino do modelo (`MLPClassifier`) e avaliação dos resultados (`classification_report` e `confusion_matrix`).

Após carregar o conjunto de dados, efetuou-se o pré-processamento dos mesmos, removendo-se a coluna das categorias de tráfego (“Label”) e aplicando-se a normalização com `StandardScaler`, de forma a garantir que todas as variáveis tivessem a mesma escala. Em seguida, os dados foram divididos em conjuntos de treino e teste, na proporção de 70% para treino e 30% para teste, com estratificação pelas classes para preservar a distribuição original das categorias.

A configuração do modelo [MLP](#) neste teste foi deliberadamente simplificada, contendo uma única camada oculta com 10 neurónios, função de ativação ReLU, algoritmo de otimização Adam e um limite máximo de 100 iterações. O modelo foi treinado com o conjunto de treino e, posteriormente, avaliado com o conjunto de teste, recorrendo-se ao relatório de classificação (`classification report`) e à matriz de confusão como métricas de desempenho.

Este teste inicial permitiu obter uma referência de base para a avaliação do desempenho do modelo e serviu como ponto de partida para as otimizações implementadas nos testes seguintes.

2.2.1.1.2 Segundo Teste

No segundo teste efetuado, introduziram-se melhorias na arquitetura e configuração do modelo relativamente à versão inicial (baseline). Tal como no teste anterior, recorreu-se às bibliotecas pandas, numpy e scikit-learn para o carregamento, pré-processamento, construção e avaliação do modelo. O ficheiro `cleaned_ids2018.csv` foi novamente utilizado como fonte de dados, mantendo-se o processo de normalização com `StandardScaler` e a divisão estratificada em conjuntos de treino (70%) e teste (30%).

Neste teste, a configuração do modelo foi significativamente aprimorada: passou a conter duas camadas ocultas, com 100 e 50 neurónios, respetivamente, mantendo-se a função de ativação ReLU e o otimizador Adam. Foram ainda aplicadas técnicas de regularização L2, com $\alpha=1e-4$, e aprendizagem com taxa adaptativa (`learning_rate='adaptive'`). Para prevenir overfitting, implementou-se paragem antecipada (early stopping), reservando-se 10% dos dados de treino para validação e interrompendo-se o processo após 10 iterações sem melhoria. O número máximo de iterações foi aumentado para 200.

A avaliação do modelo foi realizada, tal como anteriormente, através do relatório de classificação e da matriz de confusão. Este segundo teste permitiu analisar o impacto das melhorias introduzidas na arquitetura e configuração do modelo, comparando os resultados com os obtidos na versão inicial.

2.2.1.1.3 Terceiro Teste

No terceiro teste, continuaram a ser introduzidas melhorias progressivas na arquitetura do modelo [MLP](#), com o objetivo de aumentar a sua capacidade de generalização e refinar o desempenho obtido nos testes anteriores. Tal como nas fases anteriores, recorreu-se às bibliotecas pandas, numpy e scikit-learn para as operações de carregamento, normalização, treino e avaliação.

Neste teste, a arquitetura da rede foi expandida para três camadas ocultas, compostas por 150, 100 e 50 neurónios, respetivamente. Optou-se por alterar a função de ativação para tanh, em substituição da ReLU utilizada nos testes anteriores. Manteve-se o otimizador Adam e a aprendizagem adaptativa, com `learning_rate_init=1e-3`.

A regularização L2 foi novamente aplicada, agora com um valor inferior ($\alpha=1e-5$), e a paragem antecipada (early stopping) foi mantida, com uma fração de validação de 15%. O número máximo de iterações foi aumentado para 300, com uma tolerância definida de $1e-4$ e interrupção após 10 iterações sem melhoria.

Tal como nos testes anteriores, a avaliação do modelo foi realizada através do relatório de classificação (classification report) e da matriz de confusão, permitindo analisar o impacto destas novas configurações na performance do modelo e aproximar-se da versão final pretendida.

2.2.1.1.4 Quarto Teste

No quarto e último teste da série dedicada ao modelo [MLP](#), foi implementada aquela que se considerou ser a versão mais completa do modelo, incorporando várias melhorias tanto a nível estrutural como de configuração. Tal como nos testes anteriores, recorreram-se às bibliotecas pandas, numpy e scikit-learn para o carregamento, tratamento e avaliação dos dados.

Neste último teste, a rede neuronal foi construída com três camadas ocultas de maior profundidade, compostas por 256, 128 e 64 neurónios, respetivamente, com a função de ativação ReLU. Manteve-se o otimizador Adam, a aprendizagem com taxa adaptativa (learning_rate='adaptive') e uma taxa inicial de 1e-3. O treino foi efetuado em lotes (batch_size=64), o que permitiu melhorar a eficiência computacional e a estabilidade do processo de otimização.

A regularização L2 foi aplicada com alpha=1e-5, e a estratégia de paragem antecipada (early stopping) foi mantida, agora com n_iter_no_change=15 e tol=1e-5, refletindo uma maior exigência de estabilidade antes de interromper o treino. A fração de validação foi definida em 10%. Para melhorar ainda mais a convergência, foi introduzido o uso de momentum (momentum=0.9) e Nesterov momentum (nesterovs_momentum=True). O número máximo de iterações foi aumentado para 500, permitindo maior flexibilidade ao processo de aprendizagem.

A avaliação foi novamente conduzida com o relatório de classificação e a matriz de confusão, permitindo analisar o desempenho desta versão mais avançada do modelo e compará-la com os testes anteriores, com vista à seleção final da melhor configuração.

2.2.1.1.5 Modelo Final

O modelo final selecionado baseia-se na configuração utilizada no Segundo Teste, por ter evidenciado os melhores resultados, resultados esses detalhados e discutidos em secções posteriores deste relatório.

O desenvolvimento do modelo recorreu às bibliotecas pandas, numpy, scikit-learn, matplotlib, seaborn e joblib. Foi utilizado o ficheiro cleaned_ids2018.csv, cujos dados foram inicialmente preparados através da separação entre as variáveis independentes (X) e a variável

alvo (y). Posteriormente, os dados foram normalizados com StandardScaler, garantindo a uniformização da escala dos atributos. A divisão entre treino e teste foi efetuada com uma proporção de 70%-30%, aplicando estratificação pelas classes.

O classificador [MLP](#) implementado possui uma arquitetura composta por duas camadas ocultas com 100 e 50 neurónios, respetivamente, utilizando a função de ativação ReLU. O otimizador escolhido foi o algoritmo Adam, com regularização L2 ($\alpha=1e-4$) e taxa de aprendizagem adaptativa (`learning_rate='adaptive'`), com valor inicial de 0.001. O treino foi sujeito a paragem antecipada (early stopping), reservando 10% dos dados de treino para validação, sendo interrompido após 10 iterações sem melhoria. O número máximo de iterações foi definido como 200.

Após o treino, o modelo foi avaliado sobre o conjunto de teste, sendo calculadas as métricas precision, recall e f1-score com média ponderada. A matriz de confusão correspondente foi gerada e visualizada. Para complementar a avaliação, realizou-se validação cruzada com 5 folds para as três métricas referidas, permitindo analisar a sua média e desvio padrão. Adicionalmente, foi feita validação cruzada com 10 folds exclusivamente para o f1-score, cujos resultados foram representados graficamente e exportados para ficheiro .csv, de modo a observar a variação do desempenho ao longo dos diferentes subconjuntos.

Por fim, tanto o modelo final como o scaler utilizado na normalização foram guardados com joblib, permitindo a sua reutilização em ambiente real de teste.

2.2.1.2 Árvore de Decisão Random Forest

Prosseguiu-se com o desenvolvimento do segundo modelo supervisionado, recorrendo à implementação de uma Árvore de Decisão do tipo Random Forest. Tal como no modelo anterior, foram criados quatro ficheiros de teste, iniciando-se com uma versão baseline simples no primeiro teste e evoluindo para uma versão mais completa no quarto teste. O objetivo destes testes foi avaliar o impacto das diferentes configurações e identificar a versão do modelo com melhor desempenho, a ser utilizada como modelo final.

2.2.1.2.1 Primeiro Teste

No primeiro teste do modelo Random Forest, foi implementada uma versão simples com configurações padrão, com o objetivo de servir como baseline para comparação com versões mais desenvolvidas nos testes seguintes. Tal como nas experiências anteriores, utilizaram-se as

bibliotecas pandas, numpy e scikit-learn para o carregamento e preparação dos dados, construção do modelo e avaliação do seu desempenho.

O conjunto de dados (cleaned_ids2018.csv) foi carregado, tendo sido separadas as variáveis independentes (X) da variável alvo (y). Aplicou-se a normalização com StandardScaler e os dados foram divididos em treino (70%) e teste (30%) com estratificação pela classe.

A configuração do modelo nesta fase baseou-se numa floresta com 100 árvores (n_estimators=100), sem limitação de profundidade (max_depth=None) e com execução paralela ativada (n_jobs=-1) para aproveitar todos os núcleos disponíveis. Após o treino com os dados de treino, o modelo foi avaliado sobre os dados de teste através do relatório de classificação (classification report) e da matriz de confusão.

2.2.1.2.2 Segundo Teste

No segundo teste do modelo Random Forest, introduziram-se melhorias significativas relativamente à versão baseline, com o objetivo de aumentar a capacidade de generalização e controlo sobre o sobreajustamento (overfitting). O pré-processamento de dados manteve-se inalterado, utilizando pandas e numpy para o carregamento e manipulação do dataset cleaned_ids2018.csv, StandardScaler para normalização e train_test_split para a divisão estratificada dos dados em treino e teste.

Nesta configuração, aumentou-se o número de árvores para 300 (n_estimators=300) e impôs-se uma profundidade máxima de 20 níveis por árvore (max_depth=20). Para reduzir a variância e aumentar a estabilidade do modelo, foram definidos parâmetros adicionais como min_samples_split=5 e min_samples_leaf=2, garantindo que cada divisão e cada folha contenham um número mínimo de amostras. O número de atributos considerados em cada divisão foi ajustado para a raiz quadrada do total (max_features='sqrt'), estratégia comum para Random Forests.

Ativou-se a opção de bootstrap (bootstrap=True) e ponderaram-se automaticamente as classes com class_weight='balanced', de modo a lidar com possíveis desequilíbrios no conjunto de dados. Adicionalmente, foi ativada a estimativa out-of-bag ([OOB](#)) (oob_score=True), permitindo avaliar o desempenho do modelo sem recorrer ao conjunto de teste. Após o treino, o modelo foi avaliado com as métricas habituais (relatório de classificação e matriz de confusão), e o [OOB](#)-score foi igualmente apresentado, fornecendo uma estimativa adicional da performance do modelo sobre dados não vistos.

2.2.1.2.3 Terceiro Teste

No terceiro teste, aprofundaram-se as otimizações do modelo Random Forest com o intuito de explorar o seu desempenho em cenários mais complexos. O pré-processamento de dados manteve-se idêntico aos testes anteriores, com normalização através de StandardScaler e divisão estratificada em conjuntos de treino e teste.

Nesta versão, aumentou-se significativamente o número de árvores da floresta para 1000 (`n_estimators=1000`), permitindo uma agregação mais robusta das decisões. A profundidade máxima foi fixada em 40 (`max_depth=40`), mantendo-se as restrições `min_samples_split=5` e `min_samples_leaf=2`, para garantir equilíbrio entre precisão e controlo de sobreajustamento. O parâmetro `max_features` foi definido como 0.3, restringindo a seleção de atributos a 30% do total em cada divisão, incentivando a diversidade entre as árvores.

Ao contrário dos testes anteriores, foi utilizada a métrica de entropia como critério de divisão (`criterion='entropy'`), em vez da habitual gini, com o objetivo de testar o impacto de uma medida alternativa na divisão dos nós. O peso das classes foi ajustado para `balanced_subsample`, assegurando ponderações proporcionais ao subconjunto de dados de cada árvore. Manteve-se a ativação do bootstrap e da estimativa out-of-bag (`oob_score=True`), e o parâmetro `warm_start` foi explicitamente desativado para garantir treino independente em cada execução.

Após o treino, o modelo foi avaliado com o conjunto de teste, tal como nas iterações anteriores, através do relatório de classificação, da matriz de confusão e da apresentação do [OOB](#)-score. Esta versão permitiu estudar o impacto do aumento de complexidade e da alteração do critério de divisão no desempenho global do modelo.

2.2.1.2.4 Quarto Teste

No quarto e último teste do modelo Random Forest, foi implementada a versão com a configuração mais complexa entre todas as testadas, incorporando um elevado número de árvores e diversos parâmetros ajustados com o objetivo de explorar o impacto de uma maior complexidade estrutural no comportamento do modelo. O pré-processamento dos dados manteve-se consistente com os testes anteriores.

Nesta versão, o número de árvores foi elevado para 1500 (`n_estimators=1500`), reforçando a estabilidade estatística das previsões por agregação. Optou-se por não limitar a profundidade das árvores (`max_depth=None`), permitindo que estas crescessem até à separação

completa dos dados. As condições mínimas para divisão e folhas foram definidas como `min_samples_split=2` e `min_samples_leaf=1`, respetivamente, possibilitando a máxima expressividade de cada árvore.

O parâmetro `max_features` foi ajustado para 50% dos atributos disponíveis, promovendo diversidade nas divisões internas. Mantiveram-se o bootstrap e a ponderação das classes com `balanced_subsample`, garantindo a adaptação do modelo a distribuições de classes potencialmente desequilibradas. Introduziu-se também o parâmetro `min_impurity_decrease=1e-5`, forçando uma divisão apenas quando esta resulta numa diminuição mínima da impureza, medida por Gini (por omissão).

Como nos testes anteriores, ativou-se o cálculo do out-of-bag score (`oob_score=True`) para obter uma estimativa de desempenho sem recorrer ao conjunto de teste. A avaliação incluiu o relatório de classificação, a matriz de confusão e o [OOB-score](#), permitindo uma análise abrangente da eficácia deste modelo em comparação com as versões anteriores.

2.2.1.2.5 Modelo Final

O modelo final selecionado para a componente supervisionada com [RF](#) corresponde à configuração utilizada no Quarto Teste, por ter apresentado os melhores resultados. A análise detalhada dos seus desempenhos será abordada numa secção posterior deste relatório.

Tal como nas fases anteriores, o desenvolvimento foi realizado em Python, utilizando as bibliotecas `pandas`, `numpy`, `scikit-learn`, `matplotlib`, `seaborn` e `jolib`. O ficheiro `cleaned_ids2018.csv` foi carregado, sendo os atributos separados da variável alvo. A normalização dos dados foi efetuada com `StandardScaler`, seguida da divisão em conjuntos de treino e teste na proporção de 70%-30%, com estratificação pela classe.

O modelo final foi configurado com 1500 árvores (`n_estimators=1500`), sem limite de profundidade (`max_depth=None`). Foram definidos os seguintes parâmetros adicionais: `min_samples_split=2`, `min_samples_leaf=1`, `max_features=0.5` e `min_impurity_decrease=1e-5`. O modelo foi treinado com `bootstrap=True`, ponderação das classes por amostragem (`class_weight='balanced_subsample'`) e cálculo do [OOB-score](#) (`oob_score=True`), utilizado como métrica interna de avaliação sem necessidade de recorrer ao conjunto de teste.

Após o treino, o modelo foi avaliado com base no conjunto de teste, sendo registadas as métricas `precision`, `recall`, `f1-score` e `accuracy`, bem como o [OOB-score](#). A matriz de confusão correspondente foi gerada e representada graficamente com `seaborn`, permitindo visualizar os acertos e erros por classe.

Tal como no modelo [MLP](#), foi realizada validação cruzada em dois momentos distintos: primeiro, com 5 folds, avaliando as métricas precision, recall e f1-score com média ponderada; depois, uma validação cruzada com 10 folds dedicada à métrica f1-score, cujos valores foram representados graficamente e exportados em formato .csv, possibilitando a análise da estabilidade do modelo ao longo dos diferentes subconjuntos.

Por fim, tanto o modelo treinado como o scaler utilizado na normalização foram guardados com a biblioteca joblib, permitindo a sua reutilização futura em ambiente real de teste.

2.2.1.3 Máquina de Vetores de Suporte

O terceiro modelo supervisionado desenvolvido foi uma [SVM](#). Tal como nos modelos anteriores, foram realizados quatro testes distintos, com o objetivo de experimentar diferentes configurações e identificar aquela que apresentasse melhor desempenho. O primeiro teste correspondeu a uma versão baseline, enquanto o quarto representou a configuração mais completa e ajustada. Estes testes permitiram explorar o impacto dos principais parâmetros da [SVM](#) na classificação do tráfego.

2.2.1.3.1 Primeiro Teste

No primeiro teste, foi utilizada a configuração padrão da classe Support Vector Classifier ([SVC](#)) da biblioteca scikit-learn, servindo como versão baseline para comparação com configurações mais específicas nos testes seguintes. O pré-processamento foi realizado de forma idêntica aos modelos anteriores, com recurso às bibliotecas pandas e numpy para o carregamento e manipulação do conjunto de dados cleaned_ids2018.csv, e StandardScaler para a normalização das variáveis preditoras.

O conjunto de dados foi dividido em treino (70%) e teste (30%) com estratificação, garantindo a proporcionalidade das classes. O modelo [SVM](#) foi treinado com os parâmetros padrões, sem qualquer ajuste adicional.

Após o treino, o modelo foi avaliado com base nas métricas de precision, recall e f1-score com média ponderada, bem como na matriz de confusão. Este teste teve como objetivo estabelecer uma referência de desempenho inicial para o modelo [SVM](#).

2.2.1.3.2 Segundo Teste

No segundo teste do modelo [SVM](#), introduziram-se ligeiros ajustes à configuração utilizada no teste anterior, mantendo-se o kernel radial ('rbf') como função de transformação dos dados, mas especificando explicitamente os parâmetros $C=1.0$ e $\gamma='scale'$, que, embora assemelhem-se com os valores por omissão, permitiram consolidar a base para comparações futuras. Foi também incluído $random_state=42$ para garantir a reprodutibilidade do resultado.

O processo de preparação dos dados manteve-se idêntico: carregamento e separação das variáveis, normalização com StandardScaler, e divisão estratificada dos dados em treino e teste. O modelo foi treinado com esta configuração e avaliado com base no relatório de classificação e na matriz de confusão, tal como nos testes anteriores.

2.2.1.3.3 Terceiro Teste

No terceiro teste, foram realizados os primeiros ajustes significativos aos hiperparâmetros do modelo [SVM](#), com o objetivo de avaliar o impacto da complexidade do modelo no seu desempenho. Foi mantido o kernel radial ('rbf'), mas o parâmetro de regularização C foi aumentado para 5.0, permitindo ao modelo ajustar-se mais aos dados de treino. Simultaneamente, o parâmetro γ foi definido com o valor 0.01, limitando a influência de cada ponto de dados e controlando a curvatura da fronteira de decisão.

O conjunto de dados foi novamente carregado e preparado utilizando o mesmo processo dos testes anteriores: normalização com StandardScaler e divisão estratificada em conjuntos de treino (70%) e teste (30%).

Após o treino, o modelo foi avaliado com base no relatório de classificação e na matriz de confusão, permitindo observar a influência destas alterações nos resultados obtidos.

2.2.1.3.4 Quarto Teste

No quarto e último teste do modelo [SVM](#), foi implementada a configuração mais ajustada entre todas as testadas, com o objetivo de maximizar o desempenho do modelo face ao problema em estudo. Manteve-se o kernel radial ('rbf'), tendo sido aumentada a penalização de erro com $C=10.0$, tornando o modelo mais sensível a classificações incorretas. O parâmetro γ foi reduzido para 0.005, permitindo uma fronteira de decisão mais suave. Adicionalmente, foi introduzido $class_weight='balanced'$, de modo a compensar automaticamente eventuais desequilíbrios entre as classes no conjunto de dados.

O processo de preparação dos dados foi idêntico aos restantes testes: carregamento do ficheiro `cleaned_ids2018.csv`, normalização com `StandardScaler` e divisão estratificada dos dados em treino e teste.

Após o treino, o modelo foi avaliado com base no relatório de classificação e na matriz de confusão, possibilitando a análise do impacto destas afinações finais no comportamento da [SVM](#). Este teste representou a versão mais completa da abordagem com [SVM](#), servindo de referência para comparação com as configurações anteriores.

2.2.1.3.5 Modelo Final

O modelo final selecionado para a abordagem com [SVM](#) corresponde à configuração utilizada no Terceiro Teste, por ter apresentado os melhores resultados, os quais são analisados numa secção posterior deste relatório.

O desenvolvimento foi realizado em Python, utilizando as bibliotecas `pandas`, `numpy`, `scikit-learn`, `matplotlib`, `seaborn` e `jolib`. O conjunto de dados `cleaned_ids2018.csv` foi carregado e preparado, com separação das variáveis preditivas e da variável alvo, seguido da normalização dos dados com `StandardScaler`. A divisão em conjuntos de treino (70%) e teste (30%) foi realizada com estratificação.

O modelo [SVM](#) foi treinado com parâmetros ajustados da classe [SVC](#) da biblioteca `scikit-learn`, recorrendo ao kernel radial (`'rbf'`), com $C=5.0$ e $\gamma=0.01$. Esta configuração foi suficiente para obter um desempenho competitivo no contexto do problema em estudo.

Após o treino, o modelo foi avaliado sobre o conjunto de teste, sendo os resultados registados em ficheiro, incluindo o relatório de classificação, os valores das métricas `precision`, `recall` e `f1-score`, e a matriz de confusão, que foi representada graficamente com `seaborn`.

Adicionalmente, foi realizada validação cruzada (`cross-validation`) com dois objetivos complementares: numa primeira fase, com 5 folds, foram avaliadas as métricas `precision`, `recall` e `f1-score` com média ponderada; e, numa segunda fase, foi realizada uma validação com 10 folds centrada exclusivamente na métrica `f1-score`, permitindo analisar a sua variabilidade em diferentes subconjuntos. Os resultados foram exportados para ficheiros `.txt` e `.csv`, bem como visualizados graficamente.

Por fim, o modelo treinado e o respetivo scaler foram guardados com `jolib`, ficando preparados para utilização futura em ambiente real de teste.

2.2.2 Modelos Não Supervisionados Desenvolvidos

Os modelos não supervisionados utilizados neste projeto foram selecionados com base na sua capacidade de identificar padrões e anomalias sem necessidade de dados rotulados. Estes modelos foram configurados para analisar o tráfego de forma autônoma, permitindo distinguir entre comportamentos benignos e anômalos, mesmo na ausência de informação prévia sobre as classes.

Foram implementados os seguintes modelos não supervisionados:

- K-Means Clustering ([KM](#)): Uma técnica de agrupamento que organiza os dados em dois clusters (tráfego benigno e tráfego anômalo), com base nas suas características.
- Isolation Forest ([IF](#)): Um modelo especializado na detecção de anomalias, que isola rapidamente amostras anômalas com base na sua distinção em relação ao comportamento normal.
- Autoencoder ([AE](#)): Uma rede neuronal que aprende a reconstruir o comportamento normal dos dados, permitindo identificar anomalias com base na discrepância entre a entrada e a reconstrução.

Nos subcapítulos seguintes, são descritos em detalhe cada um destes modelos, incluindo os diversos testes realizados até à seleção do modelo mais adequado, bem como o seu pré-processamento, arquitetura e configuração final.

2.2.2.1 K-Means Clustering

A abordagem aos modelos não supervisionados teve início com a implementação do algoritmo K-Means, um dos métodos de agrupamento (clustering) mais utilizados. Tal como nos modelos supervisionados, foram realizados quatro testes com diferentes configurações, com o objetivo de analisar o impacto de vários parâmetros no comportamento do modelo. O primeiro teste consistiu numa versão baseline, e os testes seguintes introduziram ajustes progressivos. Através desta sequência de testes, procurou-se identificar a configuração mais adequada para segmentar os dados com base em padrões de similaridade, sem recurso a etiquetas de classe.

2.2.2.1.1 Primeiro Teste

No primeiro teste do modelo K-Means, implementou-se uma versão básica com o objetivo de estabelecer uma referência inicial de desempenho para este algoritmo de clustering. O pré-processamento dos dados seguiu o mesmo procedimento aplicado nos modelos supervisionados: carregamento do ficheiro `cleaned_ids2018.csv`, separação das variáveis

preditivas, normalização com StandardScaler e divisão estratificada dos dados em conjuntos de treino (70%) e teste (30%).

Dado que o [KM](#) é um algoritmo não supervisionado, os rótulos de classe não foram utilizados durante o treino. Contudo, para efeitos de avaliação, os rótulos foram convertidos para um formato binário: 0 para tráfego benigno e 1 para tráfego anômalo. O modelo foi treinado com dois grupos ($n_clusters=2$), correspondentes à tentativa de separar os dados entre tráfego normal e anômalo, e o treino foi realizado com o conjunto de treino.

Após a previsão sobre o conjunto de teste, foi necessário alinhar os rótulos dos clusters com os rótulos reais, assumindo inicialmente que o cluster 0 corresponde ao tráfego normal, e invertendo essa correspondência caso se verificasse o contrário. Por fim, o desempenho do modelo foi avaliado com base no relatório de classificação e na matriz de confusão, permitindo analisar a capacidade inicial de segmentação do algoritmo [KM](#) neste contexto.

2.2.2.1.2 Segundo Teste

No segundo teste, introduziram-se melhorias face à configuração base, com o objetivo de melhorar a eficiência e a separação entre os grupos identificados. Para tal, aplicou-se uma etapa de redução de dimensionalidade utilizando a técnica Principal Component Analysis ([PCA](#)). Esta técnica permitiu reduzir o número de atributos mantendo 95% da variância dos dados, o que ajudou a simplificar o espaço de representação sem perda significativa de informação.

O restante processo de preparação manteve-se idêntico: os dados foram normalizados com StandardScaler, os rótulos foram convertidos para binário (0 para tráfego benigno e 1 para anômalo), e os dados foram divididos de forma estratificada em conjuntos de treino e teste (70%-30%).

O modelo K-Means foi treinado com dois grupos ($n_clusters=2$), utilizando um número aumentado de inicializações ($n_init=20$) para melhorar a robustez da solução encontrada. Após a previsão sobre o conjunto de teste, os rótulos dos clusters foram alinhados com os rótulos reais, assumindo inicialmente que o cluster 0 correspondia ao tráfego normal, sendo invertido caso necessário.

A avaliação foi realizada com base no relatório de classificação e na matriz de confusão, permitindo verificar o impacto da redução de dimensionalidade e do reforço da inicialização na performance do modelo.

2.2.2.1.3 Terceiro Teste

No terceiro teste, foram introduzidas novas otimizações com o intuito de melhorar a qualidade dos agrupamentos. Para além da normalização dos dados e da redução de dimensionalidade com [PCA](#), foi adicionada uma etapa prévia de seleção de atributos com base na variância. Esta técnica, aplicada com `VarianceThreshold`, permitiu remover atributos com variância inferior a 0.01, ou seja, atributos com pouca capacidade discriminativa, reduzindo o ruído e a redundância presentes nos dados.

Em seguida, aplicou-se novamente a [PCA](#), preservando 95% da variância explicada. O conjunto resultante foi então dividido de forma estratificada em treino (70%) e teste (30%).

O modelo [KM](#) foi configurado com dois agrupamentos (`n_clusters=2`) e um número significativamente superior de inicializações (`n_init=50`), aumentando a probabilidade de convergir para uma solução globalmente estável e representativa.

Após o treino, as previsões foram alinhadas com os rótulos reais, assumindo inicialmente o cluster 0 como correspondente ao tráfego normal e invertendo essa correspondência, caso necessário. A avaliação do desempenho foi feita com recurso ao relatório de classificação e à matriz de confusão, permitindo analisar o impacto da filtragem por variância combinada com a redução de dimensionalidade no comportamento do modelo.

2.2.2.1.4 Quarto Teste

No quarto e último teste, foi implementada a versão mais completa e otimizada entre todas as testadas, integrando todas as melhorias aplicadas nos testes anteriores e ajustando diversos parâmetros com o objetivo de maximizar a qualidade dos agrupamentos.

O processo de preparação dos dados incluiu a normalização com `StandardScaler`, remoção de atributos com baixa variância (`VarianceThreshold` com `threshold=0.01`), e aplicação de [PCA](#) para reduzir a dimensionalidade, mantendo mais de 95% da variância explicada. Posteriormente, os dados foram divididos em treino (70%) e teste (30%) de forma estratificada.

O modelo K-Means foi configurado com dois grupos (`n_clusters=2`) e inicialização do tipo `k-means++`, que melhora a escolha dos centróides iniciais. Foi definido um número elevado de inicializações (`n_init=100`) e um limite máximo de iterações aumentado para 500 (`max_iter=500`), aumentando a probabilidade de convergir para uma solução estável e representativa dos dados.

Após o treino, os rótulos atribuídos aos clusters foram alinhados com os rótulos reais para permitir a avaliação do desempenho. Assumiu-se inicialmente que o cluster 0 correspondia ao tráfego normal, sendo esta correspondência invertida caso necessário. A avaliação foi realizada com base no relatório de classificação e na matriz de confusão, permitindo verificar o impacto global das otimizações aplicadas ao modelo K-Means.

2.2.2.1.5 Modelo Final

O modelo final selecionado para a abordagem não supervisionada com o algoritmo [KM](#) corresponde à configuração utilizada no Primeiro Teste, por ter apresentado os melhores resultados, os quais serão analisados numa secção posterior deste relatório.

O desenvolvimento foi realizado em Python, com recurso às bibliotecas pandas, numpy, scikit-learn, matplotlib, seaborn e joblib. O conjunto de dados `cleaned_ids2018.csv` foi carregado e os rótulos foram convertidos para binário (0 para tráfego benigno e 1 para tráfego anómalo). Os dados foram normalizados com `StandardScaler`.

A divisão dos dados foi feita de forma estratificada em conjuntos de treino (70%) e teste (30%). O modelo foi configurado com dois agrupamentos (`n_clusters=2`), inicialização automática (`n_init='auto'`) e `random_state=42`. Após o treino, os rótulos dos clusters foram alinhados com os rótulos reais, assumindo inicialmente que o cluster 0 correspondia à classe normal, sendo essa associação invertida se necessário.

A avaliação do modelo foi realizada com base no relatório de classificação, no F1-score e na matriz de confusão, que foi representada graficamente com seaborn. Adicionalmente, foi gerado um gráfico de barras com o F1-score para facilitar a leitura dos resultados.

Todos os ficheiros relevantes foram guardados: o modelo treinado, o scaler e os relatórios de avaliação, permitindo a reutilização do modelo em ambiente de teste.

2.2.2.2 Isolation Forest

O segundo modelo não supervisionado explorado foi o Isolation Forest, um algoritmo utilizado principalmente na deteção de anomalias, baseado no princípio de isolamento de instâncias fora do padrão. À semelhança dos restantes modelos, foram realizados quatro testes com diferentes configurações, começando com uma versão baseline e evoluindo progressivamente para versões mais otimizadas. O objetivo desta sequência de testes foi analisar a capacidade do modelo em identificar tráfego anómalo sem recurso a rótulos durante

o treino, avaliando o impacto de parâmetros como o número de estimadores, a profundidade das árvores e as estratégias de amostragem.

2.2.2.2.1 Primeiro Teste

No primeiro teste com o modelo [IE](#), foi utilizada a configuração base com os parâmetros predefinidos da biblioteca scikit-learn, servindo como versão baseline para avaliar o funcionamento geral do algoritmo no contexto da detecção de tráfego anômalo.

O conjunto de dados `cleaned_ids2018.csv` foi carregado e os rótulos foram convertidos para binário (0 para tráfego benigno e 1 para anômalo). Os dados foram normalizados com `StandardScaler` e divididos de forma estratificada em conjuntos de treino (70%) e teste (30%).

O modelo foi treinado apenas com os dados de treino, sendo configurado com `n_estimators=100` (número de árvores) e `contamination='auto'`, o que permite ao algoritmo estimar automaticamente a proporção de anomalias no conjunto de dados. Após o treino, o método `predict()` foi utilizado para classificar os exemplos do conjunto de teste, produzindo os valores 1 (normal) e -1 (anômalo), os quais foram convertidos para o formato binário esperado: 0 para normal e 1 para anômalo.

A avaliação foi realizada com base no relatório de classificação e na matriz de confusão, permitindo aferir a capacidade inicial do modelo em isolar padrões anômalos sem recurso a supervisão.

2.2.2.2.2 Segundo Teste

No segundo teste, foi introduzida uma melhoria ao nível da seleção de atributos, com o objetivo de reduzir o ruído nos dados e otimizar o desempenho do modelo. Para isso, aplicou-se o método `VarianceThreshold`, que removeu os atributos com variância inferior a 0.01, ou seja, com pouca capacidade de discriminação entre exemplos.

Tal como no teste anterior, os dados foram normalizados com `StandardScaler` e os rótulos convertidos para o formato binário (0 para tráfego benigno e 1 para anômalo). Após a remoção das colunas com baixa variância, os dados foram divididos de forma estratificada em conjuntos de treino (70%) e teste (30%).

O modelo foi treinado com os mesmos parâmetros do teste anterior: `n_estimators=100` e `contamination='auto'`, permitindo ao algoritmo estimar automaticamente a proporção de anomalias. As previsões foram obtidas com o método `predict()`, convertendo os valores 1 (normal) e -1 (anômalo) para 0 e 1, respetivamente.

A avaliação foi realizada com base no relatório de classificação e na matriz de confusão, permitindo analisar o impacto da filtragem de atributos com baixa variância no comportamento do modelo Isolation Forest.

2.2.2.2.3 Terceiro Teste

No terceiro teste, foi introduzida uma etapa adicional de redução de dimensionalidade com o objetivo de simplificar o espaço de representação e reduzir possíveis redundâncias nos dados. Após a remoção de atributos com baixa variância através de `VarianceThreshold`, aplicou-se o método [PCA](#), mantendo 95% da variância explicada.

O processo de preparação dos dados incluiu, tal como nos testes anteriores, a normalização com `StandardScaler` e a conversão dos rótulos para o formato binário (0 para tráfego benigno, 1 para anômalo). A divisão dos dados em treino e teste foi feita com estratificação, numa proporção de 70% para treino e 30% para teste.

O modelo [IF](#) manteve a configuração de `n_estimators=100` e `contamination='auto'`, sendo treinado exclusivamente com os dados de treino. A previsão foi feita sobre o conjunto de teste e os rótulos resultantes (1 para normal e -1 para anômalo) foram convertidos para 0 e 1, respetivamente, para efeitos de avaliação.

A performance do modelo foi analisada através do relatório de classificação e da matriz de confusão, permitindo avaliar o impacto da aplicação de [PCA](#) combinada com a seleção de atributos na eficácia da deteção de anomalias.

2.2.2.2.4 Quarto Teste

No quarto e último teste, foi implementada a versão mais completa e ajustada, integrando diversas otimizações ao nível do pré-processamento e dos hiperparâmetros do modelo. O objetivo foi maximizar a capacidade de deteção de anomalias, tendo em conta as características do conjunto de dados.

Os dados foram normalizados com `StandardScaler` e sujeitos a uma filtragem mais agressiva de atributos com baixa variância, utilizando `VarianceThreshold` com um limiar de 0.02. Em seguida, foi aplicada a técnica de [PCA](#), mantendo 95% da variância explicada. O conjunto resultante foi dividido, como nos testes anteriores, em treino (70%) e teste (30%) de forma estratificada.

O modelo [IF](#) foi configurado com `n_estimators=200`, aumentando o número de árvores em relação aos testes anteriores, e `max_samples=0.7`, definindo a fração de amostras utilizadas

por árvore. Mantiveram-se `max_features=1.0` para utilizar todos os atributos disponíveis em cada divisão, e o parâmetro `contamination` foi ajustado manualmente para 0.3, com base na proporção estimada de anomalias no conjunto de dados.

Após o treino com o conjunto de treino, as previsões foram obtidas para o conjunto de teste. Tal como nos testes anteriores, os valores 1 (normal) e -1 (anómalo) foram convertidos para 0 e 1, respetivamente. A avaliação foi efetuada com o relatório de classificação e a matriz de confusão, permitindo analisar o impacto global das otimizações aplicadas ao modelo Isolation Forest.

2.2.2.2.5 Modelo Final

O modelo final selecionado corresponde à configuração utilizada no Terceiro Teste, por ter apresentado os melhores resultados, os quais serão discutidos numa secção posterior deste relatório.

Tal como nos restantes modelos, o desenvolvimento foi realizado em Python, com recurso às bibliotecas `pandas`, `numpy`, `scikit-learn`, `matplotlib`, `seaborn` e `joblib`. O ficheiro `cleaned_ids2018.csv` foi carregado, os rótulos foram convertidos para binário (0 para tráfego benigno, 1 para anómalo), e os dados foram normalizados com `StandardScaler`. Posteriormente, foram removidos atributos com baixa variância (`threshold=0.01`) utilizando `VarianceThreshold` e aplicada redução de dimensionalidade com [PCA](#), preservando 95% da variância.

A divisão do conjunto de dados foi realizada de forma estratificada em treino (70%) e teste (30%). O modelo foi treinado com `n_estimators=100` e `contamination='auto'`, permitindo ao algoritmo estimar automaticamente a proporção de anomalias nos dados.

As previsões foram obtidas com o método `predict()`, cujos resultados (1 para normal e -1 para anómalo) foram convertidos para o formato binário convencional (0 e 1). A avaliação foi realizada com base no relatório de classificação, nas métricas F1-score, `precision` e `recall`, e na matriz de confusão, que foi visualizada com `seaborn`. Adicionalmente, foi gerado um gráfico de barras com o F1-score do modelo.

O modelo final treinado, o scaler e os ficheiros de avaliação foram guardados com `joblib`, ficando preparados para utilização futura em ambiente real de teste.

2.2.2.3 Autoencoder

O terceiro modelo não supervisionado aplicado foi o Autoencoder, uma rede neuronal com arquitetura simétrica, composta por uma fase de codificação e uma fase de decodificação.

Por se tratar de um modelo com funcionamento mais complexo, optou-se por realizar um teste adicional comparativamente com os modelos anteriores, dos quais quatro incidiram sobre diferentes configurações e estruturas do próprio modelo, e um focado exclusivamente no experimento de vários thresholds, com o objetivo de identificar o valor mais adequado a aplicar na deteção de anomalias. Esta abordagem permitiu explorar de forma mais aprofundada a sensibilidade do modelo a diferentes parâmetros e avaliar a sua eficácia na reconstrução de dados normais, medindo os desvios como critério para identificação de comportamentos anómalos.

2.2.2.3.1 Primeiro Teste

No primeiro teste, foi implementada uma versão básica do Autoencoder com o objetivo de avaliar a sua capacidade inicial de reconstrução de dados normais e deteção de anomalias com base no erro de reconstrução. O pré-processamento dos dados seguiu as etapas já definidas: carregamento do dataset `cleaned_ids2018.csv`, separação das variáveis preditivas, normalização com `StandardScaler` e divisão estratificada dos dados em treino (70%) e teste (30%).

Como o Autoencoder é treinado exclusivamente com dados normais, foi extraído do conjunto de treino apenas o subconjunto de amostras com rótulo benigno (0). A arquitetura utilizada foi simples e simétrica, composta por uma camada de codificação com 32 neurónios (função de ativação ReLU) e uma camada de decodificação com o mesmo número de neurónios que o input (ativação linear). O modelo foi compilado com o otimizador Adam e função de perda Mean Squared Error ([MSE](#)), e treinado durante 20 épocas com batch size de 128.

Após o treino, o modelo foi utilizado para reconstruir as amostras do conjunto de teste, sendo o erro de reconstrução calculado como a média dos quadrados das diferenças entre as amostras originais e reconstruídas. O threshold de deteção foi definido automaticamente como o percentil 95 do erro de reconstrução obtido no conjunto de treino normal, assumindo-se que valores acima deste limiar indicam anomalias.

A classificação final foi obtida comparando os erros de reconstrução com este threshold, e o desempenho do modelo foi avaliado com base no relatório de classificação, F1-score e matriz de confusão, permitindo aferir a eficácia desta versão inicial do Autoencoder na deteção de tráfego anómalo.

2.2.2.3.2 Segundo Teste

No segundo teste, foi introduzida uma otimização no pré-processamento dos dados através da aplicação da técnica de filtragem por variância, com o objetivo de eliminar atributos com pouca capacidade discriminativa. Utilizou-se o método `VarianceThreshold` com um limiar de 0.01, removendo variáveis cuja variância fosse inferior a este valor, reduzindo assim o ruído presente nos dados e potencialmente melhorando a capacidade de generalização do modelo.

O restante pipeline manteve-se semelhante ao teste anterior: os dados foram normalizados com `StandardScaler`, os rótulos foram convertidos para binário (0 para tráfego benigno e 1 para anómalo), e os dados foram divididos de forma estratificada em conjuntos de treino (70%) e teste (30%). Tal como no primeiro teste, o modelo foi treinado exclusivamente com as amostras benignas do conjunto de treino.

A arquitetura do Autoencoder manteve-se inalterada: uma camada de codificação com 32 neurónios (ativação ReLU) e uma camada de descodificação com ativação linear. O treino decorreu durante 20 épocas com batch size de 128, utilizando como função de perda o [MSE](#).

Para deteção de anomalias, foi novamente calculado o erro de reconstrução em cada amostra do conjunto de teste, e utilizado como threshold o percentil 95 dos erros observados no treino com dados benignos. As amostras com erro superior a este valor foram classificadas como anómalas.

A avaliação do desempenho foi feita com recurso ao relatório de classificação, matriz de confusão e F1-score.

2.2.2.3.3 Terceiro Teste

No terceiro teste, foi introduzida uma melhoria na arquitetura do [AE](#), com o objetivo de aumentar a capacidade de representação do modelo e melhorar a sua performance na deteção de anomalias. A nova arquitetura adotou uma estrutura mais profunda e simétrica, composta por duas camadas densas na fase de codificação (com 64 e 32 neurónios, respetivamente) e duas camadas na fase de descodificação (com 64 neurónios e saída linear), permitindo ao modelo capturar padrões mais complexos nos dados.

O conjunto de dados foi pré-processado da mesma forma que nos testes anteriores: os dados foram normalizados com `StandardScaler`, os rótulos foram convertidos para binário (0 para tráfego benigno e 1 para anómalo), e os dados foram divididos estratificadamente em treino (70%) e teste (30%).

O treino foi realizado exclusivamente com dados benignos, durante 30 épocas, utilizando um batch size de 128 e o otimizador Adam com taxa de aprendizagem de 0.001. Após o treino, o erro de reconstrução foi calculado para cada instância do conjunto de teste, e definiu-se como threshold o percentil 95 do erro de reconstrução observado nas amostras benignas do treino.

As previsões finais classificaram como anómalas todas as instâncias cujo erro de reconstrução ultrapassou esse threshold. A avaliação do modelo foi feita com base no relatório de classificação, matriz de confusão e F1-score.

2.2.2.3.4 Quarto Teste

No quarto teste, o [AE](#) foi ajustado com uma arquitetura ainda mais profunda e complexa, com o intuito de aumentar a expressividade da rede e a sua capacidade de reconstrução dos dados normais. A estrutura adotada incluiu três camadas densas na fase de codificação (com 128, 64 e 32 neurónios, respetivamente) e três camadas na fase de decodificação (64, 128 e camada de saída linear), mantendo a simetria típica deste tipo de redes.

O pré-processamento dos dados manteve-se consistente com os testes anteriores: normalização com StandardScaler, binarização dos rótulos (0 para tráfego benigno e 1 para anómalo), e divisão estratificada dos dados em treino (70%) e teste (30%). O modelo foi treinado apenas com amostras benignas durante 40 épocas, com batch size de 128 e o otimizador Adam com uma taxa de aprendizagem de 0.001.

Após o treino, foram calculados os erros de reconstrução para os dados de teste, com base na média dos erros quadráticos. O threshold para classificação foi definido com base no percentil 95 do erro das amostras benignas do treino.

As previsões finais foram obtidas comparando o erro de reconstrução com esse threshold, classificando como anómalas as instâncias com erro superior. A avaliação foi realizada com base no relatório de classificação, matriz de confusão e F1-score, permitindo observar o impacto do aumento da profundidade da rede na capacidade de deteção de tráfego anómalo.

2.2.2.3.5 Modelo Final

O modelo final selecionado para o [AE](#) corresponde à configuração aplicada no Quarto Teste, por ter apresentado o melhor desempenho entre as versões avaliadas.

O desenvolvimento foi realizado em Python, utilizando as bibliotecas pandas, numpy, scikit-learn, matplotlib, seaborn, joblib e TensorFlow. O ficheiro `cleaned_ids2018.csv` foi carregado, sendo os rótulos convertidos para binário (0 para tráfego benigno e 1 para tráfego anómalo).

Os dados foram divididos em conjuntos de treino (70%) e teste (30%) de forma estratificada. A normalização foi realizada com `StandardScaler` ajustado apenas sobre os dados benignos do conjunto de treino, garantindo que o modelo aprenda exclusivamente os padrões normais.

A arquitetura incluiu três camadas de codificação (128, 64 e 32 neurónios com ativação ReLU) e três camadas de decodificação (64 e 128 neurónios com ReLU e uma camada final linear com dimensão igual à entrada). O modelo foi treinado com os dados benignos normalizados do conjunto de treino ao longo de 40 épocas, com batch size de 128.

O erro de reconstrução foi calculado com [MSE](#) entre a entrada e a saída. O threshold utilizado para a classificação foi definido com base no percentil 95 do erro obtido nos dados de treino benignos.

A avaliação incluiu `classification_report`, F1-score, matriz de confusão e representação gráfica. Os ficheiros gerados, incluindo o modelo treinado (`ae_model.keras`), o scaler (`scaler_ae.pkl`) e os relatórios de avaliação, foram guardados para posterior utilização em ambiente real.

2.2.2.3.6 Teste Threshold

O quinto e último teste realizado teve como objetivo identificar o threshold mais adequado para a deteção de anomalias com base no erro de reconstrução. Para isso, foi utilizado o modelo previamente treinados (correspondente ao Modelo Final) e o scaler associado, ambos carregados diretamente dos ficheiros guardados.

Os dados foram normalizados com o `StandardScaler` treinado anteriormente, e o erro de reconstrução foi calculado para cada amostra do conjunto de teste através do [MSE](#) entre os valores reais e os valores reconstruídos. Em seguida, foram testados múltiplos thresholds fixos (50, 75, 100, 125, 150, 200, 300, 500, 1000), com o objetivo de avaliar a sensibilidade do modelo a diferentes limites de decisão.

Para cada threshold, foi gerada uma classificação binária (0 para normal, 1 para anómalo), sendo avaliados os valores de F1-score, precision e recall. Os resultados obtidos foram guardados num ficheiro `.csv` e o melhor threshold, com base no maior F1-score, foi

registado num ficheiro .txt para referência futura. Esta análise permitiu selecionar de forma empírica o threshold mais eficaz para a deteção de comportamentos anómalos, ajustando o equilíbrio entre falsos positivos e falsos negativos consoante o desempenho observado.

2.3 Testes Preliminares com os Modelos de Machine Learning

Nesta secção é apresentado o desenvolvimento dos testes preliminares realizados com os seis modelos finais construídos. Estes testes tiveram como objetivo avaliar o desempenho dos modelos num ambiente semelhante ao de testes, de modo a verificar o seu comportamento em contextos práticos, com dados realistas e desconhecidos pelos modelos.

2.3.1 Testes Preliminares nos Modelos Supervisionados

O primeiro teste preliminar desenvolvido (teste_supervised.py) teve como objetivo integrar os modelos finais supervisionados ([MLP](#), [RF](#) e [SVM](#)) num sistema de análise de fluxos de rede, de modo a simular um ambiente prático e próximo de um cenário real de utilização. O desenvolvimento recorreu às bibliotecas pandas, numpy, scikit-learn, matplotlib, seaborn, joblib, colorama e datetime.

O sistema foi concebido para monitorizar, de forma contínua, uma pasta específica destinada à receção de ficheiros .csv contendo fluxos de rede. Para cada ficheiro detetado, os dados recolhidos do ambiente de testes foram carregados, sendo verificada a presença das colunas esperadas com base no ficheiro features.txt e no dicionário FEATURE_MAP. Os dados foram então preparados através da normalização com os scalers, anteriormente guardados, correspondentes a cada modelo, garantindo a consistência do pré-processamento.

As previsões foram realizadas individualmente por cada modelo, sendo registado o número de fluxos suspeitos identificados e as classes detetadas, com recurso ao dicionário CLASS_NAMES para facilitar a interpretação dos resultados. Estes resultados foram documentados em relatórios, ilustrado na Figura 4, gerados automaticamente num ficheiro de texto, com indicação do ficheiro analisado, data e hora de execução, número de fluxos suspeitos e respetivas classes detetadas por cada modelo.

```

1 Relatório gerado em: 2025-07-01 11:29:04
2 Ficheiro analisado: tcp_syn_flooding.csv
3 MLP: 521 fluxos suspeitos em 2241
4 Classes detetadas: {'DDOS': 159, 'Bot': 362}
5 Random Forest: 0 fluxos suspeitos em 2241
6 Classes detetadas: {}
7 SVM: 0 fluxos suspeitos em 2241
8 Classes detetadas: {}
9

```

Figura 4 - Exemplo de Relatório de Resultados de Testes Preliminares

Fonte desta figura: Exemplo de resultado da execução do ficheiro teste_supervised.py.

Para complementar a avaliação, foram gerados gráficos comparativos para cada ficheiro processado. Estes gráficos representaram, em dois painéis, o número de fluxos suspeitos detetados por cada modelo e a distribuição das classes detetadas, permitindo uma visualização clara e imediata dos resultados. Os gráficos foram guardados automaticamente em formato de imagem, ficando disponíveis para posterior consulta.

Este teste permitiu avaliar o desempenho dos modelos supervisionados em contexto prático, procurando reforçar a análise da sua capacidade de generalização, encontrando o mais adequado para utilização no [IDS](#) em ambiente realista.

2.3.2 Testes Preliminares nos Modelos Não Supervisionados

O segundo teste preliminar desenvolvido (teste_unsupervised.py) teve como objetivo integrar os modelos finais não supervisionados ([AE](#), [IF](#) e [KM](#)) num sistema de análise contínua de fluxos de rede, aproximando o processo às condições reais de utilização do [IDS](#). O desenvolvimento do teste recorreu às bibliotecas pandas, numpy, scikit-learn, tensorflow.keras, matplotlib, seaborn, joblib, colorama e datetime.

O sistema foi desenhado para monitorizar de forma contínua uma pasta específica destinada à receção de ficheiros .csv contendo fluxos de rede. Para cada ficheiro detetado, os dados recolhidos do ambiente de testes foram carregados, sendo verificada a presença das colunas esperadas com base no ficheiro features.txt e no dicionário FEATURE_MAP. Os dados foram preparados através da normalização com os scalers previamente guardados e, no caso do modelo [IF](#), foram aplicadas adicionalmente as transformações de redução de dimensionalidade com VarianceThreshold e [PCA](#), garantindo a consistência do pré-processamento em relação ao treino dos modelos.

As previsões foram realizadas individualmente por cada modelo. No caso do [AE](#), foi calculado o [MSE](#), sendo os fluxos classificados como suspeitos quando o valor excedia o limiar previamente definido em `ae_threshold.txt`. No [IF](#), os fluxos foram classificados diretamente como normais ou anómalos com base nas previsões do modelo. No [KM](#), os fluxos foram classificados como suspeitos quando atribuídos ao cluster diferente do considerado normal. Os resultados foram registados em relatórios, semelhante ao ilustrado na Figura 4, contendo a data e hora de execução, o ficheiro analisado, o número de fluxos suspeitos identificados e a distribuição por categoria em cada modelo.

Para complementar a avaliação, foram gerados gráficos comparativos para cada ficheiro processado, representando o número de fluxos suspeitos detetados por cada modelo e a distribuição das categorias atribuídas. Estes gráficos foram guardados automaticamente em formato de imagem, ficando disponíveis para posterior consulta. Este teste permitiu observar o desempenho dos modelos não supervisionados em contexto prático, procurando reforçar a análise da sua robustez, encontrando o mais adequado para utilização no [IDS](#) em ambiente realista.

2.4 Arquitetura Final do IDS

O desenvolvimento da arquitetura final do [IDS](#) consistiu na criação de um script em Python responsável pela monitorização e análise automática de ficheiros .csv contendo fluxos de rede. Para a implementação deste script foram utilizadas as bibliotecas `numpy`, `pandas`, `joblib`, `tensorflow.keras`, `colorama`, `os`, `time` e `datetime`.

O script inicia com o carregamento do modelo supervisionado [MLP](#), modelo escolhido com base nos resultados obtidos nos testes, guardado em ficheiro `mlp.pkl`, e do respetivo `scaler_mlp.pkl`, ambos carregados com `joblib`. Em paralelo, é carregado o modelo não supervisionado [AE](#), também selecionado pelos resultados dos testes, a partir do ficheiro `ae_model.keras`, o `scaler_ae.pkl` e o valor de limiar de [MSE](#) armazenado no ficheiro `ae_threshold.txt`. É ainda carregada a lista das colunas esperadas a partir do ficheiro `features.txt`.

Foi definida a pasta “Flow_outputs” como diretório de entrada para os ficheiros .csv a processar e a pasta Relatórios como diretório de saída para os relatórios de execução. O script cria automaticamente um ficheiro de relatório com numeração sequencial (`relatorio_1.txt`, `relatorio_2.txt`, ...), garantindo que os registos de cada execução são armazenados sem sobreposição.

O ciclo principal do script verifica, com um intervalo de 1 segundo, a existência de novos ficheiros .csv na pasta “Flow_outputs”. Apenas são processados ficheiros com conteúdo e que não estejam a ser escritos no momento. Cada ficheiro é carregado e o sistema confirma a presença e a ordem correta das colunas necessárias.

O fluxo de funcionamento do [IDS](#) encontra-se representado na Figura 5, ilustrando a sequência de decisões aplicadas sobre o tráfego captado. O tráfego é inicialmente analisado pelo modelo supervisionado. Caso seja detetado um ataque, é gerado um alerta e o processo termina para esse ficheiro. Quando o modelo supervisionado não deteta ataques, os fluxos são processados pelo modelo não supervisionado. Os fluxos com erro de reconstrução superior ao limiar estabelecido são encaminhados para validação manual pelo utilizador. Quando o erro se encontra abaixo do limiar, ou após a validação, os fluxos são considerados benignos e não é acionada qualquer ação.

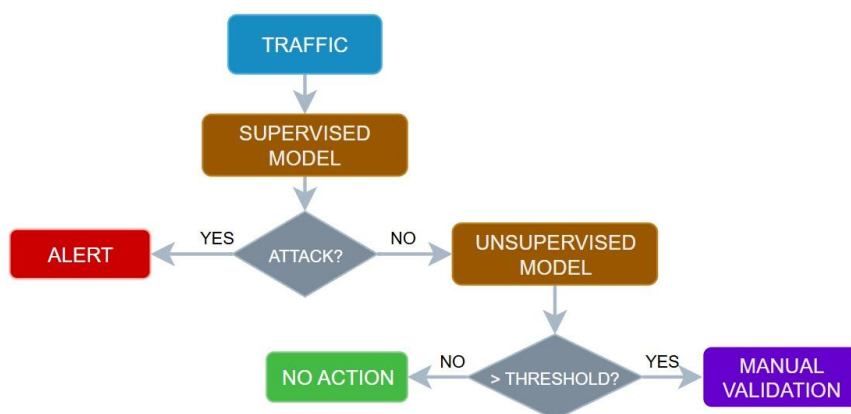


Figura 5 - Fluxo de Funcionamento da Arquitetura do IDS

Fonte desta figura: Elaboração própria.

Na primeira etapa do processo, os dados são normalizados com o `scaler_mlp` e analisados com o [MLP](#). Se forem identificados fluxos anómalos (classes diferentes de 1), o sistema regista o número de fluxos suspeitos, identifica a classe mais frequente, apresenta o resultado no terminal e escreve o registo no relatório.

Caso o [MLP](#) não detete anomalias, o ficheiro é analisado pelo [AE](#). As features são transformadas com o `scaler_ae` e processadas pelo modelo. É calculado o [MSE](#) de reconstrução de cada fluxo e comparado com o limiar definido. Os fluxos com [MSE](#) superior ao limiar são apresentados ao utilizador no terminal para validação manual, permitindo a classificação de cada fluxo como anómalo ou benigno. Os fluxos validados como anómalos são exportados para ficheiros .csv na pasta Validações.

O sistema apresenta no terminal o resumo final de cada ficheiro processado e regista todos os eventos no relatório de execução. Esta arquitetura permite o processamento automático dos fluxos de rede, a integração dos modelos supervisionado e não supervisionado e a rastreabilidade completa das decisões.

2.5 Configuração do Ambiente de Teste

Nesta secção é apresentada a configuração do ambiente de testes utilizado e configurado para a implementação e uso prático do [IDS](#).

2.5.1 Preparação das Máquinas Virtuais

A configuração do ambiente de testes teve início com a obtenção da imagem oficial do sistema operativo Kali Linux, otimizada para ambientes virtuais, mais concretamente para a plataforma VMware Workstation Pro [2]. Esta imagem foi descarregada a partir do website oficial do Kali Linux [3]. A escolha da plataforma e do sistema operativo, deveu-se à familiarização das mesmas pelo grupo.

Após concluído o processo de descarga, procedeu-se à importação da máquina virtual no VMware Workstation Pro. Como ilustrado na Figura 6 esta operação é realizada através da opção “Open a Virtual Machine”, onde se navega até ao diretório onde foi guardado o ficheiro de configuração da máquina virtual descarregada. Uma vez selecionado este ficheiro, a máquina é adicionada ao ambiente de gestão do VMware, ficando pronta para ser executada.

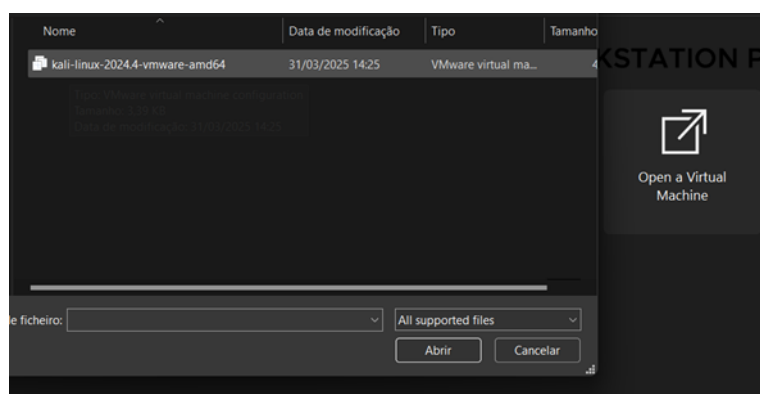


Figura 6 - Importação do Sistema Operativo Kali

Fonte desta figura: Captura de Ecrã do VMware Workstation Pro.

Após a configuração inicial da primeira máquina virtual com o sistema operativo Kali Linux na plataforma VMware Workstation Pro, procedeu-se à clonagem de mais duas máquinas virtuais adicionais através da utilização da ferramenta Clone Virtual Machine Wizard,

conforme demonstrado na Figura 7. Este procedimento teve como objetivo perfazer o total de três máquinas necessárias à execução do nosso cenário experimental. Estas máquinas correspondem respetivamente às funções de atacante (invasor), de IDS e, por último, à máquina que desempenha o papel de servidor. Esta última será a máquina-alvo das atividades maliciosas nos testes realizados, tendo como objetivo verificar a eficácia do IDS na deteção do tráfego malicioso gerado durante o processo de invasão na rede.



Figura 7 - Clone Wizard para Cópias de Máquinas Virtuais Kali

Fonte desta figura: Captura de Ecrã do VMware Workstation Pro.

Na Figura 8, é possível observar o passo seguinte no processo de clonagem, onde o assistente Clone Virtual Machine Wizard pede ao utilizador que selecione o estado a partir do qual será criada a nova máquina virtual. Neste contexto específico, optou-se pela primeira opção, por esta permitir criar diretamente um clone do estado atual da máquina, sem a necessidade prévia de criação de um snapshot, que consiste numa captura ou registo instantâneo e estático do estado exato de uma máquina virtual num dado momento específico, incluindo configurações, dados e estado de execução.

A segunda opção disponível, requer obrigatoriamente que exista já um snapshot previamente criado. Esta opção é adequada quando se pretende utilizar um ponto específico já registado da configuração da máquina virtual, garantindo uma configuração estática e consistente para a clonagem. Contudo, no nosso cenário, a flexibilidade proporcionada pela primeira opção foi preferida, uma vez que o estado atual da máquina virtual Kali Linux satisfazia plenamente os requisitos estabelecidos para a experiência.

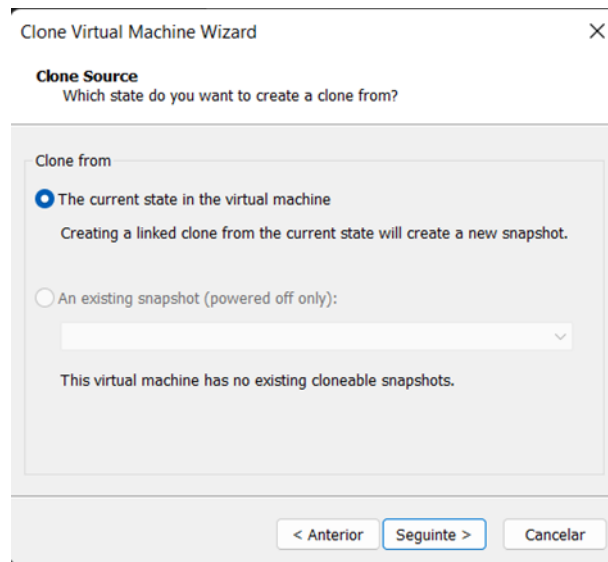


Figura 8 - Estado do Clone

Fonte desta figura: Captura de Ecrã do VMware Workstation Pro.

Na Figura 9 é apresentada a etapa do assistente Clone Virtual Machine Wizard onde é necessário selecionar o tipo de clonagem a efetuar, sendo disponibilizados dois métodos distintos: linked clone e full clone.

O método linked clone cria uma máquina virtual que funciona como uma referência dependente da máquina original. Neste tipo de clonagem, o novo sistema partilha os ficheiros de disco da máquina original, ocupando, por isso, menos espaço em disco. No entanto, esta dependência implica que a nova máquina não pode ser executada de forma autónoma sem acesso contínuo à máquina virtual original, no caso da máquina original ficar indisponível todas as outras ficariam também.

Já o método full clone gera uma cópia completa e independente da máquina virtual original no estado atual. Esta nova máquina virtual é inteiramente autónoma, não mantendo qualquer ligação técnica com a original. Embora este método exija mais espaço em disco, garante total independência entre as máquinas clonadas.

Para o nosso projeto, optou-se pela utilização do método full clone, uma vez que era fundamental garantir a autonomia total de cada uma das três máquinas virtuais. O único ponto comum entre elas deve ser apenas a rede local onde se encontram inseridas, permitindo assim simular de forma mais realista os diferentes papéis de atacante, servidor e [IDS](#), sem dependências entre máquinas.

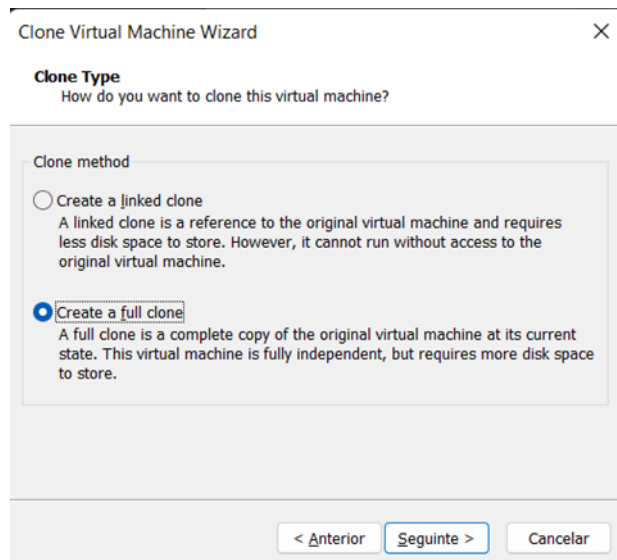


Figura 9 - Tipo de Clone

Fonte desta figura: Captura de Ecrã do VMware Workstation Pro.

Na Figura 10 é apresentado o passo final do processo de clonagem, no qual é atribuído um nome à nova máquina virtual a ser criada, bem como definida a localização no sistema de ficheiros do computador físico onde esta será armazenada. A definição de um nome claro e descritivo, neste exemplo apresentado é “Server VM”, permite uma gestão mais eficaz das diferentes máquinas virtuais, facilitando a sua identificação no contexto do projeto. Adicionalmente, a seleção do diretório de destino no armazenamento local é essencial para garantir uma organização adequada e evitar conflitos com outras máquinas virtuais existentes no sistema.

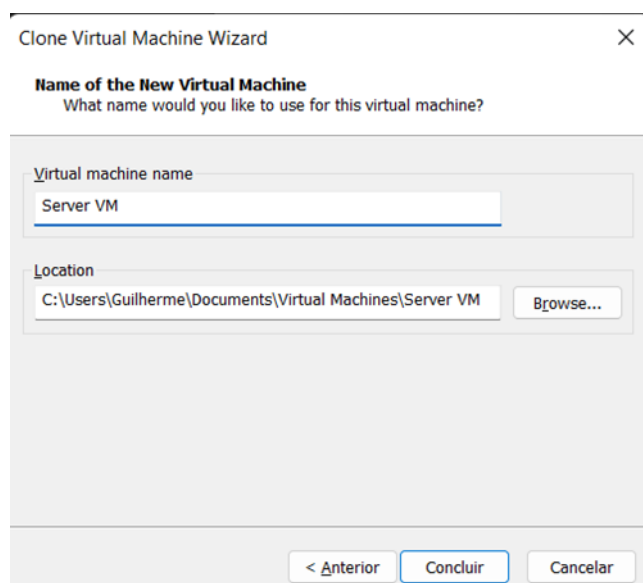


Figura 10 - Nome da Máquina e Localização da Máquina Física

Fonte desta figura: Captura de Ecrã do VMware Workstation Pro.

2.5.2 Configuração da Rede Virtual

Na Figura 11 encontra-se representada a configuração da rede virtual criada especificamente para suportar a comunicação entre as três máquinas virtuais envolvidas no ambiente de testes: o [IDS](#), o atacante e o servidor. Para tal, foi criada a interface de rede virtual designada por VMnet2, através da ferramenta Virtual Network Editor do VMware Workstation Pro.

Optou-se pelo tipo de rede Host-only, cuja principal característica consiste em estabelecer uma rede isolada entre as máquinas virtuais, sem qualquer acesso direto à rede externa ou à Internet. Esta escolha foi motivada por questões de segurança, uma vez que permite isolar completamente a rede de testes da rede real dos utilizadores, evitando possíveis interferências ou riscos de propagação de tráfego malicioso para o ambiente físico.

Adicionalmente, foi desativada a opção “Connect a host adapter to this network”, de forma a impedir que a máquina física (host) participe nesta rede virtual. Com esta medida, reforça-se ainda mais o isolamento entre a rede real e o ambiente de testes, garantindo que todo o tráfego gerado permanece restrito ao contexto virtualizado.

Para facilitar a gestão de endereços Internet Protocol ([IP](#)) no interior desta rede, foi ativado o serviço de Dynamic Host Configuration Protocol ([DHCP](#)), permitindo que os endereços [IP](#) sejam atribuídos automaticamente às máquinas virtuais que se conectarem à VMnet2. A sub-rede definida para esta rede privada foi 192.168.10.0, com uma máscara de sub-rede 255.255.255.0, assegurando uma gama suficiente de endereços para acomodar os dispositivos necessários neste cenário experimental.

Esta configuração garante, assim, um ambiente de testes controlado, isolado e funcionalmente autónomo, adequado para a realização segura de atividades de análise de tráfego e deteção de intrusões.

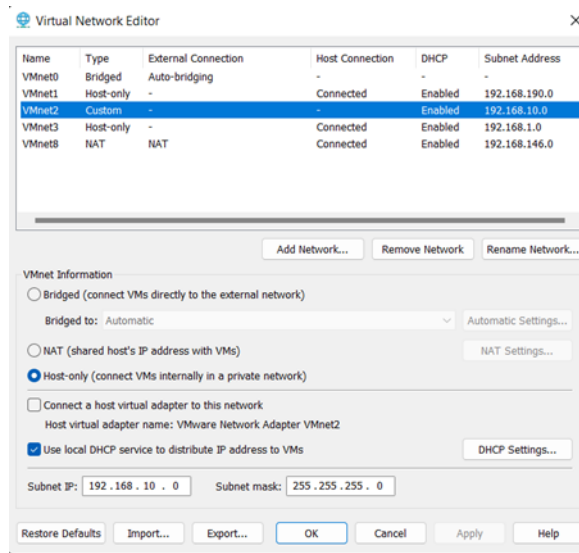


Figura 11 - Criação da VMnet2

Fonte desta figura: Captura de Ecrã do VMware Workstation Pro.

A Tabela 1 resume a configuração básica das três máquinas virtuais utilizadas no ambiente de testes. Para cada uma delas, são indicados o respetivo nome funcional, o sistema operativo instalado e o endereço [IP](#) atribuído dentro da sub-rede privada 192.168.10.0/24, previamente configurada na interface virtual VMnet2.

Tabela 1 - Configuração das Três Máquinas Virtuais

| Máquina | Sistema Operativo | IP |
|---------------------|-------------------|--------------------|
| IDS | Kali Linux | 192.168.10.128 |
| Atacante | Kali Linux | 192.168.10.129 |
| Vítima | Kali Linux | 192.168.10.130 |

Fonte desta tabela: Elaboração própria.

2.5.3 Configuração de Hardware das Máquinas Virtuais

Na Figura 12 é apresentada a configuração de hardware virtual de uma das máquinas utilizadas no ambiente de testes, neste caso, a máquina dedicada ao [IDS](#). Esta configuração foi replicada de forma idêntica nas restantes máquinas virtuais, de modo a garantir homogeneidade nos recursos atribuídos e coerência nos resultados experimentais.

Cada uma das máquinas virtuais foi configurada com 2 GB de memória Random Access Memory ([RAM](#)), valor considerado suficiente para garantir o funcionamento estável do sistema operativo Kali Linux e das ferramentas necessárias à realização dos testes. Em termo de capacidade de processamento, foram atribuídos 4 núcleos de Central Processing Unit ([CPU](#))

virtuais a cada máquina, permitindo uma execução eficiente e paralelizada de tarefas, nomeadamente durante operações computacionalmente intensivas como análises de tráfego de rede ou varreduras de portas e serviços. No que diz respeito ao armazenamento, cada máquina virtual foi provisionada com um disco virtual com uma capacidade total de 80 GB, assegurando espaço suficiente para instalação do sistema, das aplicações de suporte e para o registo de logs e resultados. Por fim, todas as máquinas foram configuradas com uma placa de rede virtual ligada à interface VMnet2, criada anteriormente através do Virtual Network Editor, permitindo que as comunicações entre máquinas se processem de forma exclusiva no interior da rede virtual isolada, conforme descrito na Figura 11.

Esta configuração assegura um equilíbrio entre desempenho e consumo de recursos físicos do sistema anfitrião, proporcionando um ambiente controlado, seguro e tecnicamente adequado à realização de testes de deteção de intrusões e análise de tráfego em redes locais simuladas.

Realizou-se ainda, temporariamente, a alteração do adaptador de rede da máquina virtual [IDS](#) para Network Address Translation ([NAT](#)), de modo a permitir instalar todas as dependências necessárias ao projeto, incluindo o Visual Studio Code ([VSC](#)) [4] e CICFlowMeter [5], este último será abordado mais à frente no relatório.

A escolha do [VSC](#) justificou-se pelo facto de se tratar de um editor de código leve, amplamente utilizado, com suporte nativo para Python, integração com terminais, sistema de extensões e funcionalidades úteis como a deteção automática de ambientes virtuais e o realce de sintaxe, o que facilita substancialmente o desenvolvimento e a depuração do código do [IDS](#).

Após a instalação de todas as dependências que requerem ligação à internet, o adaptador de rede da máquina [IDS](#) foi novamente configurado para Custom > VMnet2. A máquina foi reiniciada e o processo de configuração prosseguiu com a rede virtual já definida e isolada.

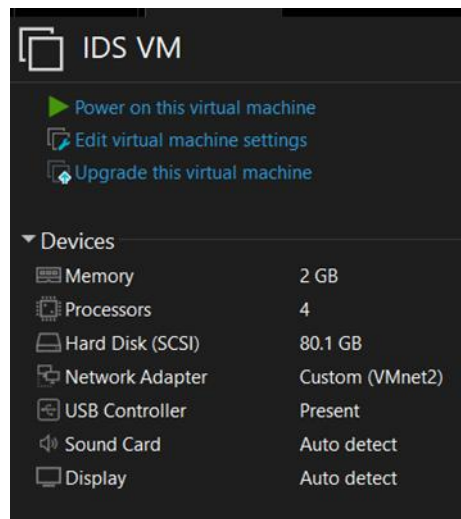


Figura 12 - Configuração das Máquinas Virtuais

Fonte desta figura: Captura de Ecrã do VMware Workstation Pro.

2.5.4 Configuração do IDS em Modo Promísco

Na Figura 13 é demonstrada a configuração, via terminal, da interface de rede da máquina virtual destinada ao [IDS](#), de forma a colocá-la em modo promísco. Esta operação foi realizada especificamente na interface de rede eth0, que corresponde à interface de rede ativa atribuída à máquina virtual aquando da sua criação no VMware Workstation Pro. A escolha da eth0 prende-se com o facto de esta ser, por padrão, a primeira e, geralmente, a única interface de rede disponibilizada e configurada automaticamente em ambientes virtuais simples, sendo, portanto, aquela por onde todo o tráfego de rede é recebido e transmitido.

```
File Actions Edit View Help
(kali@kali)-[~]
$ sudo ip link set eth0 promisc on
[sudo] password for kali:
(kali@kali)-[~]
$ ip link show eth0
2: eth0: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc fq_codel st
ate UP mode DEFAULT group default qlen 1000
    link/ether 00:0c:29:c7:03:29 brd ff:ff:ff:ff:ff:ff
(kali@kali)-[~]
$
```

Figura 13 - Configuração do IDS em Modo Promísco

Fonte desta figura: Captura de Ecrã do VMware Workstation Pro.

A ativação do modo promísco foi efetuada com o seguinte comando: `sudo ip link set eth0 promisc on`

Este comando permite que a interface eth0 passe a aceitar e processar todos os pacotes de rede que circulam no segmento de rede onde está inserida, independentemente do seu destino. Esta funcionalidade é fundamental para o correto funcionamento de um [IDS](#), pois só assim este conseguirá analisar o tráfego da rede em tempo real e identificar possíveis atividades maliciosas, mesmo que essas não sejam diretamente direcionadas para o seu próprio endereço de rede.

De seguida, foi utilizado o comando: `ip link show eth0`

Este comando serve para verificar o estado atual da interface de rede eth0, confirmando se o modo promíscuo foi corretamente ativado. Na saída apresentada é possível observar o campo “PROMISC” incluído na lista de opções da interface, o que confirma que esta se encontra em modo promíscuo, apta a monitorizar todo o tráfego que percorre a rede local. Esta configuração é, portanto, essencial para assegurar que o [IDS](#) opera com a visibilidade necessária sobre o ambiente de rede a proteger.

2.5.5 Integração do CICFlowMeter no Ambiente de Testes

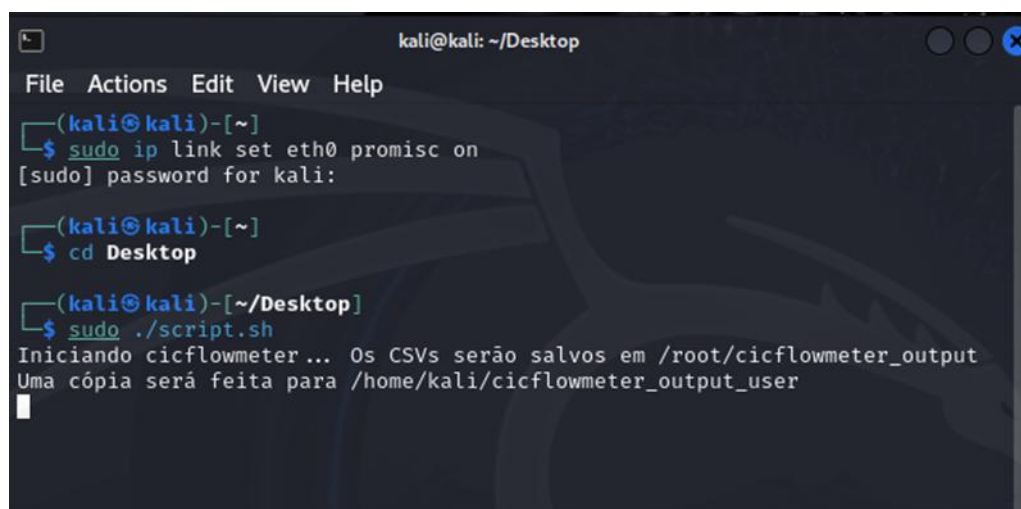
Para realizar a monitorização e extração de dados do tráfego de rede em tempo real, foi utilizada a ferramenta CICFlowMeter, instalada na [VM](#) destinada ao [IDS](#). Esta ferramenta tem como principal função a conversão de pacotes de rede em fluxos de rede (network flows), como base num conjunto de atributos técnicos como a duração do fluxo, número de pacotes, número de bytes, entre outros parâmetros relevantes. Estas features coincidem com as utilizadas na construção de diversos datasets públicos para análise de tráfego e deteção de intrusões, nomeadamente o CSE-CIC-IDS2018, o que torna o CICFlowMeter especialmente adequado à integração com sistemas de deteção baseados em [ML](#) ou outras formas de análise estatística.

A sua utilização neste projeto justifica-se precisamente pela necessidade de estabelecer uma ponte entre o tráfego de rede em tempo real e o [IDS](#). O CICFlowMeter escuta o tráfego em tempo real (neste caso, na interface eth0 do Kali Linux da máquina do [IDS](#)) e transforma essa informação em ficheiros com extensão .csv, nos quais cada linha representa um fluxo de rede descrito através das features anteriormente mencionadas. Esta conversão é essencial para que o nosso [IDS](#) possa analisar os dados, uma vez que este foi concebido para trabalhar com entradas no formato .csv, compatíveis com o modelo de dados do dataset de treino.

Durante a instalação do CICFlowMeter, optou-se por executá-la dentro de um ambiente virtual Python criado especificamente para o efeito, recorrendo as ferramentas como “venv”. Esta abordagem teve como principal objetivo isolar as dependências do CICFlowMeter, que

incluem múltiplos pacotes Python específicos, das restantes aplicações instaladas no sistema operativo da máquina virtual. Caso a instalação fosse feita diretamente no sistema, poderia comprometer pacotes partilhados como pip entre outros, levando a possíveis incompatibilidades ou corrupção de ambientes de desenvolvimento já existentes. A criação deste ambiente virtual assegura, assim, que todas as bibliotecas e versões utilizadas pelo CICFlowMeter ficam confinadas ao seu próprio contexto de execução, promovendo a estabilidade e integridade do sistema base.

Foi também desenvolvido um script automatizado que reside na máquina virtual do [IDS](#). Este script como se pode observar na Figura 14, quando executado, ativa automaticamente o ambiente virtual onde o CICFlowMeter se encontra instalado, inicia a ferramenta e coloca-a a monitorizar continuamente a interface de rede eth0 que, na máquina virtual do [IDS](#) se encontra em modo promíscuo.



```
kali@kali: ~/Desktop
File Actions Edit View Help
(kali@kali)-[~]
$ sudo ip link set eth0 promisc on
[sudo] password for kali:
(kali@kali)-[~]
$ cd Desktop
(kali@kali)-[~/Desktop]
$ sudo ./script.sh
Iniciando cicflowmeter ... Os CSVs serão salvos em /root/cicflowmeter_output
Uma cópia será feita para /home/kali/cicflowmeter_output_user
```

Figura 14 - Definição da Interface Eth0 em Modo Promíscuo e Inicialização do script.sh

Fonte desta figura: Captura de Ecrã do VMware Workstation Pro.

2.5.6 Simulação de Ataques

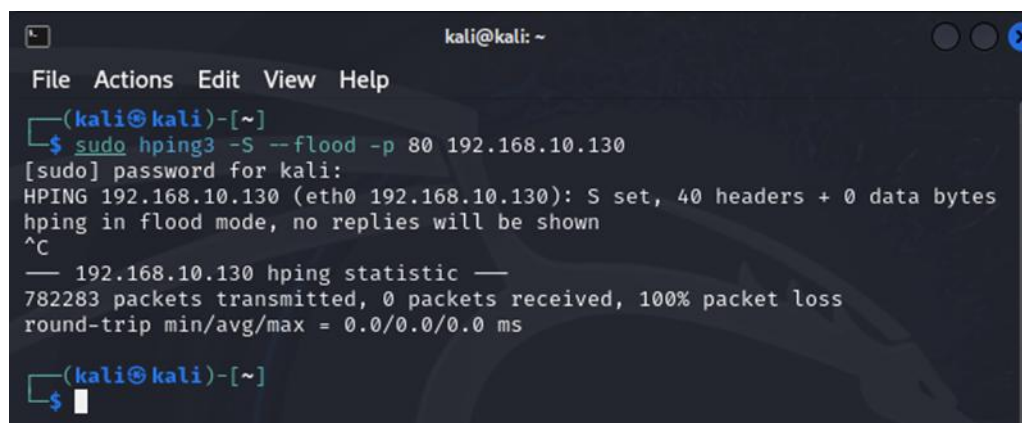
Na Figura 15, observa-se a execução de um ataque de negação de serviço do tipo SYN Flood, realizado a partir da máquina atacante com o endereço [IP](#) 192.168.10.129, tendo como alvo a máquina vítima com o [IP](#) 192.168.10.130. O ataque foi conduzido com recurso à ferramenta hping3, uma conhecida aplicação de linha de comandos utilizada para gerar tráfego personalizado a nível do protocolo Transmission Control Protocol/Internet Protocol ([TCP/IP](#)) com fins de teste, auditoria e simulação de ataques.

O comando utilizado foi o seguinte: `sudo hping3 -S -flood -p 80 192.168.10.130`

Este comando instrui o hping3 a enviar um volume elevado de pacotes [TCP](#) com flag SYN ativado, ou seja, o indicador de início de ligação [TCP](#), destinados à porta 80 da máquina da vítima, em modo flood, ou seja, em envio contínuo e agressivo sem esperar por respostas. Este tipo de ataque visa esgotar os recursos da máquina alvo, ao forçá-la a manter múltiplas ligações semi-abertas (estado SYN-RECEIVED), acabando por comprometer a sua capacidade de resposta legítima.

Durante a execução do ataque, a máquina [IDS](#) presente na mesma sub-rede virtual e com a interface de rede eth0 configurada em modo promíscuo, encontra-se a utilizar a ferramenta CICFlowMeter para capturar todo o tráfego que circula na rede local. Esta ferramenta transforma os pacotes intercetados em ficheiros .csv, contendo as features técnicas relevantes de cada fluxo de rede.

Estes ficheiros .csv gerados representam, de forma estruturada, a atividade maliciosa observada durante o ataque. A informação extraída é posteriormente utilizada pelo [IDS](#) para efeitos de deteção e classificação, permitindo avaliar a eficácia do sistema em identificar comportamentos anómalos associados a ataques de negação de serviço, como é o caso do [TCP SYN Flood](#) aqui representado.



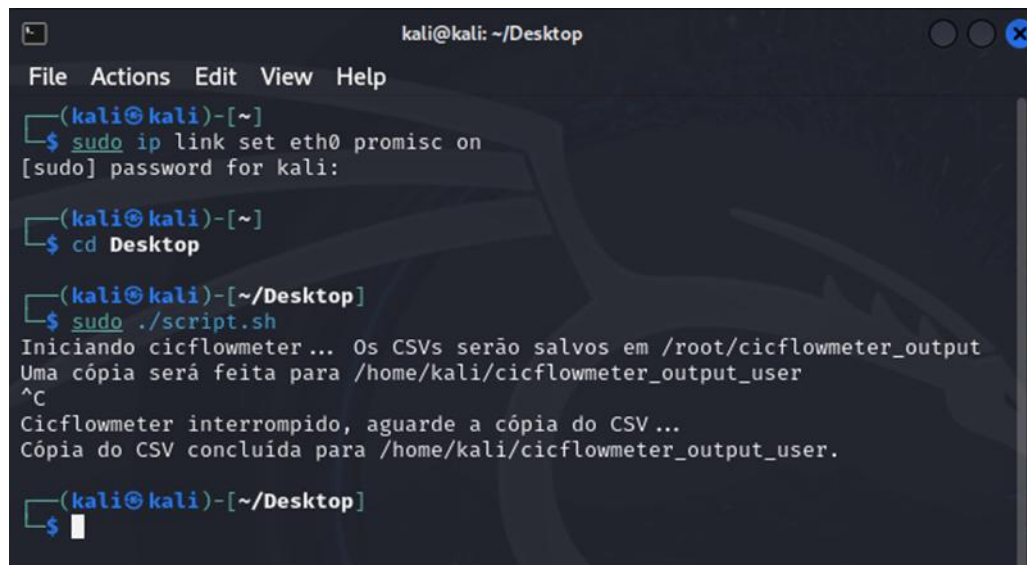
```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
$ sudo hping3 -S --flood -p 80 192.168.10.130  
[sudo] password for kali:  
HPING 192.168.10.130 (eth0 192.168.10.130): S set, 40 headers + 0 data bytes  
hping in flood mode, no replies will be shown  
^C  
— 192.168.10.130 hping statistic —  
782283 packets transmitted, 0 packets received, 100% packet loss  
round-trip min/avg/max = 0.0/0.0/0.0 ms  
(kali@kali)-[~]  
$
```

Figura 15 - Execução de Ataque TCP SYN Flood

Fonte desta figura: Captura de Ecrã do VMware Workstation Pro.

A captura e análise de tráfego decorrem até o utilizador interromper manualmente o processo com a combinação de teclas CTRL + C, como visível na Figura 16. Após a interrupção, os ficheiros .csv gerados são armazenados inicialmente numa pasta de acesso restrito, pertencente ao superutilizador (sudo), por questões de permissões associadas ao processo de captura. No entanto, como o [IDS](#) necessita de aceder aos dados gerados, o script executa um passo final onde copia automaticamente os ficheiros .csv para um diretório com permissões abertas a todos os utilizadores. Esta etapa garante que o [IDS](#) tem acesso ininterrupto

e descomplicado aos dados mais recentes do tráfego, essenciais para a detecção de atividades maliciosas na rede.



```
kali@kali: ~/Desktop
File Actions Edit View Help
(kali@kali)-[~]
$ sudo ip link set eth0 promisc on
[sudo] password for kali:

(kali@kali)-[~]
$ cd Desktop

(kali@kali)-[~/Desktop]
$ sudo ./script.sh
Iniciando cicflowmeter... Os CSVs serão salvos em /root/cicflowmeter_output
Uma cópia será feita para /home/kali/cicflowmeter_output_user
^C
Cicflowmeter interrompido, aguarde a cópia do CSV...
Cópia do CSV concluída para /home/kali/cicflowmeter_output_user.

(kali@kali)-[~/Desktop]
$
```

Figura 16 - Termino do script.sh

Fonte desta figura: Captura de Ecrã do VMware Workstation Pro.

Conforme mencionado anteriormente, o script responsável por executar o CICFlowMeter na máquina virtual do [IDS](#) encontra-se configurado para, após cada sessão de captura de tráfego, copiar os ficheiros gerados no formato .csv para um diretório acessível a todos os utilizadores do sistema. A Figura 17 mostra esse diretório, designado por “Flow_outputs”, onde se encontram armazenados os vários ficheiros de saída resultantes de sessões distintas de monitorização.

Esta abordagem garante que o [IDS](#) tem acesso direto e contínuo aos dados mais recentes do tráfego captado na rede, sem restrições de permissões ou dependência de privilégios de superutilizador. A partir desta pasta, o [IDS](#) pode então proceder à leitura, interpretação e análise dos ficheiros .csv, possibilitando a identificação de comportamentos anómalos e padrões associados a possíveis ameaças ou ataques em curso na rede monitorizada.

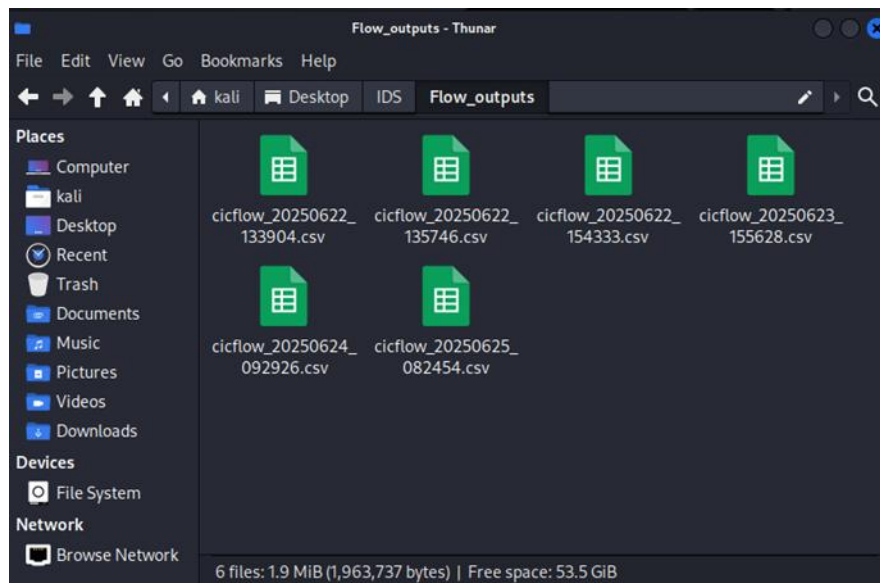


Figura 17 - Pasta onde os Ficheiros .csv são Acedidos pelo IDS

Fonte desta figura: Captura de Ecrã do VMware Workstation Pro.

Na Figura 18 é apresentada a estrutura de diretórios utilizada para organizar o funcionamento do CICFlowMeter na máquina virtual do [IDS](#). Esta estrutura foi concebida de forma a garantir o isolamento funcional da ferramenta, a segurança dos dados gerados e a acessibilidade necessária ao [IDS](#) para análise contínua. Os diretórios tem as seguintes finalidades, o “cicflowmeter_output” é o diretório onde os ficheiros .csv gerados pelo CICFlowMeter são inicialmente gravados durante a execução da captura de tráfego. Como o processo corre com privilégios de superutilizador, os ficheiros criados nesta pasta possuem permissões restritas, inacessíveis a utilizadores comuns por motivos de segurança e integridade dos dados.

“Flow_outputs”, conforme ilustrado anteriormente na Figura 17, esta pasta contém cópias dos ficheiros .csv da pasta anterior, mas com permissões alargadas, permitindo o acesso por qualquer utilizador do sistema. O script de execução do CICFlowMeter automatiza a transferência dos ficheiros para este diretório após cada sessão de captura, garantindo que o [IDS](#) tem sempre acesso aos dados atualizados sem necessitar de privilégios elevados.

“cicflowmeter_venv”, este diretório corresponde ao ambiente virtual Python criado exclusivamente para a instalação e execução do CICFlowMeter. Um ambiente virtual permite isolar dependências e bibliotecas específicas da aplicação do restante sistema operativo, evitando conflitos entre versões de pacotes Python e protegendo a estabilidade de outras aplicações. Neste ambiente estão instaladas todas as bibliotecas necessárias ao funcionamento

correto do CICFlowMeter, assegurando que a ferramenta opera num contexto controlado e independente.

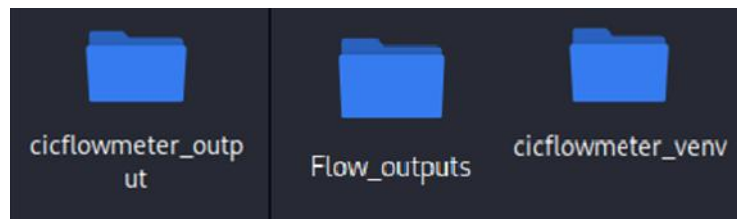


Figura 18 - Estrutura de Diretórios Associados ao CICFlowMeter e ao IDS

Fonte desta figura: Captura de Ecrã do VMware Workstation Pro.

Na Figura 19 observa-se a execução de um comando na máquina virtual com o endereço [IP](#) 192.168.10.130, correspondente à máquina vítima no ambiente de testes. O objetivo desta operação é simular a criação de um ficheiro contendo dados aleatórios.

O comando utilizado foi o seguinte: `dd if=/dev/urandom of=/tmp/segredo.bin bs=1M count=20`

Este comando faz uso da ferramenta “dd”, frequentemente utilizada para operações de baixo nível com dispositivos e ficheiros. Especificamente “if=/dev/urandom” indica que os dados de entrada serão gerados aleatoriamente, com base na fonte pseudoaleatória do sistema.

“of=/tmp/segredo.bin” define o caminho e nome do ficheiro de saída, neste caso um ficheiro binário chamado segredo.bin, armazenado temporariamente em “/tmp”. “bs=1M” define o tamanho do bloco em 1 megabyte, por fim, “count=20” especifica que serão escritos 20 blocos de 1MB, totalizando 20 megabytes de dados aleatórios.

A execução deste comando visa simular a criação de ficheiros sensíveis ou confidenciais.

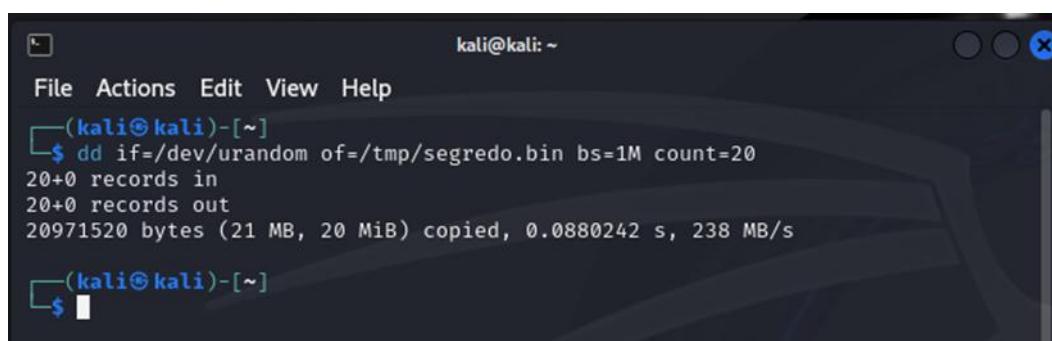


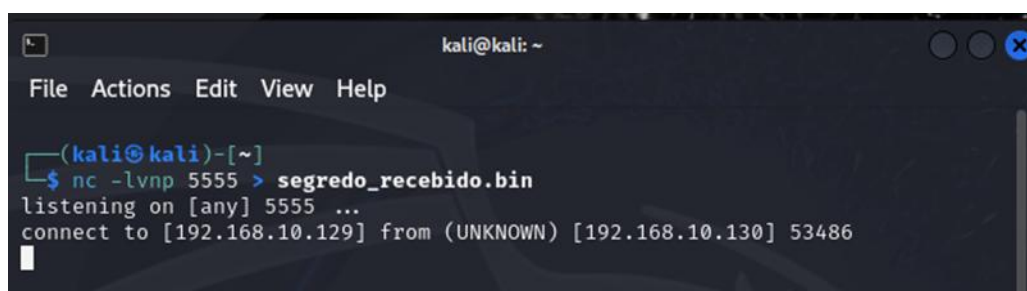
Figura 19 - Geração de um Ficheiro Binário na Máquina Vítima

Fonte desta figura: Captura de Ecrã do VMware Workstation Pro.

Na Figura 20 é exibida a linha de comandos da máquina atacante, que tem o endereço [IP](#) 192.168.10.129, a qual se encontra à escuta numa porta [TCP](#) para receber dados provenientes da máquina vítima 192.168.10.130. O objetivo deste procedimento é simular um ataque de exfiltração de informação sensível, neste caso, o ficheiro binário `segredo.bin` gerado na máquina vítima como anteriormente demonstrado na Figura 19.

O comando executado foi o seguinte: `nc -lvnp 5555 > segredo_recebido.bin`

Este comando utiliza a ferramenta Netcat (“nc”). “-l” indica que o Netcat deve operar em modo de escuta, “-v” ativa o modo verbose, que fornece detalhes sobre a ligação estabelecida, “-n” força o uso de endereços [IP](#) numéricos, ignorando a resolução de nomes, “-p 5555” define a porta [TCP](#) 5555 na qual a máquina ficará à escuta de ligações, por fim “> segredo_recebido.bin” redireciona os dados recebidos diretamente para um ficheiro com esse nome.



```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
$ nc -lvnp 5555 > segredo_recebido.bin  
listening on [any] 5555 ...  
connect to [192.168.10.129] from (UNKNOWN) [192.168.10.130] 53486
```

Figura 20 - Preparação da Máquina Atacante para Receção do Ficheiro via Netcat

Fonte desta figura: Captura de Ecrã do VMware Workstation Pro.

Este comando, apresenta na Figura 21, foi executado a partir da máquina vítima 192.168.10.130. Deve apenas ser executado após a máquina atacante 192.168.10.129 já se encontrar à escuta na porta [TCP](#) 5555.

O “nc” invoca o utilitário Netcat, utilizado para estabelecer ligações [TCP](#) diretas entre dois sistemas, 192.168.10.129 é o endereço [IP](#) da máquina atacante, ou seja, o destino da ligação, 5555 é a porta [TCP](#) onde a máquina atacante está à escuta, aguardando dados e por fim “< /tmp/segredo.bin” redireciona o conteúdo do ficheiro binário `segredo.bin`, localizado no diretório temporário da máquina vítima.

Este comando estabelece uma ligação ativa ao sistema atacante e transfere silenciosamente o conteúdo do ficheiro através de uma conexão [TCP](#), sem encriptação ou autenticação. O ficheiro é então recebido e gravado na máquina atacante, tal é possível de ser

observado na Figura 20, onde aparece, no final, a informação de que foi efetivamente estabelecida uma ligação entre as duas máquinas e o ficheiro foi transferido silenciosamente.

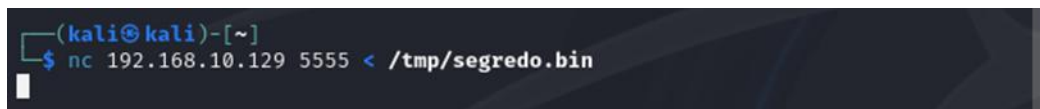


Figura 21 - Envio do Ficheiro da Vítima para o Atacante via Netcat

Fonte desta figura: Captura de Ecrã do VMware Workstation Pro.

Na Figura 22 é possível observar a execução do comando “ls -lh” nas duas máquinas envolvidas no processo de exfiltração de dados, com o intuito de comparar os tamanhos dos ficheiros e assim confirmar que a transferência foi realizada com sucesso e na sua totalidade.

Na parte inferior da imagem, referente à máquina vítima 192.168.10.130, é verificado o ficheiro original segredo.bin, armazenado no diretório temporário “/tmp/”, com um tamanho de 20 MB. Já na parte superior, correspondente à máquina atacante 192.168.10.129, é exibido o ficheiro segredo_recebido.bin, que foi recebido através da conexão [TCP](#) estabelecida via Netcat. Também este ficheiro apresenta exatamente o mesmo tamanho 20 MB.

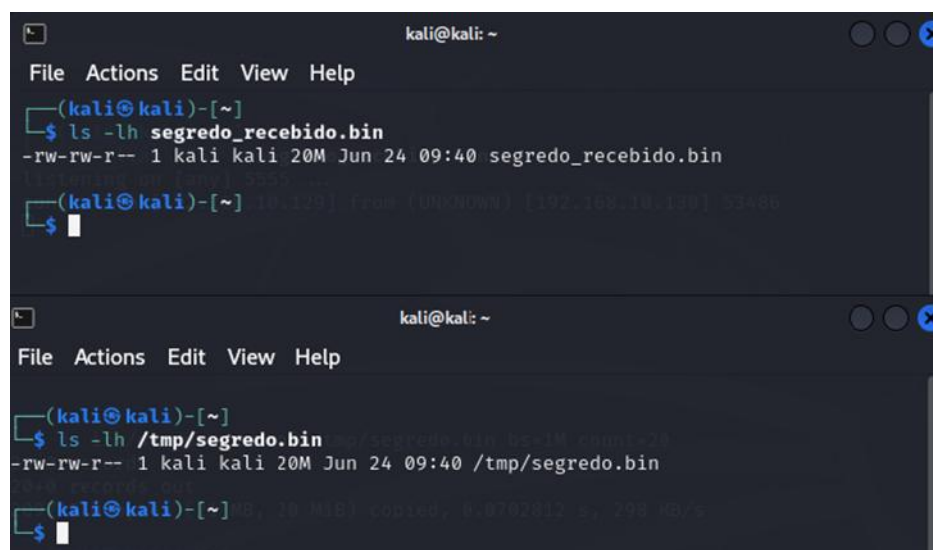


Figura 22 - Comparação do Tamanho dos Ficheiros

Fonte desta figura: Captura de Ecrã do VMware Workstation Pro.

2.5.7 Execução Automatizada do IDS

Foi criado um script de automatização em bash, responsável por preparar o ambiente necessário e lançar o processo de análise. O objetivo principal deste script é garantir que o ficheiro ids.py, seja executado dentro de um ambiente virtual Python isolado, com todas as dependências corretamente configuradas.

O script verifica inicialmente a existência de um ambiente virtual em “~/venvs/ids-env”. Caso este não exista, o script procede à sua criação utilizando o interpretador python 3.10 e, em seguida, instala localmente os pacotes essenciais ao funcionamento do [IDS](#), nomeadamente: numpy, pandas, joblib, tensorflow e colorama. A utilização do ambiente virtual é fundamental para isolar as dependências do projeto e evitar conflitos com outras aplicações ou bibliotecas instaladas globalmente no sistema Kali, ao contrário do sistema Windows onde a utilização da ferramenta pip seria o suficiente para instalar todas as dependências.

Depois de confirmar a existência, ou criar, o ambiente, o script altera o diretório de trabalho para “~/Desktop/IDS”, garantindo que todos os caminhos relativos utilizados pelo ficheiro ids.py funcionem corretamente. Em seguida, executa o script Python de forma silenciosa, com os avisos do TensorFlow suprimidos para evitar ruído desnecessário na consola.

Para comodidade do utilizador, o script foi copiado para o diretório “~/.local/bin/” com o nome “runids”, e foi criado um “alias” permanente que permite a sua invocação de forma simples a partir de qualquer diretório do terminal. Assim, basta digitar o comando: `runids`

Após a criação do script, foi necessário atribuir-lhe permissões de execução através do seguinte comando: `chmod +x ~/.local/bin/runids`

Sem esta permissão, o sistema não permite a sua execução.

Com as três máquinas virtuais em execução e a máquina virtual [IDS](#) com a interface eth0 em modo promíscuo, todas ligadas à VMnet2, pode-se iniciar o script do CICFlowMeter e o “runids” na máquina virtual do [IDS](#).

Para realizar um teste de funcionamento do [IDS](#) é necessário, com a máquina atacante, realizar um ataque direcionado à máquina vítima, no fim do ataque, na máquina [IDS](#), é necessário realizar a paragem da execução do script do CICFlowMeter de modo a permitir que o ficheiro .csv seja escrito e guardado na pasta “Flow_outputs” que o nosso [IDS](#) está constantemente a monitorizar, depois de analisado o .csv pelo nosso [IDS](#), se o ficheiro contiver tráfego válido, o [IDS](#) irá processá-lo e classificar o conteúdo que passou na rede à instantes.

O [IDS](#) permanecerá ativo e em escuta de novas entradas na pasta “Flow_outputs” até que a sua execução seja manualmente interrompida com CTRL + C.

3 Avaliações e Resultados

Nesta secção são apresentados os resultados obtidos ao longo do desenvolvimento e avaliação do sistema [IDS](#).

São analisados os dados que serviram de base ao treino e teste, incluindo a sua preparação, estrutura, estatísticas descritivas e correlações entre atributos. São também descritas as métricas de avaliação utilizadas na análise do desempenho dos modelos, bem como os resultados dos modelos de [ML](#) desenvolvidos. E ainda apresentados os resultados dos testes preliminares realizados com os modelos finais e os resultados obtidos durante a execução prática do [IDS](#) no ambiente de testes.

3.1 Resultados Obtidos com o Dataset

Nesta secção são apresentados os resultados da análise exploratória e preparação do dataset utilizado. São descritas as principais características do conjunto de dados final, incluindo a dimensão, a distribuição das classes, as estatísticas descritivas dos atributos e a correlação entre os atributos com maior variabilidade.

Os dados apresentados visam fornecer uma visão geral da estrutura e qualidade do dataset, permitindo compreender a base sobre a qual os modelos foram treinados e avaliados.

3.1.1 Dimensão e Estrutura do Dataset

O dataset final preparado para treino e teste dos modelos contém um total de 140.087 registos e 78 atributos. Os tipos de dados atribuídos aos atributos foram definidos de forma otimizada, combinando float64 para as variáveis contínuas e int64 para as variáveis discretas, assegurando a consistência do processamento e o uso eficiente da memória.

Adicionalmente, foi verificada a existência de valores nulos no dataset, tendo-se confirmado que, após a fase de limpeza dos dados, nenhuma coluna apresentava valores em falta.

3.1.2 Estatísticas Descritivas dos Atributos

Foram calculadas estatísticas descritivas para os atributos numéricos do dataset, com o objetivo de caracterizar a distribuição dos dados e apoiar o desenvolvimento dos modelos. Os indicadores obtidos incluem a média, desvio padrão, valor mínimo, 1.º quartil, mediana, 3.º quartil, valor máximo, assimetria e curtose.

Como exemplo, o atributo Dst Port apresentou uma média de 5.810,90, um desvio padrão de 14.620,53 e um valor máximo de 65.526, evidenciando elevada dispersão. O atributo Flow Duration registou valores extremos, com uma média de -5.249.193,17, desvio padrão de 3.447.398.000 e assimetria de -219,85, refletindo a presença de valores anómalos e distribuição

altamente assimétrica. Já o atributo Tot Fwd Pkts apresentou uma mediana de 2 e um valor máximo de 32.662, com assimetria de -1,89, indicando distribuição enviesada à esquerda. Por sua vez, o Tot Bwd Pkts revelou uma curtose de 6.500,85, reforçando a presença de valores extremos.

3.1.3 Distribuição das Classes

Foi também analisada a distribuição das classes, com o objetivo de confirmar o balanceamento das amostras e garantir a representatividade de cada tipo de tráfego no treino e teste dos modelos.

A Figura 23 apresenta o gráfico de barras que ilustra a distribuição das classes no dataset. É possível observar que o tráfego benigno representa a maioria dos registros, com um total de 100.000 fluxos, o que corresponde a cerca de 71% do conjunto de dados. As classes de ataque BruteForce, DDoS, DoS e Bot encontram-se equilibradas entre si, com 10.000 fluxos cada, o que corresponde a cerca de 7% por classe.

Destaca-se ainda a classe SQLInjection, que, de acordo com a quantidade real de amostras disponíveis no dataset original, apresenta apenas 87 fluxos, representando menos de 0,1% do total.

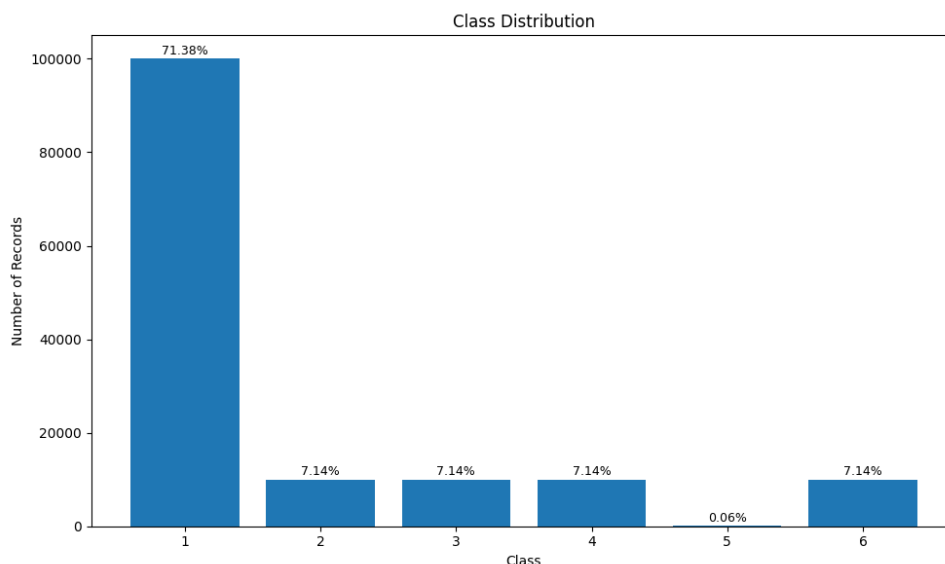


Figura 23 - Distribuição de Classes do Dataset

Fonte desta figura: Resultado da Análise Exploratória ao Dataset.

3.1.4 Correlação Entre Atributos

A Figura 24 apresenta a matriz de correlação dos 15 atributos com maior variância no dataset, representada sob a forma de heatmap.

É possível observar na matriz a presença de correlações muito fortes (próximas de 1) entre diversos atributos relacionados com a temporização de fluxos, como Flow IAT Max, Idle Max, Flow IAT Std e Idle Mean, que atingem valores de correlação superiores a 0,99. Estes atributos mostram padrões de comportamento muito semelhantes no contexto dos dados analisados.

Por outro lado, verifica-se que atributos como Fwd IAT Tot, Bwd IAT Tot e Fwd IAT Mean apresentam correlações fracas ou nulas com grande parte dos restantes atributos, indicando maior independência e potencial utilidade no enriquecimento do modelo com informação menos redundante.

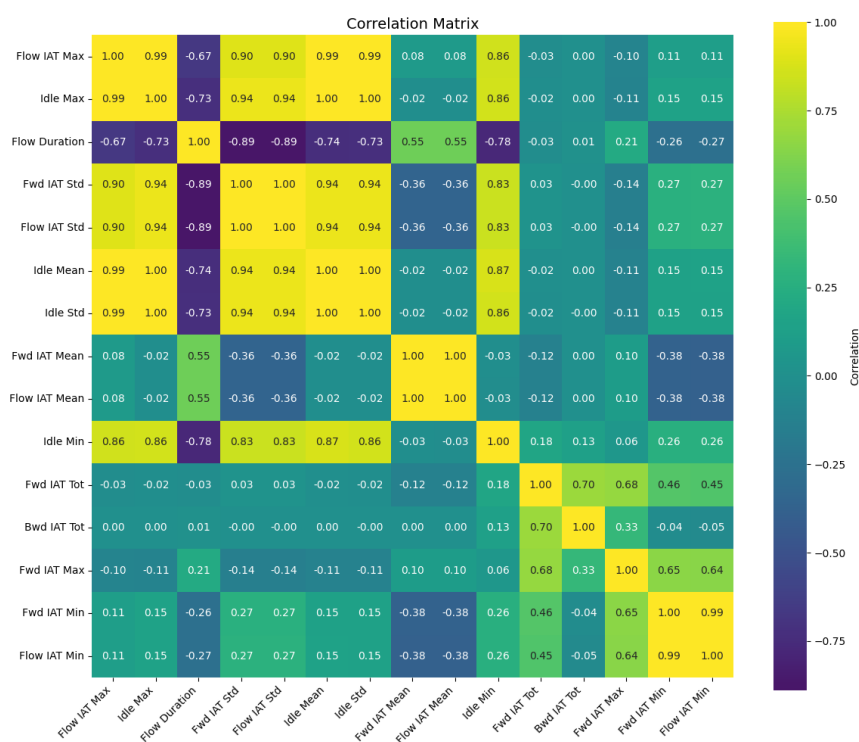


Figura 24 - Matriz de Correlação do Dataset

Fonte desta figura: Resultado da Análise Exploratória ao Dataset.

3.2 Métricas de Avaliação Utilizadas

Para avaliar o desempenho dos modelos de [ML](#) desenvolvidos para o [IDS](#), foram utilizadas as seguintes métricas de avaliação:

1. Precisão (Precision):

Mede a proporção de previsões corretas entre todas as previsões positivas feitas pelo modelo. É particularmente importante para avaliar a exatidão na detecção de ataques. A fórmula (1) é:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (1)$$

onde:

- True Positives ([TP](#)): Número de amostras corretamente classificadas como positivas (ataques corretamente detetados).
- False Positives ([FP](#)): Número de amostras incorretamente classificadas como positivas (tráfego normal incorretamente identificado como ataque).

2. Recall:

Mede a proporção de amostras positivas corretamente identificadas pelo modelo. Esta métrica (2) é crítica para garantir que o modelo não deixa passar ataques sem os detetar. É calculada como:

$$\text{Recall} = \frac{TP}{TP + FN} \quad (2)$$

onde:

- False Negatives ([FN](#)): Número de amostras incorretamente classificadas como negativas (ataques não detetados).

3. F1-Score:

Média harmónica entre Precisão e Recall, proporcionando uma medida equilibrada do desempenho, especialmente útil em problemas com classes desbalanceadas. A fórmula (3) é:

$$\text{F1 - Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3)$$

4. Matriz de Confusão:

Uma representação visual da distribuição das previsões do modelo em relação às classes reais (benigno e ataque). Esta matriz apresenta os seguintes valores:

- True Positives ([TP](#)): Ataques corretamente identificados.
- True Negatives ([TN](#)): Tráfego normal corretamente identificado.
- False Positives ([FP](#)): Tráfego normal incorretamente identificado como ataque.
- False Negatives ([FN](#)): Ataques não detetados.

As matrizes de confusão foram geradas para todos os modelos, facilitando a visualização dos padrões de erro.

5. Erro Médio Quadrático e Curva Precision-Recall:

No contexto do modelo Autoencoder, foram utilizadas duas métricas que trabalham em conjunto para otimizar o desempenho do modelo na detecção de anomalias: o [MSE](#) e a Curva Precision-Recall.

O [MSE](#) (4) é utilizado para medir o erro de reconstrução de cada amostra, comparando os valores reais com os valores reconstruídos pelo Autoencoder. É calculado como:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{x}_i)^2 \quad (4)$$

- Um baixo [MSE](#) indica que a amostra foi bem reconstruída (tráfego normal).
- Um alto [MSE](#) indica uma má reconstrução (anomalia).

No Autoencoder, o [MSE](#) é calculado para cada amostra do conjunto de teste, gerando uma distribuição de erros.

Com base no [MSE](#) de cada amostra, é gerada uma Curva Precision-Recall, que permite avaliar a relação entre Precisão (1) e Recall (2) em diferentes limiares (thresholds) de [MSE](#). A partir destes valores, é calculado o F1-Score (3) para cada limiar.

O limiar ótimo é definido como o valor de [MSE](#) que maximiza o F1-Score na Curva Precision-Recall, garantindo um equilíbrio ideal entre Precisão e Recall.

6. Validação Cruzada (Cross-Validation):

Utilizada para os modelos supervisionados ([MLP](#), [RF](#) e [SVM](#)). Foi aplicada com 5 e 10 folds, garantindo uma avaliação mais robusta e estável do desempenho dos modelos. As seguintes métricas foram calculadas em cada iteração da validação cruzada:

- Precisão (Precision)
- Recall
- F1-Score

7. F1-Score por Fold:

Nos modelos supervisionados com validação cruzada, foi gerado um gráfico que apresenta o F1-Score em cada fold, permitindo avaliar a consistência do modelo ao longo das diferentes partições de treino e teste.

Estas métricas foram selecionadas para oferecer uma visão abrangente e precisa do desempenho dos modelos, garantindo uma avaliação clara da sua capacidade de distinguir entre tráfego normal e tráfego anômalo.

8. Out-of-Bag Score:

Específico do modelo [RF](#) o [OOB](#)-score é uma métrica de validação interna que permite avaliar o desempenho do modelo sem necessidade de um conjunto de validação separado. Durante o processo de treino, cada árvore da floresta é construída com uma amostra aleatória dos dados, com reposição. Cerca de 1/3 dos dados não é utilizado no treino de cada árvore, sendo designado por conjunto “out-of-bag”.

O [OOB](#)-score é calculado avaliando o desempenho do modelo nesses dados não vistos por cada árvore, funcionando como uma forma eficiente e integrada de validação cruzada. Este valor representa a precisão média obtida nas amostras [OOB](#) e permite obter uma estimativa do desempenho generalizado do modelo.

Nos subcapítulos seguintes, são apresentados os resultados obtidos para cada modelo, com base nestas métricas.

9. Exatidão (Accuracy):

A exatidão (5) mede a proporção de previsões corretas (tanto positivas como negativas) em relação ao total de previsões efetuadas. Esta métrica fornece uma visão geral da eficácia global do modelo na classificação das amostras.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \quad (5)$$

onde:

- [TP](#): número de amostras corretamente classificadas como positivas;
- [TN](#): número de amostras corretamente classificadas como negativas;
- [FP](#): número de amostras incorretamente classificadas como positivas;
- [FN](#): número de amostras incorretamente classificadas como negativas.

10. Média Macro (Macro Average):

A Macro Average (6) calcula a média simples da métrica (precisão, recall ou f1-score) entre todas as classes, atribuindo o mesmo peso a cada uma, independentemente do seu número de amostras. Esta abordagem é útil quando se pretende avaliar o desempenho do modelo de forma equilibrada entre classes, mesmo que estejam desbalanceadas. A fórmula é:

$$\text{Macro Average} = \frac{1}{C} \sum_{i=1}^C \text{Metric}_i \quad (6)$$

onde:

- C é o número total de classes;
- Metric_i é o valor da métrica (precisão, recall ou f1-score) para a classe i;

11. Média Ponderada (Weighted Average):

A Weighted Average (7) considera o número de amostras de cada classe ao calcular a média das métricas, atribuindo maior peso às classes com mais exemplos. Esta abordagem reflete o desempenho global do modelo tendo em conta a distribuição real das classes. A fórmula é:

$$\text{Weighted Average} = \sum_{i=1}^C \left(\frac{n_i}{N} \cdot \text{Metric}_i \right) \quad (7)$$

onde:

- n_i é o número de amostras da classe i;
- N é o número total de amostras;
- Metric_i é o valor da métrica (precisão, recall ou f1-score) para a classe i;

3.3 Resultados Obtidos com os Modelos de Machine Learning

Nesta secção são apresentados os resultados obtidos com os diferentes modelos de [ML](#) desenvolvidos, tanto supervisionados como não supervisionados. Para cada modelo, são analisadas as métricas de avaliação descritas na secção anterior, nomeadamente: precisão, Recall, F1-score, matriz de confusão, erro médio quadrático (no caso do [AE](#)) e [OOB](#)-score (no caso da [RF](#)).

A avaliação é organizada por modelo, sendo detalhados os testes realizados e os respetivos desempenhos. Esta análise visa comparar o comportamento de cada modelo perante

os dados presentes no dataset, identificando as suas vantagens, limitações e adequação prática ao contexto de detecção de anomalias.

3.3.1 Modelos Supervisionados

3.3.1.1 Rede Neuronal Multi-Layer Perceptron

3.3.1.1.1 Primeiro Teste

No primeiro teste com o modelo [MLP](#), foram obtidos resultados positivos, especialmente nas classes com maior número de amostras.

A precisão e o recall foram de 1.00 nas classes 1 a 4, refletindo uma capacidade excelente do modelo para identificar corretamente estas categorias. Também a classe 6 atingiu métricas praticamente perfeitas, com uma precisão de 0.99 e recall de 1.00. No entanto, a classe 5, que apresenta apenas 26 amostras, revelou um desempenho significativamente inferior: precisão de 0.92 e recall de apenas 0.46, indicando uma elevada taxa de falsos negativos.

A matriz de confusão mostra que, na classe 5, apenas 12 amostras foram corretamente classificadas, enquanto 14 foram incorretamente rotuladas como pertencentes a outras classes. Apesar desta limitação, a accuracy global foi de 1.00, com uma média ponderada do F1-score também de 1.00, o que evidencia o domínio das classes maioritárias na performance global.

A Figura 25 apresenta o relatório de classificação e a matriz de confusão correspondentes.

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 1 | 1.00 | 1.00 | 1.00 | 30001 |
| 2 | 1.00 | 1.00 | 1.00 | 3000 |
| 3 | 1.00 | 1.00 | 1.00 | 3000 |
| 4 | 1.00 | 1.00 | 1.00 | 3000 |
| 5 | 0.92 | 0.46 | 0.62 | 26 |
| 6 | 0.99 | 1.00 | 1.00 | 3000 |
| accuracy | | | 1.00 | 42027 |
| macro avg | 0.99 | 0.91 | 0.94 | 42027 |
| weighted avg | 1.00 | 1.00 | 1.00 | 42027 |

Matriz de Confusão:

```
[[29978  0  0  6  1 16]
 [  0 3000  0  0  0  0]
 [  0  0 3000  0  0  0]
 [  0  0  0 3000  0  0]
 [ 14  0  0  0 12  0]
 [  1  0  0  0  0 2999]]
```

Figura 25 - Relatório de Classificação e Matriz de Confusão do Teste 1 MLP

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.1.1.2 Segundo Teste

No segundo teste, o modelo [MLP](#) voltou a demonstrar uma performance excelente, com melhorias claras em relação ao primeiro teste, sobretudo na classe 5, que anteriormente registava um recall muito baixo.

Tal como no teste anterior, as classes 1 a 4 e a classe 6 mantiveram resultados perfeitos, com precisão, recall e F1-score de 1.00. A classe 5, com apenas 26 amostras, evidenciou uma evolução significativa: o recall subiu para 0.65 e o F1-score aumentou para 0.79, demonstrando que o modelo foi mais eficaz na deteção de instâncias raras.

A matriz de confusão mostra uma redução do número de falsos negativos nesta classe (9 erros), o que contribuiu para um F1-score médio ponderado de 1.00 e uma média macro de 0.96. Estes resultados indicam uma melhoria na generalização do modelo mesmo perante dados desbalanceados.

A Figura 26 apresenta o relatório de classificação e a respetiva matriz de confusão.

| | precision | recall | f1-score | support |
|---------------------|-----------|--------|----------|---------|
| 1 | 1.00 | 1.00 | 1.00 | 30001 |
| 2 | 1.00 | 1.00 | 1.00 | 3000 |
| 3 | 1.00 | 1.00 | 1.00 | 3000 |
| 4 | 1.00 | 1.00 | 1.00 | 3000 |
| 5 | 1.00 | 0.65 | 0.79 | 26 |
| 6 | 1.00 | 1.00 | 1.00 | 3000 |
| accuracy | | | 1.00 | 42027 |
| macro avg | 1.00 | 0.94 | 0.96 | 42027 |
| weighted avg | 1.00 | 1.00 | 1.00 | 42027 |
| Matriz de Confusão: | | | | |
| [[29993 0 0 3 0 5] | | | | |
| [0 3000 0 0 0 0] | | | | |
| [0 0 3000 0 0 0] | | | | |
| [0 0 0 3000 0 0] | | | | |
| [9 0 0 0 17 0] | | | | |
| [1 0 0 0 0 2999]] | | | | |

Figura 26 - Relatório de Classificação e Matriz de Confusão do Teste 2 MLP

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.1.1.3 Terceiro Teste

O terceiro teste manteve a tendência de resultados altamente positivos para a maioria das classes, com todas as métricas a indicarem uma generalização muito eficaz. A exceção continua a ser a classe 5, que, embora apresente uma quantidade muito reduzida de amostras (26), volta a revelar fragilidades na deteção.

Todas as classes principais (1 a 4 e 6) mantiveram precisão e recall de 1.00, à exceção da classe 5, que obteve um recall de 0.54 e um F1-score de 0.70.

A média macro do F1-score foi de 0.95, enquanto a média ponderada permaneceu em 1.00, o que reflete a estabilidade geral do modelo. A matriz de confusão mostra que houve apenas alguns falsos negativos nas classes minoritárias, sem impacto significativo nas classes principais.

A Figura 27 apresenta o relatório de classificação e a matriz de confusão correspondentes.

| | precision | recall | f1-score | support |
|---------------------|-----------|--------|----------|---------|
| 1 | 1.00 | 1.00 | 1.00 | 30001 |
| 2 | 1.00 | 1.00 | 1.00 | 3000 |
| 3 | 1.00 | 1.00 | 1.00 | 3000 |
| 4 | 1.00 | 1.00 | 1.00 | 3000 |
| 5 | 1.00 | 0.54 | 0.70 | 26 |
| 6 | 1.00 | 1.00 | 1.00 | 3000 |
| accuracy | | | 1.00 | 42027 |
| macro avg | 1.00 | 0.92 | 0.95 | 42027 |
| weighted avg | 1.00 | 1.00 | 1.00 | 42027 |
| Matriz de Confusão: | | | | |
| [[29984 1 3 5 0 8] | | | | |
| [0 3000 0 0 0 0] | | | | |
| [0 0 3000 0 0 0] | | | | |
| [0 0 0 3000 0 0] | | | | |
| [12 0 0 0 14 0] | | | | |
| [4 0 0 0 0 2996]] | | | | |

Figura 27 - Relatório de Classificação e Matriz de Confusão do Teste 3 MLP

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.1.1.4 Quarto Teste

No quarto teste, os resultados mantiveram-se bastante consistentes em relação aos testes anteriores. As classes principais voltaram a obter valores perfeitos em todas as métricas (precisão, recall e F1-score), enquanto a classe 5 permaneceu como a mais desafiante, com um recall de 0.54 e um F1-score de 0.70, tal como verificado no teste anterior.

As médias globais continuam a refletir um desempenho sólido: o F1-score macro foi de 0.95 e o F1-score ponderado de 1.00, revelando que o modelo lida muito bem com a maior parte das classes, mesmo em cenários com desbalanceamento severo.

A matriz de confusão evidencia uma pequena quantidade de falsos negativos nas classes 5 e 6, sendo o impacto geral mínimo no desempenho do classificador.

A Figura 28 mostra o relatório de classificação e a respetiva matriz de confusão.

| | precision | recall | f1-score | support |
|---------------------|-----------|--------|----------|---------|
| 1 | 1.00 | 1.00 | 1.00 | 30001 |
| 2 | 1.00 | 1.00 | 1.00 | 3000 |
| 3 | 1.00 | 1.00 | 1.00 | 3000 |
| 4 | 1.00 | 1.00 | 1.00 | 3000 |
| 5 | 1.00 | 0.54 | 0.70 | 26 |
| 6 | 1.00 | 1.00 | 1.00 | 3000 |
| accuracy | | | 1.00 | 42027 |
| macro avg | 1.00 | 0.92 | 0.95 | 42027 |
| weighted avg | 1.00 | 1.00 | 1.00 | 42027 |
| Matriz de Confusão: | | | | |
| [[29991 0 0 3 0 7] | | | | |
| [0 3000 0 0 0 0] | | | | |
| [0 0 3000 0 0 0] | | | | |
| [0 0 0 3000 0 0] | | | | |
| [12 0 0 0 14 0] | | | | |
| [1 0 0 0 0 2999]] | | | | |

Figura 28 - Relatório de Classificação e Matriz de Confusão do Teste 4 MLP

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.1.1.5 Modelo Final

O modelo final [MLP](#), proveniente do Segundo Teste, apresentou um excelente desempenho na generalidade das classes. As classes com maior número de amostras (classes 1 a 4 e 6) atingiram valores perfeitos de precisão, recall e F1-score, todos iguais a 1.00. A classe 5, apesar do reduzido número de exemplos (apenas 26), obteve uma precisão de 1.00, mas um recall de 0.65, refletindo uma taxa de falsos negativos ainda significativa. O F1-score desta classe foi de 0.79.

A matriz de confusão mostra que, na classe 5, apenas 17 das 26 amostras foram corretamente classificadas, enquanto 9 foram atribuídas incorretamente a outras classes, todas para a classe 1. No entanto, o impacto deste desvio na performance global foi reduzido, dado o forte domínio numérico das restantes classes.

A accuracy total alcançada foi de 1.00, com uma média ponderada do F1-score também de 1.00. A média macro foi de 0.96, refletindo a ligeira penalização causada pelo desempenho inferior na classe minoritária.

A Figura 29 apresenta a matriz de confusão obtida para o modelo final, enquanto a Figura 30 ilustra os resultados da validação cruzada com 10 folds sobre o F1-score ponderado. Os valores mantiveram-se consistentemente elevados em quase todos os folds, com uma ligeira descida apenas no último, o que demonstra a estabilidade do modelo.

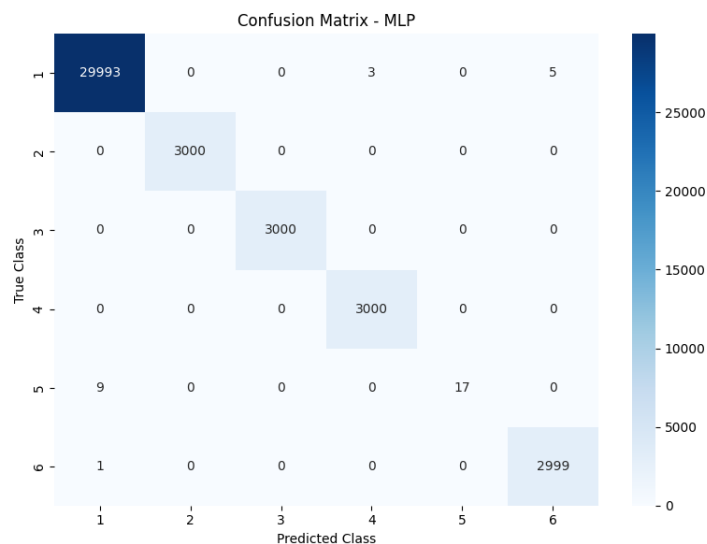


Figura 29 - Matriz de Confusão do Modelo Final MLP

Fonte desta figura: Resultados dos Testes a cada Modelo.

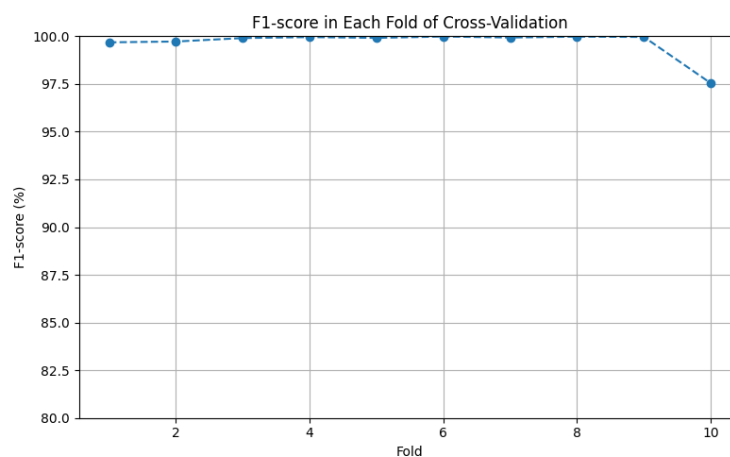


Figura 30 - F1-score em cada Fold na Validação Cruzada do Modelo Final MLP

Fonte desta figura: Resultados dos Testes a cada Modelo.

Na validação cruzada com 5 folds, os valores médios foram também elevados: 98.57% de precisão, 98.47% de recall e 98.01% de F1-score, todos com desvios padrão baixos, reforçando a robustez do modelo final em diferentes subconjuntos de dados.

3.3.1.2 Árvore de Decisão Random Forest

3.3.1.2.1 Primeiro Teste

No primeiro teste com o modelo [RF](#), os resultados obtidos foram bastante positivos, especialmente nas classes com maior representatividade no conjunto de dados. As classes 1 a 4

e 6 apresentaram valores perfeitos de precisão, recall e F1-score, todos iguais a 1.00, evidenciando uma excelente capacidade de classificação por parte do modelo.

A classe 5, composta por apenas 26 amostras, registou uma precisão de 0.95 e um recall de 0.73, o que corresponde a um F1-score de 0.83. Estes valores indicam uma taxa moderada de falsos negativos nesta classe minoritária. A matriz de confusão mostra que, das 26 amostras da classe 5, 19 foram corretamente classificadas, enquanto 7 foram incorretamente atribuídas a outras classes, nomeadamente à classe 1.

Apesar desta limitação pontual, o desempenho global foi excelente, com uma accuracy de 1.00, um F1-score médio ponderado de 1.00 e uma média macro de 0.97, penalizada apenas pelo desempenho ligeiramente inferior na classe 5.

A Figura 31 apresenta a matriz de confusão e os principais indicadores de desempenho obtidos neste primeiro teste.

| | precision | recall | f1-score | support |
|---------------------|-----------|--------|----------|---------|
| 1 | 1.00 | 1.00 | 1.00 | 30001 |
| 2 | 1.00 | 1.00 | 1.00 | 3000 |
| 3 | 1.00 | 1.00 | 1.00 | 3000 |
| 4 | 1.00 | 1.00 | 1.00 | 3000 |
| 5 | 0.95 | 0.73 | 0.83 | 26 |
| 6 | 1.00 | 1.00 | 1.00 | 3000 |
| accuracy | | | 1.00 | 42027 |
| macro avg | 0.99 | 0.95 | 0.97 | 42027 |
| weighted avg | 1.00 | 1.00 | 1.00 | 42027 |
| Matriz de Confusão: | | | | |
| [[29998 0 0 1 1 1] | | | | |
| [0 3000 0 0 0 0] | | | | |
| [2 0 2998 0 0 0] | | | | |
| [0 0 0 3000 0 0] | | | | |
| [7 0 0 0 19 0] | | | | |
| [3 0 0 0 0 2997]] | | | | |

Figura 31 - Relatório de Classificação e Matriz de Confusão do Teste 1 RF

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.1.2.2 Segundo Teste

No segundo teste, o modelo voltou a demonstrar um desempenho global de excelência. As classes com maior volume de amostras (1 a 4 e 6) atingiram valores perfeitos em todas as métricas de avaliação, com precisão, recall e F1-score iguais a 1.00.

A classe 5, ainda que minoritária com apenas 26 exemplos, apresentou uma melhoria significativa face ao teste anterior. A precisão manteve-se em 1.00, enquanto o recall subiu para 0.81 e o F1-score para 0.89, evidenciando uma redução na taxa de falsos negativos. A matriz de confusão mostra que, nesta classe, 21 amostras foram corretamente classificadas, com apenas 5 classificações incorretas, todas atribuídas à classe 1.

A accuracy total foi de 1.00, tal como o F1-score médio ponderado. A média macro das métricas também atingiu valores muito elevados, com destaque para o F1-score de 0.98. Adicionalmente, o modelo obteve um [OOB](#)-score de 0.9996, o que reforça a sua robustez.

A Figura 32 apresenta a matriz de confusão e o resumo das métricas de desempenho associadas a este segundo teste.

| | precision | recall | f1-score | support |
|---------------------|-----------|--------|----------|---------|
| 1 | 1.00 | 1.00 | 1.00 | 30001 |
| 2 | 1.00 | 1.00 | 1.00 | 3000 |
| 3 | 1.00 | 1.00 | 1.00 | 3000 |
| 4 | 1.00 | 1.00 | 1.00 | 3000 |
| 5 | 1.00 | 0.81 | 0.89 | 26 |
| 6 | 1.00 | 1.00 | 1.00 | 3000 |
| accuracy | | | 1.00 | 42027 |
| macro avg | 1.00 | 0.97 | 0.98 | 42027 |
| weighted avg | 1.00 | 1.00 | 1.00 | 42027 |
| Matriz de Confusão: | | | | |
| [[29999 0 0 1 0 1] | | | | |
| [0 3000 0 0 0 0] | | | | |
| [4 0 2996 0 0 0] | | | | |
| [0 0 0 3000 0 0] | | | | |
| [5 0 0 0 21 0] | | | | |
| [2 0 0 0 0 2998]] | | | | |
| OOB-score: 0.9996 | | | | |

Figura 32 - Relatório de Classificação e Matriz de Confusão do Teste 2 RF

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.1.2.3 Terceiro Teste

No terceiro teste, o modelo atingiu o seu melhor desempenho até ao momento. As classes 1 a 4 e 6 continuaram a apresentar métricas perfeitas, com precisão, recall e F1-score iguais a 1.00, demonstrando consistência na classificação das categorias mais representadas.

A classe 5, com apenas 26 amostras, alcançou um desempenho significativamente superior ao dos testes anteriores: precisão de 1.00, recall de 0.92 e F1-score de 0.96. A matriz de confusão revela que 24 das 26 amostras foram corretamente identificadas, com apenas 2 classificações incorretas, novamente atribuídas à classe 1.

O modelo obteve uma accuracy global de 1.00, com F1-score ponderado de 1.00 e média macro de 0.99. Estes resultados refletem um equilíbrio excecional entre todas as classes, mesmo perante um desequilíbrio acentuado no número de amostras.

O [OOB](#)-score foi de 0.9997, reforçando a generalização e estabilidade do modelo neste terceiro teste. A Figura 33 apresenta a matriz de confusão e o resumo das métricas de desempenho associadas a este terceiro teste.

| | precision | recall | f1-score | support |
|---------------------|-----------|--------|----------|---------|
| 1 | 1.00 | 1.00 | 1.00 | 30001 |
| 2 | 1.00 | 1.00 | 1.00 | 3000 |
| 3 | 1.00 | 1.00 | 1.00 | 3000 |
| 4 | 1.00 | 1.00 | 1.00 | 3000 |
| 5 | 1.00 | 0.92 | 0.96 | 26 |
| 6 | 1.00 | 1.00 | 1.00 | 3000 |
| accuracy | | | 1.00 | 42027 |
| macro avg | 1.00 | 0.99 | 0.99 | 42027 |
| weighted avg | 1.00 | 1.00 | 1.00 | 42027 |
| Matriz de Confusão: | | | | |
| [[29999 0 0 1 0 1] | | | | |
| [0 3000 0 0 0 0] | | | | |
| [2 0 2998 0 0 0] | | | | |
| [0 0 0 3000 0 0] | | | | |
| [2 0 0 0 24 0] | | | | |
| [2 0 0 0 0 2998]] | | | | |
| OOB-score: 0.9997 | | | | |

Figura 33 - Relatório de Classificação e Matriz de Confusão do Teste 3 RF

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.1.2.4 Quarto Teste

No quarto teste, o modelo manteve a consistência nos excelentes resultados obtidos nos testes anteriores. As classes 1 a 4 e 6 continuaram a apresentar métricas perfeitas, com precisão, recall e F1-score iguais a 1.00.

A classe 5 voltou a destacar-se positivamente, atingindo uma precisão de 1.00, um recall de 0.96 e um F1-score de 0.98. De acordo com a matriz de confusão, 25 das 26 amostras desta classe foram corretamente classificadas, com apenas um erro, atribuído à classe 1.

A accuracy global do modelo foi de 1.00, com F1-score médio ponderado igualmente de 1.00 e uma média macro de 0.99, valores que demonstram a elevada capacidade de generalização do modelo. O [OOB](#)-score, também de 0.9997, reforça esta estabilidade e precisão.

A Figura 34 apresenta a matriz de confusão e o resumo das métricas de desempenho associadas a este quarto teste.

| | precision | recall | f1-score | support |
|---------------------|-----------|--------|----------|---------|
| 1 | 1.00 | 1.00 | 1.00 | 30001 |
| 2 | 1.00 | 1.00 | 1.00 | 3000 |
| 3 | 1.00 | 1.00 | 1.00 | 3000 |
| 4 | 1.00 | 1.00 | 1.00 | 3000 |
| 5 | 1.00 | 0.96 | 0.98 | 26 |
| 6 | 1.00 | 1.00 | 1.00 | 3000 |
| accuracy | | | 1.00 | 42027 |
| macro avg | 1.00 | 0.99 | 1.00 | 42027 |
| weighted avg | 1.00 | 1.00 | 1.00 | 42027 |
| Matriz de Confusão: | | | | |
| [[29998 0 1 1 0 1] | | | | |
| [0 3000 0 0 0 0] | | | | |
| [2 0 2998 0 0 0] | | | | |
| [0 0 0 3000 0 0] | | | | |
| [1 0 0 0 25 0] | | | | |
| [1 0 0 0 0 2999]] | | | | |
| OOB-score: 0.9997 | | | | |

Figura 34 - Relatório de Classificação e Matriz de Confusão do Teste 4 RF

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.1.2.5 Modelo Final

O modelo final [RF](#) apresentou um desempenho de elevado nível em praticamente todas as classes. As classes maioritárias (1 a 4 e 6) atingiram valores perfeitos de precisão, recall e F1-score, todos iguais a 1.00, demonstrando a excelente capacidade do modelo para identificar corretamente estas categorias.

Na classe 5, composta por apenas 26 amostras, o modelo alcançou uma precisão de 1.00, um recall de 0.96 e um F1-score de 0.98. A matriz de confusão mostra que 25 das 26 amostras desta classe foram corretamente classificadas, com apenas uma atribuída incorretamente à classe 1.

A accuracy global foi de 1.00, tal como o F1-score médio ponderado, com uma média macro também de 1.00 para precisão, 0.99 para recall e 1.00 para F1-score. O OOB-score obtido foi de 0.9997, reforçando a robustez do modelo na generalização para novos dados.

A validação cruzada com 5 folds apresentou valores médios elevados: precisão ponderada de 98.52%, recall ponderado de 98.43% e F1-score ponderado de 97.97%, com desvios padrão reduzidos. A validação cruzada com 10 folds para o F1-score confirmou a estabilidade do modelo, com resultados elevados na generalidade dos folds e apenas uma ligeira descida no último fold. A Figura 35 apresenta a matriz de confusão e a Figura 36 o gráfico dos F1-scores obtidos na validação cruzada.

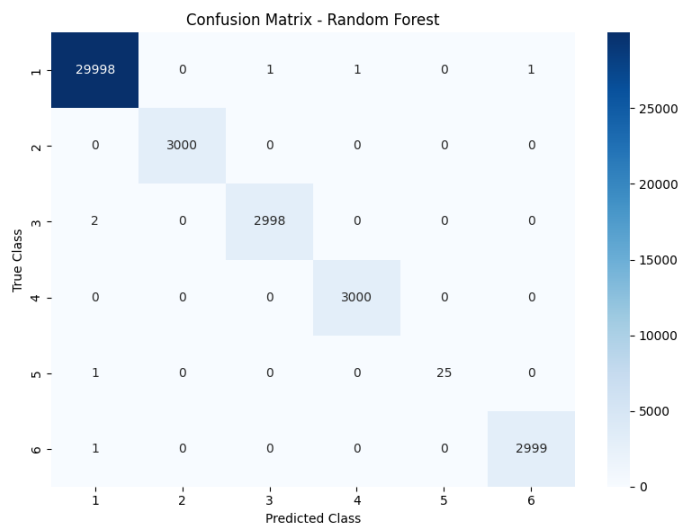


Figura 35 - Matriz de Confusão do Modelo Final RF

Fonte desta figura: Resultados dos Testes a cada Modelo.

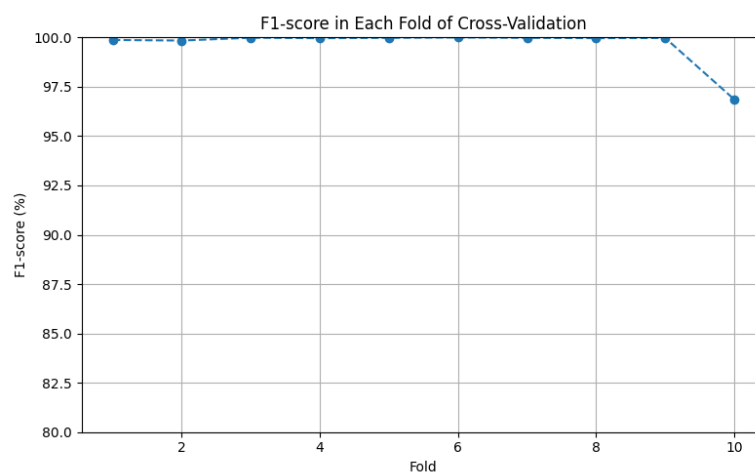


Figura 36 - F1-score em cada Fold na Validação Cruzada do Modelo Final RF

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.1.3 Máquina de Vetores de Suporte

3.3.1.3.1 Primeiro Teste

No primeiro teste com o modelo [SVM](#), os resultados obtidos foram bastante positivos, especialmente nas classes com maior representatividade no conjunto de dados. As classes 1 a 4 apresentaram valores perfeitos de precisão, recall e F1-score, todos iguais a 1.00, evidenciando uma excelente capacidade de classificação nestas categorias. A classe 6 registou também um bom desempenho, com uma precisão de 0.96, um recall de 1.00 e um F1-score de 0.98. A classe 5, composta por apenas 26 amostras, obteve uma precisão de 0.77 e um recall de 0.38, o que

corresponde a um F1-score de 0.51, refletindo algumas dificuldades do modelo na detecção desta classe minoritária.

A matriz de confusão mostra que, das 26 amostras da classe 5, 10 foram corretamente classificadas, enquanto 16 foram atribuídas incorretamente a outras classes, nomeadamente à classe 1. O desempenho global foi excelente, com uma accuracy de 1.00, um F1-score médio ponderado de 1.00 e uma média macro de 0.91.

A Figura 37 apresenta a matriz de confusão e os principais indicadores de desempenho obtidos neste primeiro teste.

```

=== SVM - Teste 1 ===
      precision    recall  f1-score   support

     1       1.00      1.00      1.00     30001
     2       1.00      1.00      1.00      3000
     3       1.00      1.00      1.00      3000
     4       1.00      1.00      1.00      3000
     5       0.77      0.38      0.51         26
     6       0.96      1.00      0.98      3000

 accuracy          1.00     42027
 macro avg         0.95     0.90     0.91     42027
 weighted avg      1.00     1.00     1.00     42027

F1-score: 1.00
Precision: 1.00
Recall: 1.00
Matriz de Confusão:
[[29858   11    1    6    3   122]
 [    0 3000    0    0    0    0]
 [    3    0 2997    0    0    0]
 [    2    0    0 2998    0    0]
 [   16    0    0    0   10    0]
 [    4    0    0    0    0 2996]]

```

Figura 37 - Relatório de Classificação e Matriz de Confusão do Teste 1 SVM

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.1.3.2 Segundo Teste

No segundo teste, os resultados foram idênticos aos do primeiro teste, mantendo um desempenho elevado na maioria das classes. As classes 1 a 4 apresentaram novamente valores perfeitos de precisão, recall e F1-score, todos iguais a 1.00, demonstrando a robustez do modelo na identificação destas classes. A classe 6 registou uma precisão de 0.96, um recall de 1.00 e um F1-score de 0.98. A classe 5, com apenas 26 amostras, obteve uma precisão de 0.77, um recall de 0.38 e um F1-score de 0.51, mostrando as mesmas dificuldades na detecção correta desta classe minoritária.

A matriz de confusão evidencia que, das 26 amostras da classe 5, 10 foram corretamente classificadas e 16 foram atribuídas incorretamente, maioritariamente à classe 1. O desempenho

global manteve-se excelente, com uma accuracy de 1.00, um F1-score médio ponderado de 1.00 e uma média macro de 0.91.

A Figura 38 apresenta a matriz de confusão e os principais indicadores de desempenho obtidos neste segundo teste.

```

=== SVM - Teste 2 ===

```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 1 | 1.00 | 1.00 | 1.00 | 30001 |
| 2 | 1.00 | 1.00 | 1.00 | 3000 |
| 3 | 1.00 | 1.00 | 1.00 | 3000 |
| 4 | 1.00 | 1.00 | 1.00 | 3000 |
| 5 | 0.77 | 0.38 | 0.51 | 26 |
| 6 | 0.96 | 1.00 | 0.98 | 3000 |
| accuracy | | | 1.00 | 42027 |
| macro avg | 0.95 | 0.90 | 0.91 | 42027 |
| weighted avg | 1.00 | 1.00 | 1.00 | 42027 |

Matriz de Confusão:

```

[[29858  11    1    6    3  122]
 [   0 3000    0    0    0    0]
 [   3    0 2997    0    0    0]
 [   2    0    0 2998    0    0]
 [  16    0    0    0   10    0]
 [   4    0    0    0    0 2996]]

```

Figura 38 - Relatório de Classificação e Matriz de Confusão do Teste 2 SVM

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.1.3.3 Terceiro Teste

No terceiro teste, os resultados melhoraram comparativamente com os dois testes anteriores. As classes 1 a 4 continuaram a apresentar precisão, recall e F1-score perfeitos (1.00). A classe 6 registou precisão de 0.96, recall de 1.00 e F1-score de 0.98. A classe 5 apresentou melhoria, com precisão de 0.92, recall de 0.46 e F1-score de 0.62, indicando alguma progressão na identificação desta classe minoritária.

A matriz de confusão revela que 12 das 26 amostras da classe 5 foram corretamente classificadas, enquanto 14 foram incorretamente atribuídas a outras classes, principalmente à classe 1. O desempenho global manteve-se excelente, com accuracy de 1.00, F1-score médio ponderado de 1.00 e média macro de 0.93.

A Figura 39 apresenta a matriz de confusão e os indicadores de desempenho correspondentes a este terceiro teste.


```

=== SVM - Teste 3 ===
              precision    recall  f1-score   support

     1         1.00        1.00        1.00     30001
     2         1.00        1.00        1.00      3000
     3         1.00        1.00        1.00      3000
     4         1.00        1.00        1.00      3000
     5         0.92        0.46        0.62         26
     6         0.96        1.00        0.98      3000

 accuracy              1.00      42027
 macro avg           0.98        0.91        0.93      42027
 weighted avg        1.00        1.00        1.00      42027

Matriz de Confusão:
[[29869    9    1    6    1   115]
 [    0 3000    0    0    0    0]
 [    1    0 2999    0    0    0]
 [    2    0    0 2998    0    0]
 [   14    0    0    0   12    0]
 [    4    0    0    0    0 2996]]

```

Figura 39 - Relatório de Classificação e Matriz de Confusão do Teste 3 SVM

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.1.3.4 Quarto Teste

No quarto teste, o modelo manteve a consistência nos resultados obtidos nos testes anteriores, embora com uma redução significativa na classe minoritária. As classes 1 a 4 e 6 apresentaram métricas elevadas, com precisão, recall e F1-score próximos de 1.00.

A classe 5 apresentou uma precisão de 0.01, recall de 1.00 e F1-score de 0.02, refletindo um desempenho bastante limitado nesta classe minoritária. A matriz de confusão indica que 26 amostras foram corretamente classificadas nesta classe, sem erros aparentes, o que sugere alguma incoerência na avaliação das métricas.

A accuracy global do modelo foi de 0.94, com F1-score médio ponderado de 0.96 e média macro de 0.82, demonstrando um bom desempenho geral, apesar das limitações identificadas na classe 5.

A Figura 40 apresenta a matriz de confusão e o resumo das métricas de desempenho associadas a este quarto teste.

```

=== SVM - Teste 4 ===

```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 1 | 1.00 | 0.92 | 0.96 | 30001 |
| 2 | 1.00 | 1.00 | 1.00 | 3000 |
| 3 | 0.96 | 1.00 | 0.98 | 3000 |
| 4 | 0.99 | 1.00 | 1.00 | 3000 |
| 5 | 0.01 | 1.00 | 0.02 | 26 |
| 6 | 0.96 | 1.00 | 0.98 | 3000 |
| accuracy | | | 0.94 | 42027 |
| macro avg | 0.82 | 0.99 | 0.82 | 42027 |
| weighted avg | 0.99 | 0.94 | 0.96 | 42027 |

Matriz de Confusão:

```

[[27484  11  124  17 2252  113]
 [  0 3000  0  0  0  0]
 [  0  0 3000  0  0  0]
 [  2  0  0 2998  0  0]
 [  0  0  0  0 26  0]
 [  0  0  0  0  0 3000]]

```

Figura 40 - Relatório de Classificação e Matriz de Confusão do Teste 4 SVM

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.1.3.5 Modelo Final

O modelo final [SVM](#), proveniente do Terceiro Teste, apresentou um desempenho muito sólido, destacando-se sobretudo nas classes mais representativas do conjunto de dados. As classes 1 a 4 e 6 atingiram valores perfeitos de precisão, recall e F1-score, todos iguais a 1.00, demonstrando a excelente capacidade do modelo para identificar corretamente estas categorias. A classe 5, com apenas 26 amostras, registou uma precisão de 1.00, um recall de 0.77 e um F1-score de 0.87, refletindo uma melhoria na capacidade de detecção desta classe minoritária face a configurações anteriores.

A matriz de confusão evidencia que, na classe 5, 20 das 26 amostras foram corretamente classificadas, enquanto 6 foram atribuídas incorretamente a outras classes, todas à classe 1. Apesar deste desvio, o impacto no desempenho global foi reduzido devido ao peso das restantes classes no conjunto.

A accuracy global do modelo foi de 1.00, com um F1-score médio ponderado também de 1.00 e uma média macro de 0.98, reforçando a consistência do desempenho entre classes.

A Figura 41 apresenta a matriz de confusão obtida para o modelo final, enquanto a Figura 42 mostra o gráfico do F1-score em cada fold na validação cruzada com 10 folds, onde os valores mantiveram-se elevados em todos os folds, evidenciando a estabilidade do modelo.

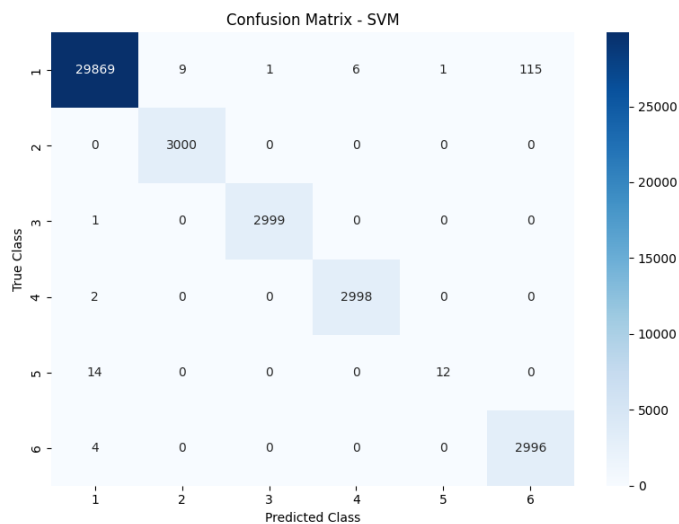


Figura 41 - Matriz de Confusão do Modelo Final SVM

Fonte desta figura: Resultados dos Testes a cada Modelo.

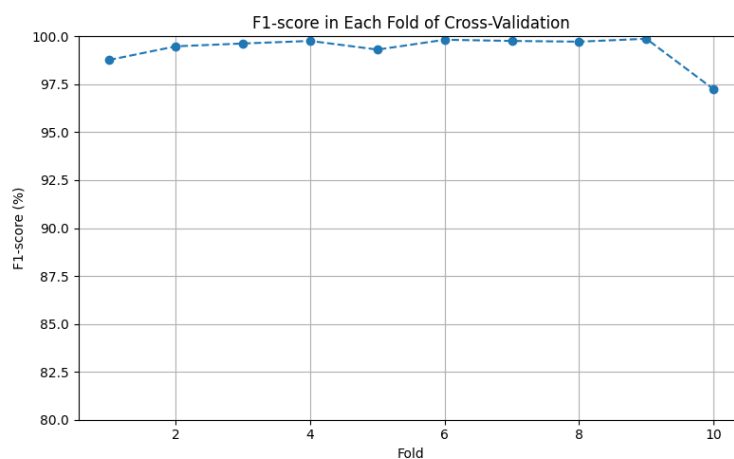


Figura 42 - F1-score em cada Fold na Validação Cruzada do Modelo Final SVM

Fonte desta figura: Resultados dos Testes a cada Modelo.

A validação cruzada com 5 folds, indicou médias de 99.12% de precisão, 99.10% de recall e 99.05% de F1-score, com desvios padrão reduzidos, confirmando a robustez do modelo final.

3.3.2 Modelos Não Supervisionados

3.3.2.1 K-Means Clustering

3.3.2.1.1 Primeiro Teste

No primeiro teste com o modelo [KM](#), os resultados obtidos foram mais modestos face aos modelos supervisionados anteriormente apresentados. A classe normal obteve uma precisão de 0.77 e um recall de 0.73, resultando num F1-score de 0.75. Já a classe anomaly apresentou valores inferiores, com uma precisão de 0.41, recall de 0.47 e F1-score de 0.44, refletindo dificuldades do modelo na correta identificação das amostras anómalas.

A matriz de confusão mostra que, das 30001 amostras da classe normal, 21853 foram corretamente identificadas, enquanto 8148 foram classificadas como anomaly. Para a classe anomaly, das 12026 amostras, 5644 foram corretamente identificadas e 6382 foram incorretamente atribuídas à classe normal.

A accuracy global foi de 0.65, com F1-score médio ponderado de 0.66 e média macro de 0.59, demonstrando um desempenho limitado do modelo neste primeiro teste. A Figura 43 apresenta a matriz de confusão e o resumo das métricas de desempenho associadas a estes resultados.

| | precision | recall | f1-score | support |
|---------------------|---------------|--------|----------|---------|
| normal | 0.77 | 0.73 | 0.75 | 30001 |
| anomaly | 0.41 | 0.47 | 0.44 | 12026 |
| accuracy | | | 0.65 | 42027 |
| macro avg | 0.59 | 0.60 | 0.59 | 42027 |
| weighted avg | 0.67 | 0.65 | 0.66 | 42027 |
| Matriz de Confusão: | | | | |
| | [[21853 8148] | | | |
| | [6382 5644]] | | | |

Figura 43 - Relatório de Classificação e Matriz de Confusão do Teste 1 KM

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.2.1.2 Segundo Teste

No segundo teste, os resultados obtidos foram semelhantes aos do primeiro teste, com ligeira descida nas métricas. A classe normal obteve uma precisão de 0.74, um recall de 0.73 e um F1-score de 0.73. A classe anomaly registou valores mais baixos, com uma precisão de 0.34, recall de 0.35 e F1-score de 0.34, mantendo as dificuldades na deteção correta de anomalias.

A matriz de confusão evidencia que, das 30001 amostras da classe normal, 21902 foram corretamente classificadas e 8099 foram classificadas como anomaly. Relativamente à classe

anomaly, 4168 das 12026 amostras foram corretamente identificadas, enquanto 7858 foram incorretamente rotuladas como normal.

A accuracy global foi de 0.62, com F1-score médio ponderado de 0.62 e média macro de 0.54. A Figura 44 apresenta a matriz de confusão e o resumo das métricas de desempenho associadas a este segundo teste.

| | precision | recall | f1-score | support |
|---------------------|-----------|--------|----------|---------|
| normal | 0.74 | 0.73 | 0.73 | 30001 |
| anomaly | 0.34 | 0.35 | 0.34 | 12026 |
| accuracy | | | 0.62 | 42027 |
| macro avg | 0.54 | 0.54 | 0.54 | 42027 |
| weighted avg | 0.62 | 0.62 | 0.62 | 42027 |
| Matriz de Confusão: | | | | |
| [[21902 8099] | | | | |
| [7858 4168]] | | | | |

Figura 44 - Relatório de Classificação e Matriz de Confusão do Teste 2 KM

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.2.1.3 Terceiro Teste

No terceiro teste, o modelo apresentou exatamente os mesmos resultados verificados no segundo teste. A classe normal obteve uma precisão de 0.74, recall de 0.73 e F1-score de 0.73. Já a classe anomaly manteve métricas mais reduzidas, com precisão de 0.34, recall de 0.35 e F1-score de 0.34.

A matriz de confusão mostra que, das 30001 amostras da classe normal, 21902 foram corretamente classificadas e 8099 foram incorretamente atribuídas à classe anomaly. Na classe anomaly, 4168 das 12026 amostras foram corretamente identificadas, enquanto 7858 foram classificadas como normal.

A accuracy global foi de 0.62, o F1-score ponderado de 0.62 e a média macro de 0.54. A Figura 45 apresenta a matriz de confusão e o resumo das métricas de desempenho associadas a este terceiro teste.

| | precision | recall | f1-score | support |
|---------------------|-----------|--------|----------|---------|
| normal | 0.74 | 0.73 | 0.73 | 30001 |
| anomaly | 0.34 | 0.35 | 0.34 | 12026 |
| accuracy | | | 0.62 | 42027 |
| macro avg | 0.54 | 0.54 | 0.54 | 42027 |
| weighted avg | 0.62 | 0.62 | 0.62 | 42027 |
| Matriz de Confusão: | | | | |
| [[21902 8099] | | | | |
| [7858 4168]] | | | | |

Figura 45 - Relatório de Classificação e Matriz de Confusão do Teste 3 KM

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.2.1.4 Quarto Teste

No quarto e último teste, os resultados obtidos foram idênticos aos verificados nos testes anteriores. A classe normal apresentou uma precisão de 0.74, recall de 0.73 e F1-score de 0.73. A classe anomaly manteve métricas inferiores, com uma precisão de 0.34, recall de 0.35 e F1-score de 0.34.

A matriz de confusão confirma estes valores: das 30001 amostras da classe normal, 21902 foram corretamente classificadas e 8099 foram classificadas como anomaly. Para a classe anomaly, 4168 das 12026 amostras foram corretamente identificadas, enquanto 7858 foram rotuladas como normal.

A accuracy global foi de 0.62, com F1-score ponderado de 0.62 e média macro de 0.54. A Figura 46 apresenta a matriz de confusão e o resumo das métricas de desempenho associadas a este quarto teste.

| === K-Means Otimizado === | | | | |
|---------------------------|-----------|--------|----------|---------|
| | precision | recall | f1-score | support |
| normal | 0.74 | 0.73 | 0.73 | 30001 |
| anomaly | 0.34 | 0.35 | 0.34 | 12026 |
| accuracy | | | 0.62 | 42027 |
| macro avg | 0.54 | 0.54 | 0.54 | 42027 |
| weighted avg | 0.62 | 0.62 | 0.62 | 42027 |
| Matriz de Confusão: | | | | |
| [[21902 8099] | | | | |
| [7858 4168]] | | | | |

Figura 46 - Relatório de Classificação e Matriz de Confusão do Teste 4 KM

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.2.1.5 Modelo Final

No modelo final [KM](#), os resultados obtidos revelaram limitações na distinção entre tráfego normal e anômalo. A classe normal apresentou uma precisão de 0.77, um recall de 0.73 e um F1-score de 0.75. A classe anomaly obteve valores mais modestos, com uma precisão de 0.41, um recall de 0.47 e um F1-score de 0.44.

A matriz de confusão mostra que, das 30001 amostras da classe normal, 21853 foram corretamente classificadas, enquanto 8148 foram atribuídas à classe anomaly. Para a classe anomaly, 5644 das 12026 amostras foram corretamente identificadas, enquanto 6382 foram classificadas como normal.

A accuracy global do modelo foi de 0.65, com um F1-score médio ponderado de 0.66 e uma média macro de 0.59. A Figura 47 apresenta a matriz de confusão gerada para este modelo.

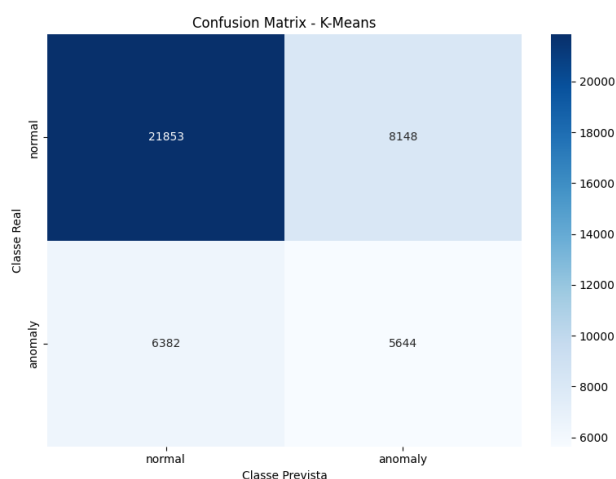


Figura 47 - Matriz de Confusão do Modelo Final KM

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.2.2 Isolation Forest

3.3.2.2.1 Primeiro Teste

No primeiro teste com o modelo [IF](#), os resultados obtidos foram mais modestos face aos modelos supervisionados anteriormente apresentados. A classe normal obteve uma precisão de 0.70 e um recall de 0.90, resultando num F1-score de 0.79. Já a classe anomaly apresentou valores inferiores, com uma precisão de 0.16, recall de 0.05 e F1-score de 0.07, refletindo dificuldades do modelo na correta identificação das amostras anômalas.

A matriz de confusão mostra que, das 30001 amostras da classe normal, 27907 foram corretamente identificadas, enquanto 2904 foram classificadas como anomaly. Para a classe

anomaly, das 12026 amostras, 557 foram corretamente identificadas e 11469 foram incorretamente atribuídas à classe normal.

A accuracy global foi de 0.66, com F1-score médio ponderado de 0.58 e média macro de 0.43. A Figura 48 apresenta a matriz de confusão e o resumo das métricas de desempenho associadas a estes resultados.

```

=== Isolation Forest (Simples) ===
      precision    recall  f1-score   support

   normal         0.70      0.90      0.79     30001
  anomaly         0.16      0.05      0.07     12026

 accuracy          0.66          0.66     42027
 macro avg         0.43      0.47      0.43     42027
weighted avg         0.55      0.66      0.58     42027

Matriz de Confusão:
[[27097  2904]
 [11469   557]]

```

Figura 48 - Relatório de Classificação e Matriz de Confusão do Teste 1 IF

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.2.2.2 Segundo Teste

No segundo teste, os resultados obtidos foram semelhantes aos do primeiro teste, mantendo as dificuldades na detecção correta de anomalias. A classe normal apresentou uma precisão de 0.70, um recall de 0.90 e um F1-score de 0.79. A classe anomaly registou valores mais baixos, com uma precisão de 0.09, um recall de 0.03 e um F1-score de 0.04.

A matriz de confusão evidencia que, das 30001 amostras da classe normal, 26976 foram corretamente classificadas e 3025 foram classificadas como anomaly. Relativamente à classe anomaly, 314 das 12026 amostras foram corretamente identificadas, enquanto 11712 foram rotuladas como normal.

A accuracy global foi de 0.65, com F1-score médio ponderado de 0.57 e média macro de 0.41. A Figura 49 apresenta a matriz de confusão e o resumo das métricas de desempenho associadas a este segundo teste.

```

=== Isolation Forest - Teste 2 (com VarianceThreshold) ===
      precision    recall  f1-score   support

   normal         0.70      0.90      0.79     30001
  anomaly         0.09      0.03      0.04     12026

 accuracy          0.65          0.65     42027
 macro avg         0.40      0.46      0.41     42027
weighted avg         0.52      0.65      0.57     42027

Matriz de Confusão:
[[26976  3025]
 [11712   314]]

```

Figura 49 - Relatório de Classificação e Matriz de Confusão do Teste 2 IF

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.2.2.3 Terceiro Teste

No terceiro teste, os resultados obtidos foram semelhantes aos dos testes anteriores. Embora se tenham verificado ligeiras melhorias, as dificuldades na deteção correta de anomalias mantiveram-se. A classe normal apresentou uma precisão de 0.70, um recall de 0.91 e um F1-score de 0.79. A classe anomaly registou valores mais baixos, com uma precisão de 0.16, um recall de 0.04 e um F1-score de 0.07.

A matriz de confusão evidencia que, das 30001 amostras da classe normal, 27167 foram corretamente classificadas e 2834 foram classificadas como anomaly. Relativamente à classe anomaly, 527 das 12026 amostras foram corretamente identificadas, enquanto 11499 foram rotuladas como normal.

A accuracy global foi de 0.66, com F1-score médio ponderado de 0.58 e média macro de 0.43. A Figura 50 apresenta a matriz de confusão e o resumo das métricas de desempenho associadas a este terceiro teste.

```
=== Isolation Forest - Teste 3 (com PCA) ===
      precision    recall  f1-score   support

   normal         0.70         0.91         0.79     30001
  anomaly         0.16         0.04         0.07     12026

 accuracy          0.66          0.66          0.66     42027
 macro avg         0.43         0.47         0.43     42027
 weighted avg      0.55         0.66         0.58     42027

Matriz de Confusão:
[[27167  2834]
 [11499   527]]
```

Figura 50 - Relatório de Classificação e Matriz de Confusão do Teste 3 IF

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.2.2.4 Quarto Teste

No quarto teste, os resultados obtidos foram diferentes dos testes anteriores, apresentando um maior equilíbrio entre as métricas das duas classes, embora com uma descida na performance global. A classe normal apresentou uma precisão de 0.69, um recall de 0.68 e um F1-score de 0.69. A classe anomaly registou uma precisão de 0.24, um recall de 0.24 e um F1-score de 0.24.

A matriz de confusão evidencia que, das 30001 amostras da classe normal, 20514 foram corretamente classificadas e 9487 foram classificadas como anomaly. Relativamente à classe

anomaly, 2938 das 12026 amostras foram corretamente identificadas, enquanto 9088 foram rotuladas como normal.

A accuracy global foi de 0.56, com F1-score médio ponderado de 0.56 e média macro de 0.46. A Figura 51 apresenta a matriz de confusão e o resumo das métricas de desempenho associadas a este quarto teste.

```

=== Isolation Forest - Teste 4 ===
precision    recall  f1-score   support

   normal     0.69     0.68     0.69     30001
  anomaly     0.24     0.24     0.24     12026

 accuracy          0.56     42027
 macro avg         0.46     0.46     0.46     42027
weighted avg         0.56     0.56     0.56     42027

Matriz de Confusão:
[[20514  9487]
 [ 9088  2938]]

```

Figura 51 - Relatório de Classificação e Matriz de Confusão do Teste 4 IF

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.2.2.5 Modelo Final

No modelo final IF, os resultados obtidos refletiram os resultados do Terceiro Teste, mantendo as dificuldades na detecção correta de anomalias. A classe normal apresentou uma precisão de 0.70, um recall de 0.91 e um F1-score de 0.79. A classe anomaly registou uma precisão de 0.16, um recall de 0.04 e um F1-score de 0.07.

A matriz de confusão evidencia que, das 30001 amostras da classe normal, 27167 foram corretamente classificadas e 2834 foram classificadas como anomaly. Relativamente à classe anomaly, 527 das 12026 amostras foram corretamente identificadas, enquanto 11499 foram rotuladas como normal.

A accuracy global foi de 0.66, com F1-score médio ponderado de 0.58 e média macro de 0.43. A Figura 52 apresenta a matriz de confusão associada ao modelo final.

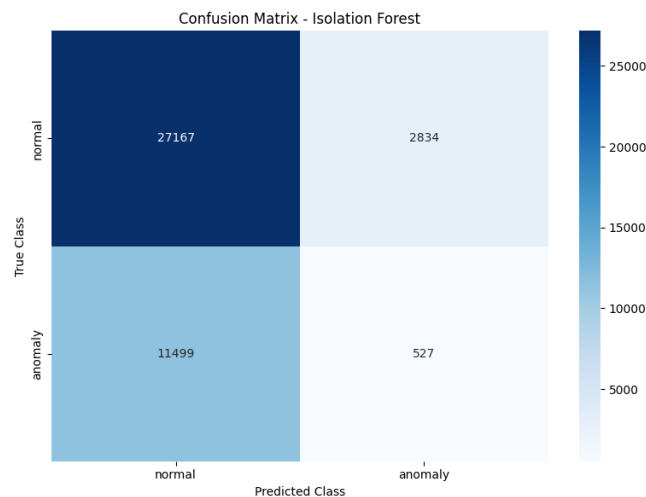


Figura 52 - Matriz de Confusão do Modelo Final IF

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.2.3 Autoencoder

3.3.2.3.1 Primeiro Teste

No primeiro teste com o modelo [AE](#), a classe normal apresentou uma precisão de 0.84, um recall de 0.95 e um F1-score de 0.89. A classe anomaly registou uma precisão de 0.81, um recall de 0.54 e um F1-score de 0.65.

A matriz de confusão evidencia que, das 30001 amostras da classe normal, 28505 foram corretamente classificadas e 1496 foram classificadas como anomaly. Relativamente à classe anomaly, 6446 das 12026 amostras foram corretamente identificadas, enquanto 5580 foram rotuladas como normal.

A accuracy global foi de 0.83, com F1-score médio ponderado de 0.82 e média macro de 0.77. A Figura 53 apresenta a matriz de confusão e o resumo das métricas de desempenho associadas a este primeiro teste.

| | | | | |
|------------------------------------------|-----------|--------|----------|---------|
| === Autoencoder - Teste 1 (Baseline) === | | | | |
| | precision | recall | f1-score | support |
| normal | 0.84 | 0.95 | 0.89 | 30001 |
| anomaly | 0.81 | 0.54 | 0.65 | 12026 |
| accuracy | | | 0.83 | 42027 |
| macro avg | 0.82 | 0.74 | 0.77 | 42027 |
| weighted avg | 0.83 | 0.83 | 0.82 | 42027 |
| Matriz de Confusão: | | | | |
| [[28505 1496] | | | | |
| [5580 6446]] | | | | |
| F1-score: 0.65 | | | | |

Figura 53 - Relatório de Classificação e Matriz de Confusão do Teste 1 AE

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.2.3.2 Segundo Teste

No segundo teste, a classe normal apresentou uma precisão de 0.83, um recall de 0.95 e um F1-score de 0.89. A classe anomaly registou uma precisão de 0.81, um recall de 0.53 e um F1-score de 0.64.

A matriz de confusão evidencia que, das 30001 amostras da classe normal, 28519 foram corretamente classificadas e 1482 foram classificadas como anomaly. Relativamente à classe anomaly, 6367 das 12026 amostras foram corretamente identificadas, enquanto 5659 foram rotuladas como normal.

A accuracy global foi de 0.83, com F1-score médio ponderado de 0.82 e média macro de 0.76. A Figura 54 apresenta a matriz de confusão e o resumo das métricas de desempenho associadas a este segundo teste.

```
=== Autoencoder - Teste 2 (com VarianceThreshold) ===
      precision    recall  f1-score   support

   normal         0.83         0.95         0.89        30001
  anomaly         0.81         0.53         0.64        12026

 accuracy                   0.83        42027
 macro avg         0.82         0.74         0.76        42027
 weighted avg         0.83         0.83         0.82        42027

Matriz de Confusão:
[[28519  1482]
 [ 5659  6367]]
F1-score: 0.64
```

Figura 54 - Relatório de Classificação e Matriz de Confusão do Teste 2 AE

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.2.3.3 Terceiro Teste

No terceiro teste, a classe normal apresentou uma precisão de 0.87, um recall de 0.95 e um F1-score de 0.91. A classe anomaly registou uma precisão de 0.84, um recall de 0.66 e um F1-score de 0.74.

A matriz de confusão evidencia que, das 30001 amostras da classe normal, 28529 foram corretamente classificadas e 1472 foram classificadas como anomaly. Relativamente à classe anomaly, 7889 das 12026 amostras foram corretamente identificadas, enquanto 4137 foram rotuladas como normal.

A accuracy global foi de 0.87, com F1-score médio ponderado de 0.86 e média macro de 0.82. A Figura 55 apresenta a matriz de confusão e o resumo das métricas de desempenho associadas a este terceiro teste.

```

=== Autoencoder - Teste 3 ===
      precision    recall  f1-score   support

   normal         0.87      0.95      0.91      30001
   anomaly         0.84      0.66      0.74      12026

 accuracy          0.87          0.87      42027
 macro avg         0.86      0.80      0.82      42027
 weighted avg      0.86      0.87      0.86      42027

Matriz de Confusão:
[[28529  1472]
 [ 4137  7889]]
F1-score: 0.74

```

Figura 55 - Relatório de Classificação e Matriz de Confusão do Teste 3 AE

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.2.3.4 Quarto Teste

No quarto teste, a classe normal apresentou uma precisão de 0.91, um recall de 0.95 e um F1-score de 0.93. A classe anomaly registou uma precisão de 0.86, um recall de 0.78 e um F1-score de 0.82.

A matriz de confusão evidencia que, das 30001 amostras da classe normal, 28518 foram corretamente classificadas e 1483 foram classificadas como anomaly. Relativamente à classe anomaly, 9367 das 12026 amostras foram corretamente identificadas, enquanto 2659 foram rotuladas como normal.

A accuracy global foi de 0.90, com F1-score médio ponderado de 0.90 e média macro de 0.88. A Figura 56 apresenta a matriz de confusão e o resumo das métricas de desempenho associadas a este quarto teste.

```

=== Autoencoder - Teste 4 ===
      precision    recall  f1-score   support

   normal         0.91      0.95      0.93      30001
   anomaly         0.86      0.78      0.82      12026

 accuracy          0.90          0.90      42027
 macro avg         0.89      0.86      0.88      42027
 weighted avg      0.90      0.90      0.90      42027

Matriz de Confusão:
[[28518  1483]
 [ 2659  9367]]
F1-score: 0.82

```

Figura 56 - Relatório de Classificação e Matriz de Confusão do Teste 4 AE

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.2.3.5 Modelo Final

No modelo final com o [AE](#), os resultados obtidos demonstraram um desempenho ótimo na detecção de tráfego anômalo. A classe normal apresentou uma precisão de 0.91, um recall de 0.95 e um F1-score de 0.93. A classe anomaly obteve uma precisão de 0.86, um recall de 0.76 e um F1-score de 0.80.

A matriz de confusão mostra que, das 30001 amostras da classe normal, 28479 foram corretamente classificadas, enquanto 1522 foram atribuídas à classe anomaly. Para a classe anomaly, 9095 das 12026 amostras foram corretamente identificadas e 2931 foram incorretamente rotuladas como normal.

A accuracy global foi de 0.89, com F1-score médio ponderado de 0.89 e média macro de 0.87. A Figura 57 apresenta a matriz de confusão do modelo final.

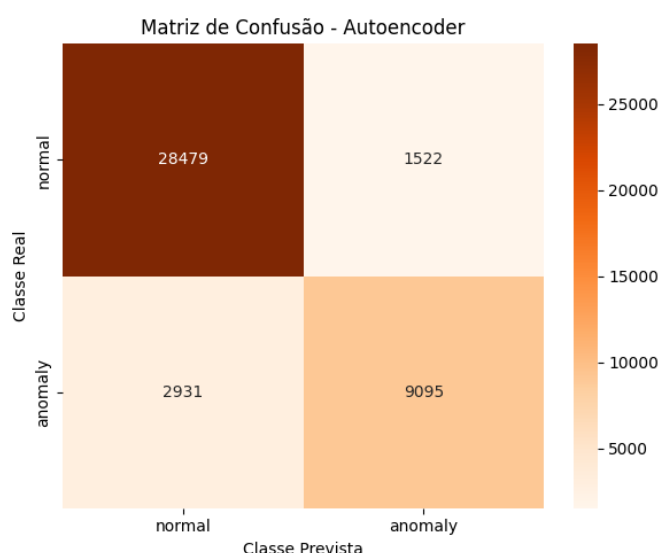


Figura 57 - Matriz de Confusão do Modelo Final AE

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.3.2.3.6 Teste Threshold

No teste de thresholds aplicado ao [AE](#), foram obtidos os seguintes resultados:

- Threshold 50: F1-score 0.0816, precision 0.9922, recall 0.0426
- Threshold 75: F1-score 0.0816, precision 0.9922, recall 0.0426
- Threshold 100: F1-score 0.0816, precision 0.9922, recall 0.0426
- Threshold 125: F1-score 0.0816, precision 0.9922, recall 0.0426
- Threshold 150: F1-score 0.0816, precision 0.9922, recall 0.0426
- Threshold 200: F1-score 0.0816, precision 0.9922, recall 0.0426

- Threshold 300: F1-score 0.0816, precision 0.9922, recall 0.0426
- Threshold 500: F1-score 0.0816, precision 0.9922, recall 0.0426
- Threshold 1000: F1-score 0.0817, precision 0.9942, recall 0.0426

3.4 Resultados Obtidos com os Testes Preliminares aos Modelos de Machine Learning

Nesta secção são apresentados os resultados obtidos com os testes preliminares realizados sobre os modelos finais de [ML](#) desenvolvidos para o [IDS](#). Estes testes tiveram como objetivo avaliar o desempenho prático individual dos modelos num ambiente pré-preparado e simulado com cenários realistas de tráfego de rede. Os testes incluíram os modelos supervisionados e os modelos não supervisionados sendo os resultados documentados em relatórios e gráficos, ilustrando o número de fluxos classificados como suspeitos e a distribuição das classes detetadas por cada modelo.

3.4.1 Resultados Obtidos com os Testes Preliminares aos Modelos Supervisionados

No primeiro teste, com o ficheiro `exfiltracao_dados_tcp.csv`, todos os modelos classificaram o fluxo analisado como benigno, não tendo sido detetados fluxos suspeitos.

No segundo teste, com o ficheiro `normal.csv`, os três modelos voltaram a classificar corretamente o fluxo como benigno, sem fluxos suspeitos detetados.

No terceiro teste, com o ficheiro `normal2.csv`, o modelo [MLP](#) detetou 6 fluxos suspeitos em 32 fluxos analisados, todos classificados como pertencentes à classe DDoS. Por sua vez, os modelos [RF](#) e [SVM](#) não detetaram quaisquer fluxos suspeitos. Os detalhes encontram-se nos gráficos da Figura 58.

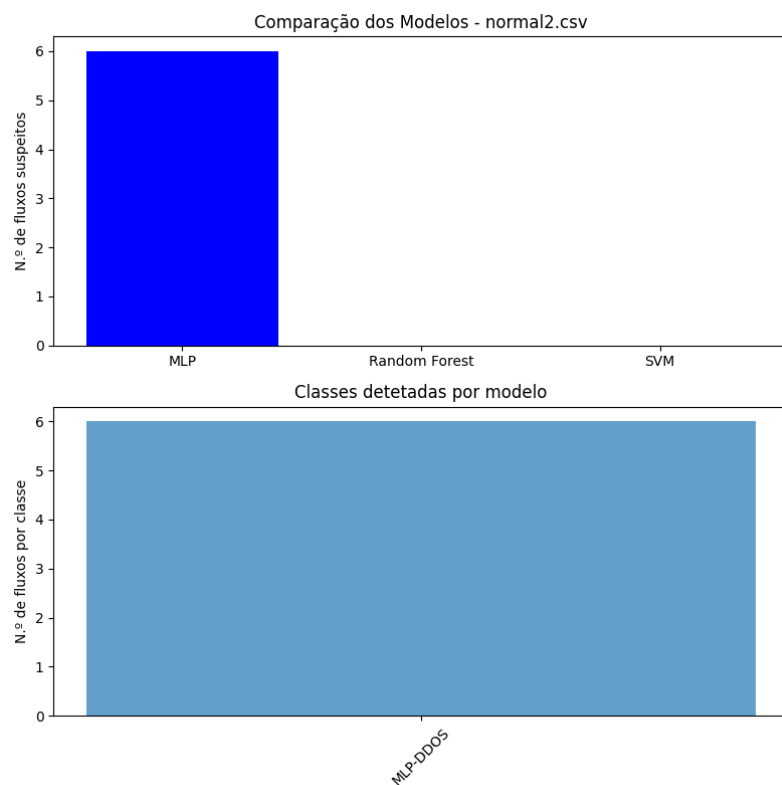


Figura 58 - Gráfico do Terceiro Teste Supervisionado

Fonte desta figura: Resultados dos Testes a cada Modelo.

No quarto teste, realizado com o ficheiro `tcp_syn_flooding.csv`, o modelo [MLP](#) identificou 521 fluxos suspeitos em 2241 fluxos analisados, distribuídos pelas classes DDoS (159 fluxos) e Bot (362 fluxos). Os modelos [RF](#) e [SVM](#) não detetaram fluxos suspeitos neste teste. Estes resultados estão ilustrados nos gráficos da Figura 59.

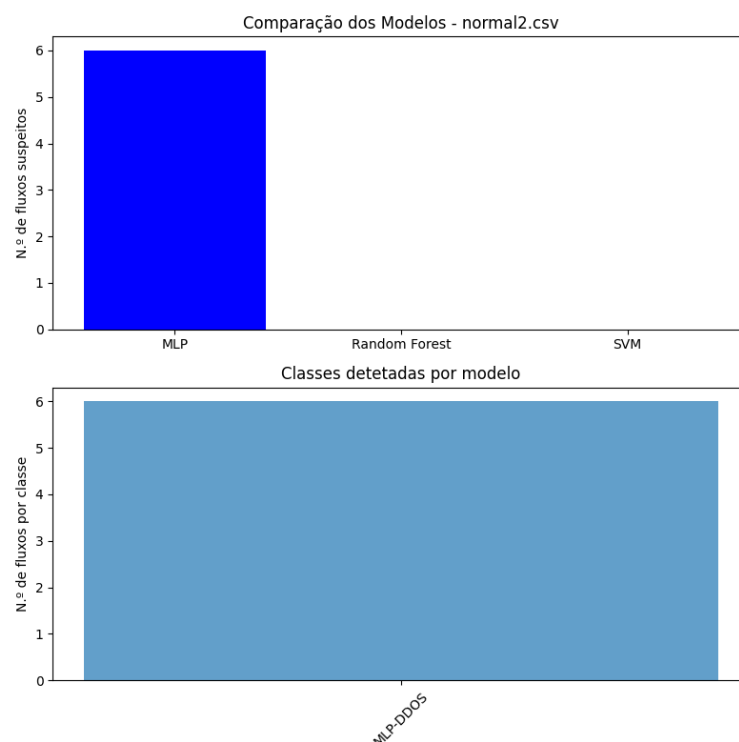


Figura 59 - Gráficos do Quarto Teste Supervisionado

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.4.2 Resultados Obtidos com os Testes Preliminares aos Modelos Não Supervisionados

No primeiro teste, com o ficheiro `exfiltragem_dados_tcp.csv`, todos os modelos identificaram o fluxo analisado como suspeito. O [AE](#), o [IF](#) e o [KM](#) detetaram 1 fluxo suspeito em 1 analisado. Os gráficos correspondentes a este teste encontram-se na Figura 60.

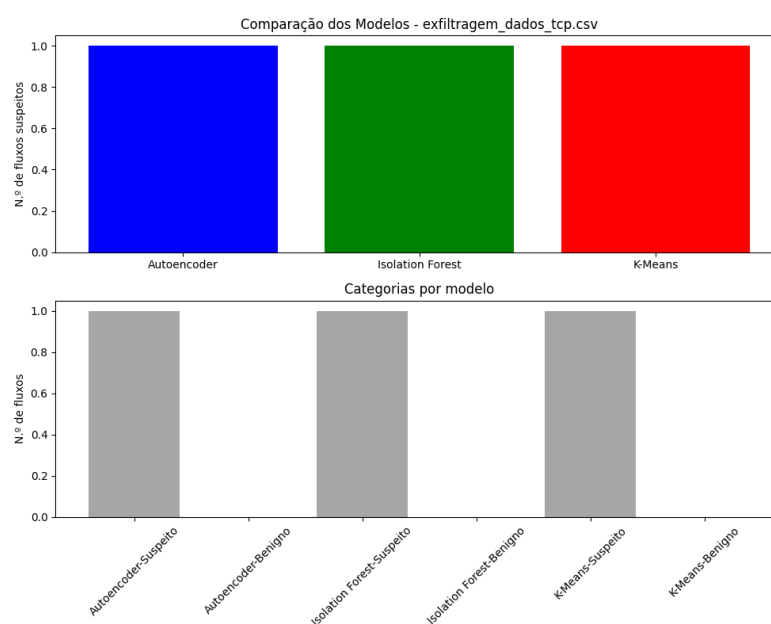


Figura 60 - Gráficos do Primeiro Teste Não Supervisionado

Fonte desta figura: Resultados dos Testes a cada Modelo.

No segundo teste, com o ficheiro normal.csv, o [AE](#) e o [KM](#) classificaram corretamente o fluxo como benigno, enquanto o [IF](#) identificou o fluxo como suspeito. Este resultado está ilustrado nos gráficos da Figura 61.

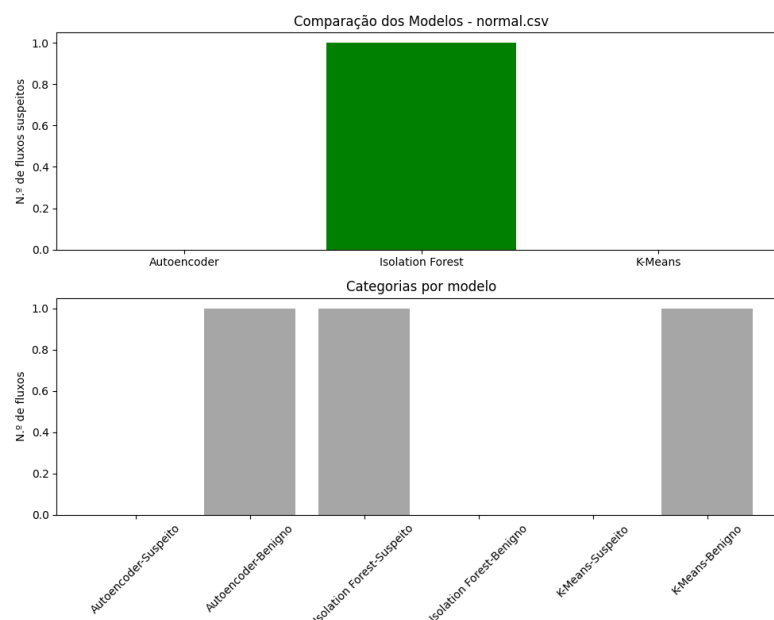


Figura 61 - Gráficos do Segundo Teste Não Supervisionado

Fonte desta figura: Resultados dos Testes a cada Modelo.

No terceiro teste, com o ficheiro normal2.csv, o [AE](#) detetou 9 fluxos suspeitos em 32 fluxos analisados, o [IF](#) detetou os 32 fluxos como suspeitos e o [KM](#) identificou 21 fluxos como suspeitos e 11 como benignos. Os detalhes encontram-se nos gráficos da Figura 62.

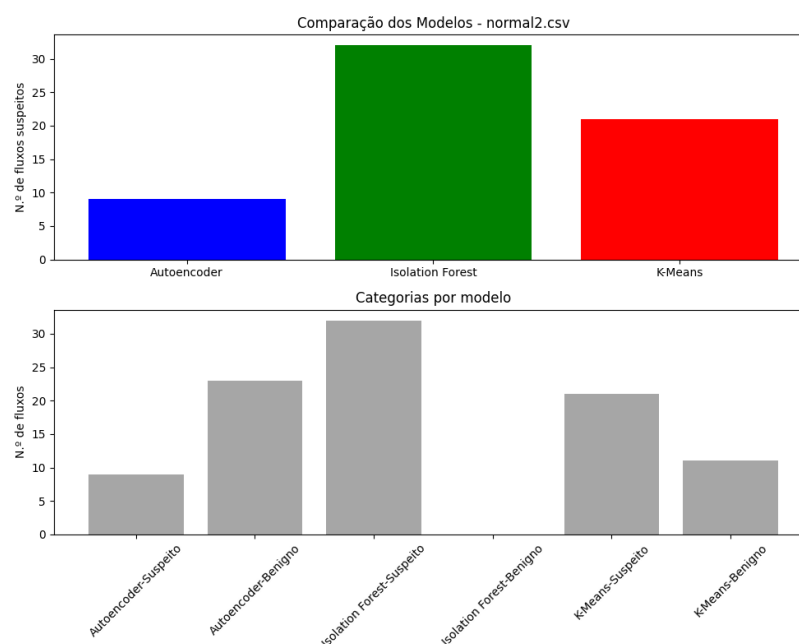


Figura 62 - Gráficos do Terceiro Teste Não Supervisionado

Fonte desta figura: Resultados dos Testes a cada Modelo.

No quarto teste, realizado com o ficheiro `tcp_syn_flooding.csv`, o [AE](#) e o [KM](#) não detetaram fluxos suspeitos, classificando todos os 2241 fluxos como benignos. Por outro lado, o [IF](#) identificou os 2241 fluxos como suspeitos. Estes resultados estão ilustrados nos gráficos da Figura 63.

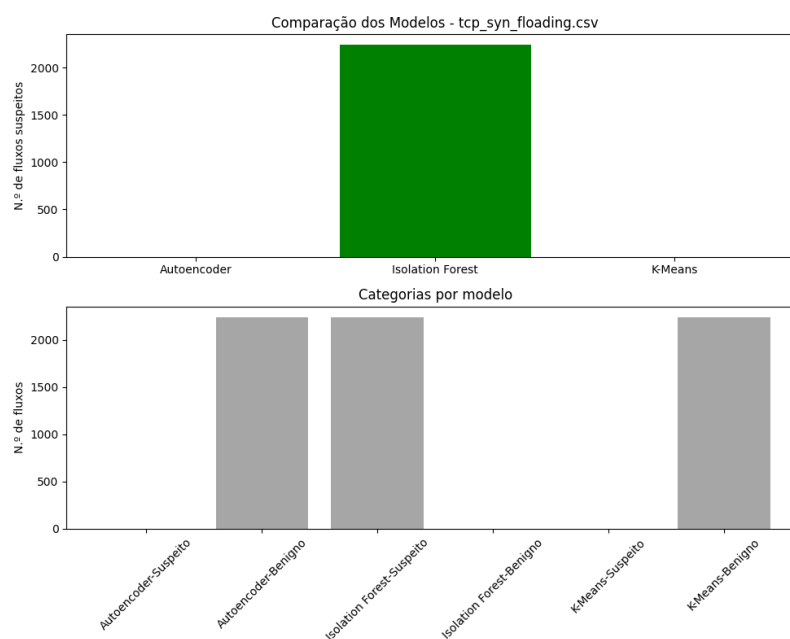


Figura 63 - Gráficos do Quarto Teste Não Supervisionado

Fonte desta figura: Resultados dos Testes a cada Modelo.

3.5 Resultados Obtidos do Ambiente de Testes

Nesta secção são apresentados os resultados obtidos durante a execução prática do [IDS](#) no ambiente de testes. Foram realizados diferentes cenários, simulando tráfego de rede com diferentes características, com o objetivo de avaliar o comportamento do sistema em situações reais de deteção.

Cada cenário envolveu a análise de ficheiros de fluxos de rede capturados pelo CICFlowMeter e processados pelo script “runids”, aplicando a arquitetura com os modelos de [ML](#) definida. Os resultados incluem o número de fluxos processados, a classificação atribuída pelo sistema e as ações realizadas (alertas ou ausência de ação).

3.5.1 Análise de Tráfego com Elevado Número de Fluxos

Num primeiro teste, ilustrado na Figura 64, foi avaliada a resposta do [IDS](#) face a um comportamento suspeito com um volume significativo de fluxos no tráfego através de um SYN Flood. O ficheiro analisado continha um total de 2241 fluxos de rede, tendo o modelo [MLP](#) identificado 521 fluxos suspeitos, os quais foram classificados como pertencentes a um ataque do tipo Bot. Perante este resultado, o sistema emitiu o respetivo alerta ao utilizador, e terminou a análise do fluxo recebido, conforme previsto na arquitetura do funcionamento do [IDS](#).

```
(kali@kali) - [~/Desktop/IDS]
$ runids
✓ Ambiente virtual já existe. Pronto para rodar.
🐍 Usando Python: Python 3.10.14
🔥 Executando ids.py...
[INFO] IDS a correr. À espera de fluxos...

[INFO] Ficheiro recebido: cicflow_20250622_133904.csv (2241 fluxos)

» [MLP] A analisar ficheiro completo com Rede Neuronal...
MLP detetou 521 fluxos anómalos num total de 2241
↳ Vários fluxos classificados como 'Bot' pela MLP.

Ficheiro: cicflow_20250622_133904.csv
Data/Hora: 2025-06-27 15:58:29
Resultado: ALERT - 521 fluxos suspeitos (Bot)
Modelo: MLP
```

Figura 64 - Resultado do IDS em Tráfego com Elevado Número de Fluxos

Fonte desta figura: Captura de Ecrã do IDS no Ambiente de Testes.

Foi ainda realizado um teste semelhante, desta vez simulando um ataque do tipo DDoS direcionado a uma das máquinas da rede. A Figura 65 apresenta a análise de um ficheiro capturado neste cenário, contendo 32 fluxos de rede.

De acordo com a arquitetura do sistema, o modelo [MLP](#) foi aplicado à classificação dos fluxos e identificou 6 fluxos anómalos, associados a um ataque do tipo DDoS. Perante esta deteção, o sistema emitiu automaticamente um alerta ao utilizador.

```
[INFO] Ficheiro recebido: cicflow_20250623_155628.csv (32 fluxos)

» [MLP] A analisar ficheiro completo com Rede Neuronal...
MLP detetou 6 fluxos anómalos num total de 32
↳ Vários fluxos classificados como 'DDoS' pela MLP.

Ficheiro: cicflow_20250623_155628.csv
Data/Hora: 2025-06-27 16:00:43
Resultado: ALERT - 6 fluxos suspeitos (DDoS)
Modelo: MLP
```

Figura 65 - Resultado do IDS em Tráfego com Ataque DDoS Presente

Fonte desta figura: Captura de Ecrã do IDS no Ambiente de Testes.

3.5.2 Análise de Tráfego com Fluxo Normal

A Figura 66 apresenta os resultados da análise de um ficheiro contendo apenas fluxo de rede normal, recolhido no ambiente de testes.

De acordo com a arquitetura do sistema, o fluxo foi inicialmente analisado pelo modelo [MLP](#), que não detetou qualquer comportamento anómalo. A análise prosseguiu então com o modelo [AE](#), que classificou o fluxo como benigno, atribuindo-lhe um [MSE](#) de 89,9596, valor abaixo do limiar previamente definido.

Por essa razão, o resultado final foi “NO ACTION”, indicando que não foi necessário uma validação manual por parte do utilizador.

```
[INFO] Ficheiro recebido: cicflow_20250622_135746.csv (1 fluxos)

» [MLP] A analisar ficheiro completo com Rede Neuronal...
» MLP não detetou ataques. [AE] A analisar ficheiro com Autoencoder...

Resultados por fluxo (Autoencoder):
-----
[✓] Fluxo 001 | MSE = 89.9596 --> Benigno

Todos os fluxos analisados têm MSE abaixo do limiar. Considerado benigno.

Ficheiro: cicflow_20250622_135746.csv
Data/Hora: 2025-06-27 15:58:30
Resultado: NO ACTION
Modelo: Autoencoder
```

Figura 66 - Resultado do IDS em Tráfego Normal

Fonte desta figura: Captura de Ecrã do IDS no Ambiente de Testes.

3.5.3 Análise de Tráfego com Fluxo Anómalo

A Figura 67 apresenta os resultados da análise de um ficheiro contendo um fluxo correspondente ao ataque de exfiltração de dados realizado, recolhido no ambiente de testes.

Neste cenário, o modelo [MLP](#) não identificou qualquer anomalia, ativando o segundo nível de análise com o modelo [AE](#). Este classificou o fluxo como suspeito, atribuindo-lhe um valor [MSE](#) extremamente elevado (1138354383379,5376), ultrapassando largamente o limiar previamente definido.

O sistema solicitou validação manual ao utilizador, e após confirmação, o fluxo foi considerado malicioso, tendo sido emitido o respetivo alerta pelo [IDS](#).

```
[INFO] Ficheiro recebido: cicflow_20250622_154333.csv (1 fluxos)

» [MLP] A analisar ficheiro completo com Rede Neuronal...
» MLP não detetou ataques. [AE] A analisar ficheiro com Autoencoder...

Resultados por fluxo (Autoencoder):
-----
[Δ] Fluxo 001 | MSE = 1138354383379.5376 --> Suspeito

[VALIDAÇÃO MANUAL] Fluxos suspeitos detetados. É necessária validação por parte do utilizador.

s

1 fluxos validados como anomalia.

Ficheiro: cicflow_20250622_154333.csv
Data/Hora: 2025-06-27 15:58:30
Resultado: ALERT - 1 fluxos confirmados
Modelo: Autoencoder
```

Figura 67 - Resultado do IDS em Tráfego Anómalo

Fonte desta figura: Captura de Ecrã do IDS no Ambiente de Testes.

4 Discussão

Nesta secção procede-se à discussão dos resultados obtidos ao longo do desenvolvimento do projeto. Serão discutidos os pontos fortes, limitações e implicações práticas identificados, assim como as escolhas adotadas, que orientaram a conceção e implementação do [IDS](#). Este estudo visa também proporcionar uma compreensão aprofundada do impacto das decisões tomadas e das condições que influenciaram a eficácia do sistema, estabelecendo também as bases na identificação melhorias e trabalho futuro.

4.1 Dataset

A análise dos resultados obtidos na preparação e exploração do dataset permitiu confirmar que os dados utilizados fornecem uma base sólida para o desenvolvimento dos modelos de [ML](#), embora com algumas limitações a considerar no contexto prático do projeto.

Em primeiro lugar, a dimensão e estrutura do dataset final, com 140087 registros e 78 atributos, revelaram-se, por um lado, adequadas para o treino e teste dos modelos, assegurando diversidade suficiente de fluxos e características técnicas para a identificação de padrões. Por outro lado, esta dimensão poderia representar um risco para a generalização dos modelos e, em alguns casos, aumentar a probabilidade de ocorrência de overfitting.

A inexistência de valores nulos, confirmada após a limpeza dos dados, foi um aspecto positivo, garantindo a integridade da informação disponibilizada ao processo de aprendizagem.

A distribuição das classes apresentou um equilíbrio satisfatório entre as categorias de ataque BruteForce, DDoS, DoS e Bot, com 10000 registros cada. No entanto, destaca-se como limitação a reduzida representatividade da classe SQLInjection, que com apenas 87 amostras, representando menos de 0,1% do total, o que poderá dificultar a aprendizagem e generalização dos modelos relativamente a este tipo de ataque. Esta carência de dados poderá contribuir para uma menor capacidade do sistema em identificar com fiabilidade este tipo específico de intrusão.

Relativamente às estatísticas descritivas, os resultados mostraram grande diversidade nos valores dos atributos, como indicado pelos elevados valores de assimetria e curtose em alguns casos. Este fator poderá impactar a normalização dos dados e influenciar o comportamento dos modelos durante o treino, reforçando a importância de um pré-processamento cuidadoso e de um ajuste rigoroso dos algoritmos utilizados.

A análise da matriz de correlação evidenciou fortes relações entre vários atributos associados à temporização e duração dos fluxos, como Flow IAT Max, Idle Max, Flow IAT Std e Idle Mean, o que sugere alguma redundância nos dados, o que se pode considerar normal, dado o elevado número de registros. Ainda assim, a presença de atributos com baixa correlação com os restantes, como Fwd IAT Tot e Bwd IAT Tot, revelou-se relevante, uma vez que contribuem com informação complementar e menos redundante para os modelos.

Ainda no que diz respeito aos atributos, teria sido vantajoso remover features com baixa relevância no dataset, de modo a evitar tempos prolongados de treino e teste. Contudo, dada a presença de ataques com características semelhantes, como DDoS e DoS, optou-se por manter todos os atributos, permitindo que os modelos pudessem captar informações subtis ou menos relevantes que, ainda assim, pudessem ser determinantes para a predição de um ataque.

Desta forma, os resultados obtidos com o dataset permitiram construir uma base sólida, aplicável ao projeto e representativa para o treino dos modelos.

4.2 Modelos de Machine Learning

A análise dos resultados obtidos nos diferentes testes realizados com os modelos de [ML](#) permitiu identificar os pontos fortes, as limitações e as razões objetivas que conduziram às escolhas para os modelos finais e posteriormente para integração no [IDS](#).

4.2.1 Modelos Supervisionados

4.2.1.1 Rede Neuronal Multi-Layer Perceptron

A análise dos resultados obtidos com o modelo [MLP](#) permitiu identificar o impacto das diferentes configurações na capacidade do classificador para generalizar e detetar corretamente as diversas classes presentes no conjunto de dados. Verificou-se que as versões mais simples, como a do primeiro teste, embora conseguissem um desempenho perfeito nas classes maioritárias, evidenciaram limitações claras na deteção da classe minoritária (classe 5), refletidas no baixo valor de recall e na elevada taxa de falsos negativos.

A evolução introduzida nas arquiteturas seguintes, com destaque para o segundo teste, contribuiu para um equilíbrio mais adequado entre complexidade do modelo e desempenho, com ganhos notórios na identificação das amostras raras. O segundo teste revelou-se o mais eficaz, com melhorias significativas no recall e F1-score da classe 5, sem comprometer a performance global nas restantes categorias.

Os testes subsequentes, apesar de apresentarem arquiteturas mais complexas, não trouxeram ganhos adicionais relevantes, mostrando que o aumento da profundidade e do número de neurónios nem sempre se traduz numa melhor capacidade de generalização, especialmente quando o conjunto de treino já se encontra devidamente balanceado e o modelo otimizado. Este facto evidencia a importância de evitar o excesso de complexidade e o risco de overfitting em cenários com classes desbalanceadas.

A validação cruzada reforçou a robustez e consistência do modelo final, demonstrando a sua estabilidade perante diferentes subconjuntos dos dados.

Por estas razões a configuração utilizada no segundo teste foi a escolhida para integrar o modelo final [MLP](#).

4.2.1.2 Árvore de Decisão Random Forest

Relativamente ao modelo [RF](#), verificou-se um comportamento igualmente sólido, com excelente capacidade de generalização e estabilidade. Demonstrou-se a eficácia desta abordagem na classificação de tráfego de rede, com capacidade notável para generalizar mesmo

perante um cenário de classes desbalanceadas. Desde o primeiro teste, o modelo revelou uma forte aptidão para classificar corretamente as classes majoritárias, obtendo valores perfeitos nas principais métricas (precisão, recall e F1-score). Contudo, a classe minoritária (classe 5) apresentou, nos testes iniciais, limitações ao nível do recall, o que evidenciava a existência de falsos negativos a necessitar de mitigação.

À medida que se introduziram melhorias na configuração do modelo, observou-se um aumento progressivo no desempenho da classe 5, com destaque para os testes terceiro e quarto, onde se atingiram valores praticamente perfeitos de precisão e recall. O quarto teste destacou-se como o mais robusto, evidenciando um equilíbrio excepcional entre todas as classes, com uma taxa de erro mínima na classe minoritária.

A utilização de técnicas como o aumento do número de árvores, o ajuste de parâmetros como `max_features`, `min_samples_leaf` e `min_impurity_decrease`, bem como a ponderação das classes, contribuiu para reforçar a estabilidade e a capacidade do modelo em lidar com o desbalanceamento dos dados. O [OOB](#)-score consistente e elevado ao longo dos testes confirmou a boa generalização do modelo sobre dados não vistos.

A validação cruzada reforçou ainda mais a confiança na solução final, demonstrando a sua estabilidade e reduzida variabilidade do desempenho entre diferentes subconjuntos dos dados. A seleção da configuração do quarto teste como modelo final refletiu, assim, a escolha mais acertada, combinando elevada precisão, capacidade de generalização e robustez no tratamento de classes minoritárias.

4.2.1.3 Máquina de Vetores de Suporte

No que diz respeito ao modelo [SVM](#), verificou-se um comportamento positivo, embora com maior sensibilidade à configuração dos hiperparâmetros e ao desbalanceamento das classes. Desde os primeiros testes, o modelo apresentou resultados perfeitos nas classes majoritárias, mas revelou dificuldades na deteção da classe minoritária (classe 5), com valores baixos de recall e F1-score nos testes iniciais, refletindo uma taxa elevada de falsos negativos.

À medida que foram introduzidas afinações nos hiperparâmetros, especialmente no terceiro teste, observou-se uma melhoria no desempenho da classe 5, com um aumento no recall e F1-score, aproximando-se dos valores ideais. No entanto, o quarto teste demonstrou que alterações excessivas nos parâmetros, como o aumento de C e a redução de γ , não garantem necessariamente melhores resultados e podem conduzir a incoerências na

classificação, resultando em valores baixíssimos, como evidenciado nos indicadores dessa configuração.

A validação cruzada reforçou a estabilidade do modelo final, confirmando a sua capacidade de generalização em diferentes subconjuntos dos dados. A escolha do terceiro teste como configuração final refletiu o melhor equilíbrio entre desempenho global e capacidade de detecção da classe minoritária, evitando os desvios observados em outras configurações.

4.2.2 Modelos Não Supervisionados

4.2.2.1 *K-Means Clustering*

A análise dos resultados obtidos com o modelo [KM](#) revelou as limitações inerentes à aplicação de algoritmos não supervisionados na detecção de tráfego anômalo. O modelo demonstrou capacidade razoável na identificação da classe normal, mas dificuldades na detecção das amostras anômalas, refletidas em valores reduzidos de precisão, recall e F1-score nesta classe.

Apesar das otimizações introduzidas nos testes posteriores, como a aplicação de [PCA](#), seleção de atributos por variância e aumento do número de inicializações, não se verificaram melhorias significativas no desempenho do modelo. O primeiro teste, com configuração básica, acabou por apresentar os melhores resultados globais, com um ligeiro equilíbrio entre as classes e uma accuracy global de 0.65.

Estes resultados evidenciam as limitações da abordagem [KM](#) neste contexto, reforçando a dificuldade de segmentar padrões complexos de tráfego apenas com base na similaridade geométrica dos dados. Ainda assim, o modelo forneceu uma primeira visão sobre as abordagens seguintes mais apropriadas, sendo útil como referência comparativa no âmbito dos modelos não supervisionados.

4.2.2.2 *Isolation Forest*

No que diz respeito ao segundo modelo desenvolvido, o modelo [IF](#), a análise dos resultados obtidos ao longo dos diferentes testes permitiu identificar as dificuldades e limitações associadas à detecção de tráfego anômalo. Verificou-se, desde o primeiro teste, que o modelo apresentava uma capacidade limitada para identificar corretamente as amostras da classe anomaly, evidenciada pelos baixos valores de recall e F1-score dessa classe e pela elevada taxa de falsos negativos. Apesar de conseguir manter um desempenho consistente na classificação da classe normal, com valores de precisão e recall na ordem dos 0.70 e 0.90,

respetivamente, a eficácia global do modelo foi prejudicada pela fraca capacidade de detecção de anomalias.

As afinações introduzidas nos testes seguintes, incluindo ajustes dos hiperparâmetros e tentativas de melhorar o equilíbrio entre as classes, não permitiram alcançar melhorias significativas. O terceiro teste, escolhido como base para o modelo final, manteve os mesmos padrões de desempenho, reforçando a consistência do modelo na classe normal, mas sem avanços relevantes na identificação das amostras anómalas. O quarto teste evidenciou um maior equilíbrio entre as métricas das duas classes, mas à custa de uma descida geral no desempenho global, com impacto negativo na classificação da classe normal e na accuracy total do modelo.

Estes resultados demonstram as limitações do modelo [IF](#). Ainda assim, tal como no modelo anterior, a [IF](#) tornou-se útil como referência para comparação com outras diferentes abordagens não supervisionadas, contribuindo desta forma, para a necessidade de focar o desenvolvimento em modelos mais sofisticados, como o terceiro modelo desenvolvido, o [AE](#).

4.2.2.3 *Autoencoder*

No que diz respeito ao terceiro modelo desenvolvido, o [AE](#), a análise dos resultados obtidos ao longo dos diferentes testes revelou um desempenho superior ao dos modelos não supervisionados anteriores. Desde o primeiro teste, o modelo demonstrou já uma capacidade promissora, com a classe normal a registar uma precisão de 0.84, um recall de 0.95 e um F1-score de 0.89, enquanto a classe anomaly obteve uma precisão de 0.81, um recall de 0.54 e um F1-score de 0.65. Estes valores refletiram uma boa capacidade inicial de generalização e um equilíbrio razoável entre a identificação de tráfego normal e anómalo.

À medida que se avançou para os testes seguintes, foram observadas melhorias progressivas. O terceiro teste destacou-se ao apresentar uma precisão de 0.87 e um recall de 0.95 na classe normal, com um F1-score de 0.91, enquanto a classe anomaly registou uma precisão de 0.84, um recall de 0.66 e um F1-score de 0.74. Este teste evidenciou um avanço claro na capacidade de detecção de anomalias, com uma redução significativa da taxa de falsos negativos em relação aos testes anteriores.

O quarto teste revelou-se o mais robusto e com melhor desempenho, apresentando os melhores resultados globais. A classe normal obteve uma precisão de 0.91, um recall de 0.95 e um F1-score de 0.93. Já a classe anomaly atingiu uma precisão de 0.86, um recall de 0.78 e um F1-score de 0.82. Estes valores traduziram-se numa accuracy global de 0.90, com um F1-score médio ponderado de 0.90 e uma média macro de 0.88, confirmando o excelente desempenho

do modelo. Este resultado refletiu a melhor capacidade do [AE](#) em identificar amostras anómalas, enquanto manteve elevada precisão na classificação do tráfego normal.

Assim, o modelo final foi definido com base na configuração do quarto teste, garantindo um desempenho ótimo, como evidenciado pela precisão de 0.91 e recall de 0.95 na classe normal, e pela precisão de 0.86 e recall de 0.76 na classe anomaly, resultando num F1-score globalmente equilibrado de 0.89.

Relativamente ao teste de thresholds, verificou-se que as diferentes configurações não tiveram impacto significativo nos resultados, o que indica que o ajuste do threshold, isoladamente, não contribuiu para ganhos no desempenho do modelo e que a configuração final já se encontrava otimizada em termos de equilíbrio entre precisão e recall. No entanto, o teste revelou-se importante para a definição do limiar de decisão do modelo, uma vez que, até então, durante o processo de treino e testes, o valor sugerido era demasiado baixo face aos valores que posteriormente, no ambiente de testes, se mostraram mais elevados. Este facto levava o modelo a tomar decisões sistematicamente incorretas, o que foi corrigido com a escolha de um limiar mais ajustado.

De forma geral, o [AE](#) revelou-se uma solução eficaz para o problema em estudo, apresentando uma elevada capacidade de generalização, uma boa taxa de deteção de tráfego anómalo e um equilíbrio robusto entre as métricas das duas classes. Este modelo demonstrou ser o mais adequado no contexto das abordagens não supervisionadas desenvolvidas ao longo do projeto.

4.2.3 Escolha dos Modelos para o IDS

A análise dos resultados obtidos nos testes preliminares aos modelos supervisionados e não supervisionados revelou diferenças claras no desempenho e na capacidade de deteção de intrusões, que foram determinantes para a seleção dos modelos finais a integrar no sistema.

Os modelos supervisionados apresentaram elevada especificidade na identificação de fluxos benignos e de fluxos malicioso com ataques conhecidos, com exceção do modelo [MLP](#), que revelou ser mais sensível na deteção de fluxos suspeitos, nomeadamente em cenários com maior fluxo de tráfego. Por outro lado, os modelos [RF](#) e [SVM](#) mostraram menor sensibilidade a estes ataques, não detetando fluxos suspeitos em tráfego malicioso, o que corresponde a falsos negativos e limita a sua eficácia em situações reais de deteção. Assim, o modelo [MLP](#) sobressai como o mais adequado entre os supervisionados devido à sua maior capacidade de deteção,

ainda que com a ocorrência de falsos positivos, situação que se considera preferível face aos falsos negativos apresentados pelos outros modelos.

Por outro lado, os modelos não supervisionados apresentaram comportamentos distintos, sendo consideravelmente melhores na deteção de comportamento suspeito desconhecido, apesar de uma capacidade inferior à dos modelos supervisionados, conforme seria esperado. A [IF](#) evidenciou uma elevada taxa de deteção de fluxos suspeitos, classificando praticamente todos os fluxos como anómalos, o que sugere um número elevado de falsos positivos. O [AE](#) e o [KM](#), apesar de mais seletivos, apresentaram uma performance equilibrada, com deteção eficaz de fluxos suspeitos em determinados testes, sem classificar de forma errada todos os fluxos como anómalos. Contudo, o [KM](#) mostrou ser mais propício a gerar falsos positivos em tráfego benigno relativamente ao [AE](#). Aumentar o threshold do [KM](#) para reduzir estes falsos positivos poderia, por sua vez, aumentar os falsos negativos. Por esta razão, o [AE](#) revela-se a opção mais adequada para integrar o sistema final.

4.3 Escolha da Arquitetura do IDS

A disparidade de comportamento observado nos testes preliminares entre os modelos sugere que a integração conjunta de um modelo supervisionado, como o [MLP](#), com um modelo não supervisionado equilibrado, como o [AE](#), poderá melhorar a capacidade geral do sistema de deteção, conciliando uma boa capacidade de identificação de atividade maliciosa conhecida, à deteção de comportamento anómalo desconhecido, reduzindo, assim, a taxa de decisões incorretas.

Esta abordagem híbrida, demonstrada na Figura 5, permite tirar proveito das vantagens de ambos os tipos de modelos: a precisão dos supervisionados, baseada em aprendizagem com dados rotulados, e a capacidade dos não supervisionados em identificar padrões anómalos sem dependência direta de exemplos de ataque. Dessa forma, a escolha dos modelos para o sistema considerou não só os resultados obtidos nos testes preliminares, mas também a complementaridade funcional e o potencial para aumento da robustez na deteção de intrusões em ambientes reais.

Deste modo, a escolha da arquitetura de funcionamento do [IDS](#), baseia-se na combinação dos dois modelos ([MLP](#) e [AE](#)) de forma sequencial. O modelo supervisionado, atua como primeiro filtro, gerando alertas imediatos quando deteta intrusões já classificadas. Os fluxos que não são classificados como ataques seguem para o modelo não supervisionado, que

avalia comportamentos anómalos ou desconhecidos, utilizando um threshold. Quando o valor é ultrapassado, o fluxo é submetido a validação manual.

Esta escolha proporciona um equilíbrio eficaz entre rapidez, precisão e adaptabilidade, permitindo ao [IDS](#) responder tanto a ataques conhecidos como a comportamentos anómalos desconhecidos, reforçando a fiabilidade e robustez do sistema em ambientes dinâmicos e de constante evolução.

4.4 Ambiente de Testes

A análise dos resultados obtidos ao longo dos testes permitiu avaliar o desempenho do [IDS](#) implementado, destacando tanto os seus pontos fortes como as limitações identificadas durante a execução prática.

Do ponto de vista da interpretação dos resultados, observou-se que o modelo [MLP](#) demonstrou eficiência na deteção de anomalias associadas a ataques de negação de serviço. Curiosamente, nos testes com o ataque de SYN Flood, o mesmo padrão de tráfego foi, por vezes, classificado como Bot e, noutras ocasiões, como DDoS. Esta variação poderá estar relacionada com pequenas flutuações nos fluxos capturados, que influenciam a decisão do modelo de acordo com as características aprendidas durante o treino. Este comportamento, embora coerente dentro dos limites do modelo, reforça a importância de compreender como pequenas alterações nos dados podem afetar a categorização.

Em relação às expectativas versus realidade, o sistema demonstrou ser eficaz, detetando todos os ataques realizados no ambiente de testes. Em alguns casos, como nos ataques SYN Flood, a classificação foi direta e automática. Noutros, como o caso de exfiltração de dados com Netcat, a incerteza levou à ativação do módulo de validação manual, o que se revelou uma estratégia prudente. Importa sublinhar que, segundo os testes realizados, nenhum ataque passou despercebido (ou seja, nenhum foi rotulado com “NO ACTION”), o que reforça a fiabilidade do mecanismo de decisão composto.

Relativamente à ferramenta CICFlowMeter, esta desempenhou um papel crucial na ponte entre o tráfego capturado e os dados estruturados para o [IDS](#), utilizando as mesmas características (features) do dataset de treino. Contudo, essa vantagem veio acompanhada de uma limitação significativa e negativamente impactante: muitos ataques realizados no ambiente virtual não foram capturados pela ferramenta, resultando frequentemente em ficheiros .csv vazios. Esta situação dificultou a simulação prática de diversos cenários e reduziu a variedade de testes possíveis para serem posteriormente analisados pelo [IDS](#).

No que diz respeito ao desempenho e limitações, importa referir que o sistema não opera em tempo real, mas sim em quase tempo real. A análise apenas ocorre após a terminação manual do script.sh, responsável por finalizar a captação de pacotes e gerar o ficheiro .csv. Assim, o [IDS](#) depende da presença desse ficheiro para iniciar a sua análise. Apesar disso, quando confrontado com ataques previamente conhecidos, o sistema tem um desempenho sólido, apresentando classificações automáticas consistentes. Já nos ataques não conhecidos, há uma dependência notória do mecanismo de validação manual, frequentemente acionado pelo [AE](#) ao identificar anomalias. Ainda assim, é importante destacar que o sistema não ignora ataques desconhecidos, o que representa um fator positivo em termos de segurança e prevenção.

Com base na experiência, uma melhoria futura fundamental passaria pela substituição do CICFlowMeter por uma ferramenta mais eficiente e compatível com a análise em tempo real, de modo a maximizar o potencial do [IDS](#) e expandir a capacidade de deteção na prática.

Deste modo, os resultados obtidos demonstram um sistema funcional, seguro e adaptável, com espaço claro para evolução, sobretudo no que diz respeito à integração de ferramentas mais robustas e à redução da intervenção humana.

5 Conclusões

O desenvolvimento deste projeto permitiu atingir os objetivos propostos, nomeadamente a criação e implementação de um [IDS](#) capaz de identificar tráfego normal e anómalo, com o apoio de diferentes modelos de [ML](#). Ao longo das várias etapas, foi possível aplicar conhecimentos teóricos e práticos na conceção de um sistema funcional, seguro e validado em ambiente controlado.

A fase de contextualização teórica revelou-se fundamental para a definição da arquitetura do sistema e das abordagens mais adequadas a adotar, permitindo compreender as vantagens e limitações das diferentes técnicas e tipos de [IDS](#). A construção do dataset e a seleção dos modelos de ML foram etapas críticas para o sucesso do projeto, uma vez que a qualidade dos dados e a escolha das abordagens tiveram impacto direto no desempenho do sistema. No entanto, apesar da boa qualidade do dataset utilizado, verificou-se que a sua natureza acabou por limitar a robustez dos modelos, uma vez que a predição ficou dependente de características presentes e conhecidas no treino dos modelos. Esta limitação reforçou a necessidade de recorrer ao CICFlowMeter para gerar os .csv com dados provenientes do tráfego compatíveis com os modelos, mas trouxe também desafios, nomeadamente no que diz respeito à capacidade dos modelos em lidar com tráfego mais variado.

Concluímos que teria sido mais vantajoso treinar os modelos com diferentes conjuntos de dados, ou com um conjunto igualmente extenso, como o CSE-CIC-IDS2018, mas com uma maior diversidade de tráfego. Para além disso, uma abordagem faseada no treino, focada em subconjuntos de labels e nas respetivas features mais relevantes, poderia ter reduzido o tempo de treino e aumentado a precisão na identificação das características de cada tipo de tráfego, fosse ele normal ou associado a algum tipo de ataque. Uma outra melhoria seria preparar os modelos para lidar com situações em que partes do tráfego ou features estivessem em falta, algo que é expectável em redes reais e em cenários com atividade suspeita.

Relativamente aos modelos supervisionados, o [MLP](#) destacou-se pela sua capacidade de generalização, pela eficácia na deteção das diversas classes, incluindo as minoritárias, e pelo bom desempenho demonstrado no ambiente realista do [IDS](#). Os modelos não supervisionados apresentaram um desempenho mais modesto, como seria expectável face à complexidade dos padrões de tráfego e à ausência de dados rotulados. O [AE](#) evidenciou-se como a melhor abordagem não supervisionada, alcançando bons resultados na deteção de tráfego anómalo e demonstrando um equilíbrio robusto entre precisão e recall no modelo final e no ambiente de testes do [IDS](#).

A implementação do ambiente de testes com recurso ao VMware Workstation Pro permitiu simular cenários realistas e validar o desempenho dos modelos de forma segura e controlada. Ainda assim, a dependência do CICFlowMeter para a geração dos fluxos revelou várias limitações, em particular na captação de determinados tipos de ataques, o que condicionou a diversidade e riqueza dos dados obtidos durante os testes e, consequentemente, a avaliação completa do sistema.

Importa ainda referir que um dos requisitos iniciais do projeto era o uso de [IAGEN](#) para a criação de dados sintéticos. No entanto, após discussão em grupo e com o orientador, e tendo em conta a qualidade e diversidade do dataset utilizado, concluiu-se que não se justificava a geração de mais dados sintéticos para o treino dos modelos. Ainda assim, recorreremos à ferramenta ChatGPT [6], uma forma de [IAGEN](#), para nos apoiar na orquestração de diferentes formas de ataque no ambiente de testes. Apesar de muitos destes ataques não terem sido captados pelo CICFlowMeter, esta abordagem demonstrou a utilidade da [IAGEN](#) na identificação e planeamento de cenários de ataque, inclusive novos e desconhecidos.

Em termos globais, o projeto permitiu validar a viabilidade da solução proposta e identificar as suas principais limitações e pontos de melhoria. Entre estes destaca-se a necessidade de reforçar os mecanismos de deteção de ataques desconhecidos, integrar novas fontes de tráfego e preparar os modelos para lidar com dados incompletos ou parciais. O

trabalho realizado constitui assim uma base sólida para o desenvolvimento de futuras soluções de [IDS](#) mais robustas, precisas e adaptadas a ambientes reais e exigentes.

6 Trabalho futuro

O trabalho desenvolvido ao longo deste projeto abriu caminho para diversas possibilidades de evolução e melhoria do [IDS](#) implementado. Entre as principais sugestões de trabalho futuro destaca-se, em primeiro lugar, a necessidade de ampliar a diversidade dos dados utilizados no treino dos modelos. A integração de novos conjuntos de dados, com tráfego de rede mais variado e realista, poderá contribuir para o aumento da robustez dos modelos e para a sua capacidade de deteção de ataques desconhecidos ou menos comuns. Para além disso, a aplicação do [IDS](#) num contexto realista, associado a uma rede e a um modelo de negócio específico, permitiria definir com maior rigor os tipos de dados e as necessidades a serem atendidas, evitando a inclusão de elementos desnecessários.

Outra linha de evolução passa pela adoção de uma estratégia de treino faseado, em que os modelos sejam treinados de forma segmentada por tipos de tráfego ou classes de ataques, com seleção cuidadosa das features mais relevantes em cada fase. Esta abordagem poderá reduzir o tempo de treino, melhorar a precisão na deteção de padrões específicos e permitir a criação de modelos mais especializados e eficazes.

No que respeita ao ambiente de testes, será importante explorar a utilização de outras ferramentas de captação de tráfego, complementando ou substituindo o CICFlowMeter, de forma a superar as limitações identificadas e garantir uma maior cobertura de cenários de ataque, garantindo também o funcionamento total em tempo real. A construção de uma infraestrutura de rede mais realista e completa, que não se baseie apenas em três máquinas virtuais, constituiria também uma mais-valia. A integração do sistema num modelo de negócio concreto permitiria tornar o trabalho mais direcionado e aplicável. Para além disso, seria interessante explorar outras arquiteturas de [IDS](#), incluindo modelos Zero-Trust, zonas desmilitarizadas ([DMZ](#)), e soluções integradas com firewalls e outros mecanismos de defesa.

Outra área relevante de trabalho futuro passa pela adaptação dos modelos para lidar com dados incompletos ou parciais, preparando o [IDS](#) para situações realistas em que partes do tráfego possam estar ausentes ou corrompidas. Paralelamente, será importante estudar a integração de mecanismos de deteção em tempo real, com capacidade de resposta automática a determinados tipos de ataques, aproximando o sistema de soluções híbridas de deteção e prevenção.

Adicionalmente, seria interessante explorar a integração de uma API do ChatGPT da Open AI [7] na validação manual das decisões do [IDS](#), permitindo que a classificação como normal ou anômala pudesse ser validada automaticamente com base no tráfego capturado. Esta funcionalidade não foi implementada no presente projeto devido ao ambiente de rede privada, que não permitia o acesso à Internet durante os testes.

Por fim, seria importante e certamente uma melhoria, a investigação e o teste de novas abordagens de [ML](#), incluindo modelos baseados em deep learning e técnicas de ensemble, com vista a melhorar ainda mais a precisão, a generalização e a resiliência do sistema proposto. Embora já fora do âmbito direto do projeto, seria igualmente interessante testar o sistema com métodos de ethical hacking, analisando as suas vulnerabilidades e outras oportunidades de melhoria.

Bibliografia

- [1] UNB, “CSE-CIC-IDS2018 on AWS,” UNB, [Online]. Available: <https://www.unb.ca/cic/datasets/ids-2018.html>. [Acedido em 27 junho 2025].
- [2] VMware, “Fusion and Workstation,” VMware, [Online]. Available: <https://www.vmware.com/products/desktop-hypervisor/workstation-and-fusion>. [Acedido em 27 abril 2025].
- [3] Kali Linux, “Kali Linux Virtual Machines,” Kali Linux, [Online]. Available: <https://www.kali.org/get-kali/#kali-virtual-machines>. [Acedido em 2 maio 2025].
- [4] Visual Studio Code, “Download Visual Studio Code,” Visual Studio Code, [Online]. Available: <https://code.visualstudio.com/Download>. [Acedido em 27 junho 2025].
- [5] L. Hiêu, “CICFlowMeter,” Github, 2020. [Online]. Available: <https://github.com/hieulw/cicflowmeter>. [Acedido em 17 junho 2025].
- [6] OpenAI, “ChatGPT,” OpenAI, 2025. [Online]. Available: <https://chat.openai.com/>. [Acedido em junho 2025].
- [7] OpenAI, “OpenAI API Platform,” OpenAI, 2025. [Online]. Available: <https://openai.com/api/>. [Acedido em junho 2025].

Anexo 01 – Manual de Instruções/Instalação

1 Instalação do Python

Para o funcionamento do sistema é necessário ter o Python instalado. Recomenda-se a utilização da versão 3.10. Para instalar:

- 1) Aceder ao site oficial do Python: <https://www.python.org/downloads/>
- 2) Transferir o instalador adequado ao sistema operativo (Windows, Linux ou macOS).
- 3) Executar o instalador:
 - No Windows, marcar a opção Add Python to PATH antes de avançar na instalação.
 - Seguir os passos do assistente até à conclusão da instalação.

Para verificar se a instalação foi concluída com sucesso, abrir o terminal cmd (ou PowerShell) e executar o comando: `python --version`

Deverá ser apresentada a versão instalada.

2 Instalação das Bibliotecas

Para instalar as todas as bibliotecas encontra disponível, na pasta “IDS_ML”, o ficheiro “requirements.txt” que contém todas as bibliotecas necessárias. Para proceder com a instalação deverá abrir uma linha de comandos dentro do diretório da pasta “IDS_ML”. De seguida, deverá executar o comando: `pip install -r requirements.txt`. Por fim, só terá de aguardar até ao término da instalação.

3 Dataset

3.1 Instalação do Visual Studio Code

Instalar o Visual Studio Code a partir do site oficial [4] e transferir a versão Windows do instalador para o sistema.

3.2 Abrir Pasta com os Ficheiros do Dataset

Com o Visual Studio Code aberto, passar o rato sobre a opção “File” no friso superior. De seguida escolher a opção “Open Folder” e deverá escolher a pasta “IDS_ML” que deverá estar no diretório de “Transferências”, ou no diretório onde decidiu colocar a pasta.

Com a pasta já aberta no Visual Studio Code, de entre as diferentes pastas disponíveis, encontrará uma com o nome “Dataset”. Dentro desta pasta irá encontrar mais duas pastas (“CSE-CIC-IDS2018” e “Analise_Exploratoria”) e ainda o ficheiro “cleaned_ids2018.csv” e o ficheiro “prep_ids2018.py”.

Dentro da pasta “CSE-CIC-IDS2018” irá encontrar os ficheiros originais do dataset que contêm o tráfego de rede capturado e guardado nos diferentes dias correspondentes. O ficheiro “prep_ids2018.py” é o responsável por juntar todos estes ficheiros e fazer todo o tratamento e transformação dos dados, resultando no ficheiro final “cleaned_ids2018.csv”.

Dentro da pasta “Analise_Exploratoria” encontrará o ficheiro “analise_exploratoria.py” e os diversos ficheiros resultantes do correr deste ficheiro. O ficheiro “analise_exploratoria.py” é responsável por toda a análise feita ao ficheiro “cleaned_ids2018.csv”.

3.3 Gerar Ficheiro do Dataset

Para correr o ficheiro “prep_ids2018.py”, deve abrir uma linha de comandos, via cmd ou no próprio Visual Studio Code (aconselhamos no Visual Studio Code), no diretório onde colocou a pasta “IDS_ML”. De seguida deve escrever o comando `cd Dataset`, e já estando dentro do diretório respetivo, deverá escrever o comando `python prep_ids2018.py`. Ao terminar de correr, o ficheiro “cleaned_ids2018.csv” será criado.

Nota de Atenção: Para correr o ficheiro deverá ter o Python instalado. Caso não tenha, deve seguir o primeiro passo “Instalação do Python”, que encontra no início deste documento. Para além disso, deverá ter as bibliotecas necessárias instaladas, para tal, aconselhamos instalar previamente todas as bibliotecas necessárias para o funcionamento de todo o sistema, seguindo os passos do tópico “Instalação das Bibliotecas” que encontra no topo no documento.

3.4 Análise Exploratória do Dataset

Para esta e seguintes etapas, assumimos que já instalou o Python, todas as bibliotecas, e que já efetuou o passo anterior para criar o ficheiro “cleaned_ids2018.csv”. Caso contrário deverá fazê-lo.

Para proceder à análise exploratória do dataset deverá correr o ficheiro “analise_exploratoria.py” que se encontra dentro da pasta “Analise_Exploratoria”. Para isso, deve abrir uma linha de comandos, via cmd ou no próprio Visual Studio Code (aconselhamos no Visual Studio Code), no diretório onde colocou a pasta “IDS_ML”. De seguida deve escrever o comando `cd Dataset`, e já estando dentro do diretório respetivo, deverá escrever o

comando `python analise_exploratoria.py`. Ao terminar de correr, poderá ver diversos elementos avaliativos e estatísticos, quer no terminal, quer posteriormente nos ficheiros guardados na pasta “Analise_Exploratoria”.

4 Modelos de Machine Learning

O primeiro passo para poder correr cada modelo é compreender como estão organizadas as pastas e os ficheiros. Para tal, conforme a Figura 68, segue a exemplificação da estrutura e organização das pastas e ficheiros de forma genérica:

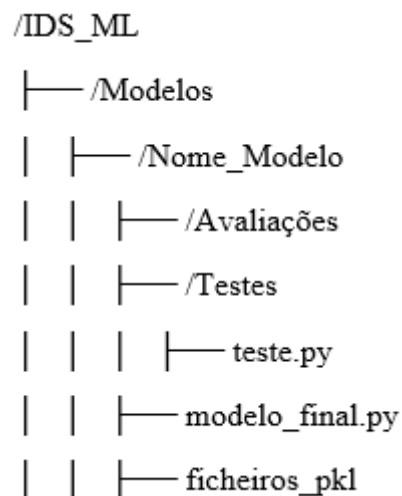


Figura 68 - Exemplificação da Estrutura e Organização das Pastas e Ficheiros (1)

Fonte desta figura: Elaboração Própria.

Tendo agora disponível a estrutura e organização das pastas e ficheiros dos Modelos, para correr qualquer um dos ficheiros Python relacionados, quer aos testes, quer aos modelos finais, deverá seguir os seguintes passos:

- 1) Abrir uma linha de comandos, via cmd ou no próprio Visual Studio Code (aconselhamos no Visual Studio Code), no diretório onde colocou a pasta “IDS_ML”.

- 2) Executar o comando `cd Modelos`

Já estando dentro da pasta Modelos, resta escolher qual o modelo que pretende executar.

Para tal, deverá seguir os seguintes passos genéricos:

- 1) Executar o comando `cd Nome_Modelo`

Caso queira executar o modelo final:

- 2) Executar o comando `python nome_ficheiro.py`

Caso pretenda correr algum dos ficheiros teste em cada modelo:

- 3) Executar o comando `cd Testes`
- 4) Executar o comando `python teste_num.py`

5 IDS

Neste ponto indicamos como executar o [IDS](#) caso pretenda utilizá-lo apenas no Visual Studio Code sem aceder ao Ambiente de Testes.

Para executar o [IDS](#) dentro do ambiente do Visual Studio Code, poderá fazê-lo de duas maneiras. A primeira é executando o [IDS](#) unicamente com os modelos supervisionados ou não supervisionados para predição. A segunda forma, é executando o [IDS](#) que segue a arquitetura definida, apresentada na Figura 5.

5.1 Execução do IDS por tipo de Modelo

O primeiro passo será compreender como estão organizadas as pastas e os ficheiros [IDS](#). Para tal, conforme a Figura 69, segue a estrutura e organização das pastas e ficheiros:

```
/IDS_ML
├── /IDS
│   ├── /Flow_outputs
│   ├── /Testes
│   │   ├── teste_supervised.py
│   │   ├── teste_unsupervised.py
│   └── gerador_fluxos.py
```

Figura 69 - Exemplificação da Estrutura e Organização das Pastas e Ficheiros (2)

Fonte desta figura: Elaboração Própria.

Tendo agora disponível a estrutura e organização das pastas e ficheiros [IDS](#), para correr qualquer um dos ficheiros Python disponíveis, deverá verificar se a pasta “Flow_outputs” contém ficheiros .csv. Se não estiverem disponíveis terá duas opções: a primeira será utilizar o próprio ambiente de testes para gerar estes ficheiros .csv com tráfego; a segunda opção é executar o ficheiro `gerador_fluxos.py`. Nesta segunda opção os ficheiros gerados são provenientes do dataset.

Para executar qualquer um dos ficheiro teste para testar o [IDS](#) por tipo de Modelo deverá seguir os seguintes passos:

- 1) Abrir uma linha de comandos, via cmd ou no próprio Visual Studio Code (aconselhamos no Visual Studio Code), no diretório onde colocou a pasta “IDS_ML”.
- 2) Executar o comando `cd IDS`
- 3) Executar o comando `python teste_supervised.py` ou `python teste_unsupervised.py`

5.2 Executar o IDS Utilizando a Arquitetura Definida

Para correr o ficheiro `ids.py`, deverá verificar se a pasta “Flow_outputs” contém ficheiros `.csv`. Se não estiverem disponíveis terá duas opções: a primeira será utilizar o próprio ambiente de testes para gerar estes ficheiros `.csv` com tráfego; a segunda opção é executar o ficheiro `gerador_fluxos.py`. Nesta segunda opção os ficheiros gerados são provenientes do dataset.

Para executar o ficheiro do [IDS](#) utilizando a arquitetura definida deverá seguir os seguintes passos:

- 1) Abrir uma linha de comandos, via cmd ou no próprio Visual Studio Code (aconselhamos no Visual Studio Code), no diretório onde colocou a pasta “IDS_ML”.
- 2) Executar o comando `cd IDS`
- 3) Executar o comando `python ids.py`

6 Ambiente de Testes

6.1 Instalação do VMware Workstation Pro

O primeiro passo consiste na instalação do VMware Workstation Pro [2]. Esta ferramenta de virtualização que permite a criação e gestão de múltiplas máquinas virtuais num só sistema anfitrião, para tal devemos ir ao site conforme ilustrado na Figura 70 e realizar o download do software.

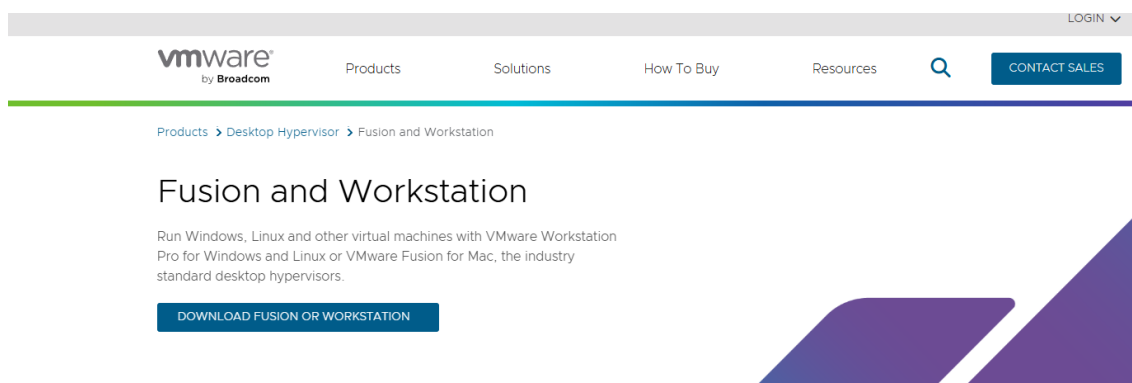


Figura 70 - Website do VMware

Fonte desta figura: Captura de Ecrã Retirada de [2].

6.2 Obtenção da Imagem do Kali Linux

De seguida, aceder ao site oficial do Kali Linux [3] e transferir a imagem especificamente preparada para utilização com o VMware Workstation Pro conforme ilustrado na Figura 71. Esta imagem já se encontra configurada com diversas ferramentas de cibersegurança.



Figura 71 - Website do Kali Linux

Fonte desta figura: Captura de Ecrã Retirada de [3].

6.3 Importação da Máquina Virtual Kali Linux

Com o VMware aberto, seleccionar a opção “Open a Virtual Machine” e escolhe-se o ficheiro “.vmx” da imagem descarregada conforme a Figura 72. A máquina é assim adicionada ao VMware.

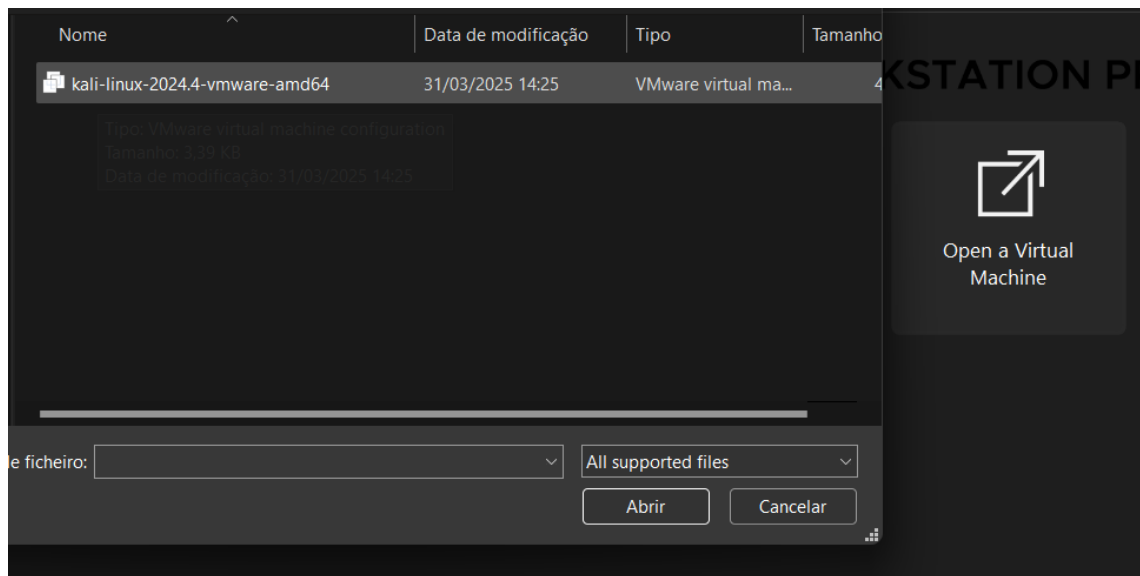


Figura 72 - Importação do Ficheiro .vmx

Fonte desta figura: Captura de Ecrã Retirada do VMware.

6.4 Clonagem da Máquina Virtual

Ao clicar com o botão direito na máquina criada, selecionar “Manage > Clone”. O assistente “Clone Virtual Machine Wizard” irá guiar o processo. Devem ser seguidos os seguintes passos:

- 1) Escolher “The current state in the virtual machine” como estado de clonagem.
- 2) Selecionar “Create a full clone” como tipo de clone.
- 3) Atribuir um nome à nova máquina.
- 4) Concluir o processo.

Estes procedimentos devem ser repetidos mais duas vezes, de forma a obter um total de três máquinas virtuais com Kali Linux.

6.5 Criação da Rede Virtual VMnet2

No VMware Workstation Pro, aceder a “Edit > Virtual Network Editor”. Após clicar em “Change Settings” (requer permissões de administrador), selecionar a interface “VMnet2” e:

- 1) Ativar a opção “Host-only”.
- 2) Ativar “Use local DHCP service to distribute IP address to VMs”.
- 3) Definir o “Subnet IP” como “192.168.10.0” e a “Subnet mask” como “255.255.255.0”.

Confirmar se as definições selecionadas vão ao encontro da Figura 73, de seguida aplicar as alterações e fechar o editor.

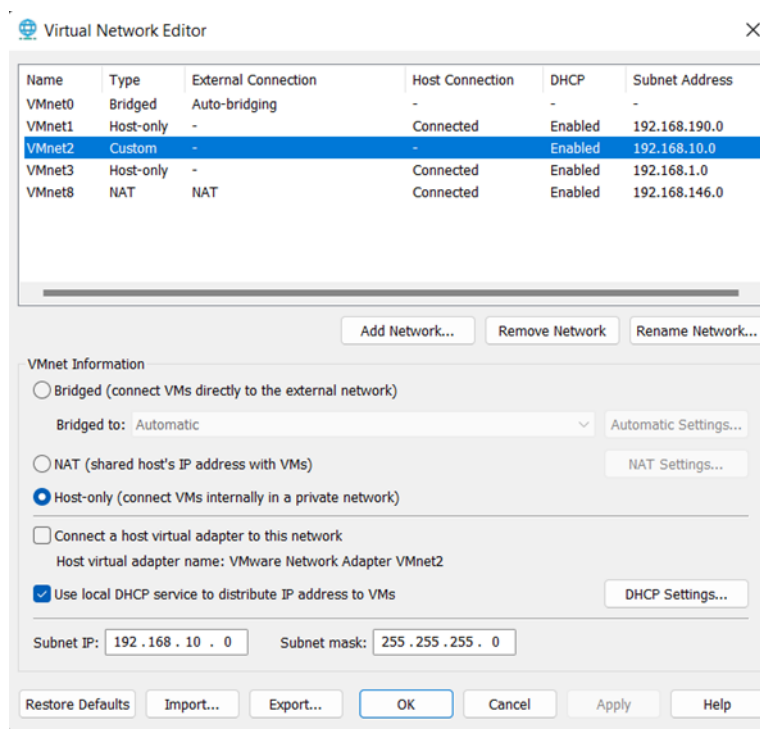


Figura 73 - Configuração da VMnet2

Fonte desta figura: Captura de Ecrã Retirada do VMware.

6.6 Configuração do Adaptador de Rede

Após ter as três máquinas prontas e a rede virtual “VMnet2” configurada, é necessário configurar a placa de rede virtual em cada computador. Em “Network Adapter”, escolher a opção “Custom: VMnet2”.

6.7 Acesso Temporário à Internet para Instalações

Na máquina IDS, alterar temporariamente a rede para “NAT”, de forma a garantir acesso à internet. Nesta fase, instalar o Visual Studio Code a partir do site [4] e transferir o instalador para o sistema, a opção escolhida é a “.deb” conforme se visualiza na .

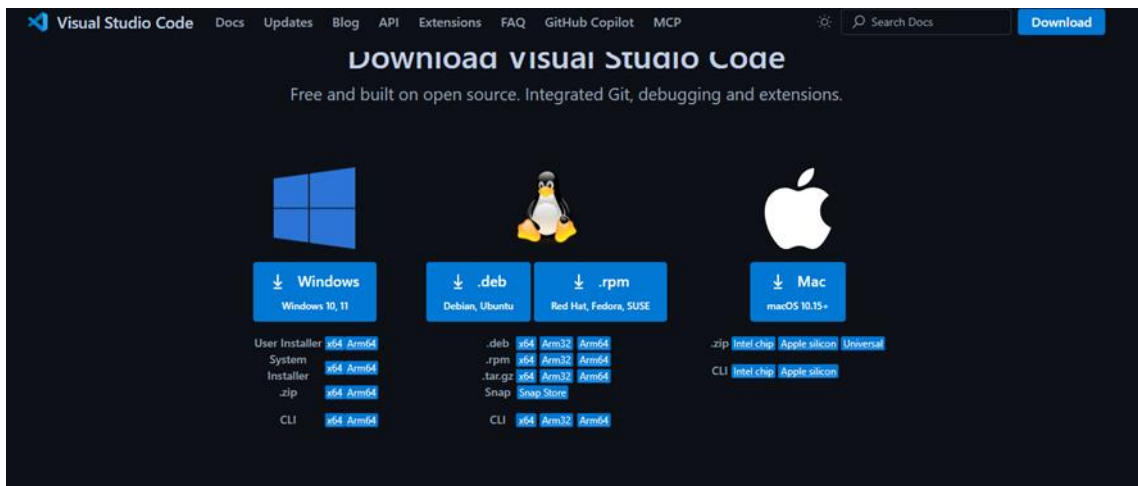


Figura 74 - Site do Visual Studio Code

Fonte desta figura: Captura de Ecrã Retirada de [4].

Por fim, abrir o terminal e seguir os seguintes passos:

- 1) Entrar no diretório de Downloads executando o comando `cd ~/Downloads`
- 2) Executar o comando `sudo apt install nome_ficheiro.deb`
- 3) Por fim, abrir o Visual Studio Code executando o comando `code`

6.8 Instalação do CICFlowMeter

Nota Prévia: De modo evitar possíveis erros, os comandos descritos nos passos 2) e 3) devem ser escritos de uma vez só no terminal.

- 1) Dentro da máquina virtual, abrir a linha de comandos.
- 2) Criar um ambiente virtual Python isolado:

```
Python3 -m venv /home/kali/cicflowmeter_vnv source
/home/kali/cicflowmeter_vnv/bin/activate
```

- 3) Instalar o “CICFlowMeter”:

```
git clone https://github.com/hieulw/cicflowmeter
cd cicflowmeter
pip install .
cd ~/Desktop
```

Uma vez instalado o “CICFlowmeter” no ambiente virtual Python, proceder ao uso do “script.sh” que automatiza os restantes passos para abrir o ambiente virtual sempre que se pretenda usar o “CICFlowmeter”, executar a ferramenta, guardar o “.csv” e por fim fazer uma copia para a pasta “Flow_outputs”.

- 4) Dar permissões de execução ao script:

```
chmod +x script.sh
```

```
sudo ./script.sh
```

6.9 Adicionar a Pasta IDS e Configurar alias

Arrastar a pasta do projeto [IDS](#) da máquina real para a pasta “Desktop” da máquina virtual [IDS](#).

- 1) Abrir o projeto no Visual Studio Code.

- 2) Dar permissões ao script:

```
chmod +x ~/Desktop/IDS/run_ids.sh
```

- 3) Copiar para o PATH e tornar executável:

```
sudo cp ~/Desktop/IDS/run_ids.sh ~/.local/bin/runids
```

```
sudo chmod +x ~/.local/bin/runids
```

- 4) Executar o script com:

```
runids
```

O script instalará todas as dependências e iniciará o [IDS](#), que ficará a monitorizar continuamente a pasta “Flow_outputs”.

6.10 Reconfigurar para a Rede VMnet2

Após a instalação, voltar a configurar, na máquina [IDS](#), o adaptador de rede para “Custom > VMnet2”, garantindo comunicação entre todas as máquinas do projeto que estão todas presentes na “VMnet2”.

6.11 Ativar o Modo Promíscoo na Interface eth0 da Máquina IDS

- 1) Executar o seguinte comando:

```
sudo ip link set eth0 promisc on
```

Desta forma a interface *eth0* fica em modo promíscoo.

- 2) Executar o seguinte comando:

```
ip link show eth0
```

Desta forma verificamos que efetivamente a interface “eth0” ficou em modo promíscoo.

6.12 Execução Final do Sistema

Com as três máquinas virtuais em execução, e todas ligadas à “VMnet2”, pode-se iniciar o script do “CICFlowMeter” e o “runids” na máquina virtual do [IDS](#).

Para realizar um teste de funcionamento do IDS deve-se com a máquina atacante agora realizar um ataque direcionado à máquina vítima, no final, deve, na máquina [IDS](#) ser parado com CTRL + C a execução do script do “CICFlowMeter” de modo a permitir que o ficheiro .csv seja escrito e guardado na pasta “Flow_outputs” que o nosso [IDS](#) está constantemente a monitorizar, depois de analisado o .csv pelo nosso [IDS](#), se o ficheiro contiver tráfego válido, o [IDS](#) irá processá-lo e classificar o conteúdo que passou na rede à instantes.

O [IDS](#) permanecerá ativo e em escuta de novas entradas na pasta “Flow_outputs” até que a sua execução seja manualmente interrompida com CTRL + C.