UNIVERSIDADE D
COIMBRA

Ângelo Huang

Ferramentas de machine learning na deteção de intrusões em redes de computadores

Dissertação no âmbito do Mestrado de Engenharia Eletrotécnica e de Computadores, Especialização em Computadores, orientada pelo Professor Tony Richard de Oliveira de Almeida e apresentada ao Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro de 2023

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Ferramentas de Machine Learning na deteção de intrusões em redes de computadores

Orientador:

Professor Tony Richard de Oliveira de Almeida

Júris:

Prof. Paulo José Monteiro Peixoto

Prof. Jorge Miguel Sá Silva

Prof. Tony Richard de Oliveira de Almeida

Dissertação apresentada em cumprimento parcial para o grau de Mestre em Ciências em Engenharia Eletrotécnica e de Computadores

Coimbra, Setembro 2023

Agradecimentos

Gostaria de expressar a minha mais profunda gratidão ao Professor Tony Richard de Oliveira de Almeida pelo apoio, orientação e preocupação ao longo do desenvolvimento desta dissertação.

Agradeço também aos meus pais por me darem a oportunidade de seguir os meus estudos, por sempre acreditarem em mim e por me darem apoio constante.

Gostaria também de agradecer ao meu colega e amigo Bernardo Alexandre dos Santos Costa Leite pelo apoio e motivação prestada.

Resumo

Hoje em dia, cada vez mais se usa dispositivos tecnológicos devido ao avanço tecnológico e como todas as outras coisas, acaba por ter uma parte negativa. Ameaças à segurança como ataques cibernéticos e intrusões maliciosas têm incrementado bastante durante estas décadas com vários motivos como por exemplo roubo de informações, extorsão ou interrupção de serviços. A deteção eficiente e precisa dessas ameaças é fundamental para garantir a integridade, confidencialidade e disponibilidade dos sistemas de informação.

Para resolver esses problemas, ferramentas de deteção, prevenção e proteção foram criadas. Mais recentemente, deu-se atenção aos modelos de *machine learning*. *Machine Learning* permite que os sistemas de segurança analisem grandes volumes de dados de rede, identificando padrões e comportamentos anómalos que podem indicar atividades intrusivas. Essas ferramentas também têm a capacidade de aprender com os dados e se adaptar a novos tipos de ataques, tornando-se mais eficazes ao longo do tempo.

O objetivo principal desta dissertação é criar um modelo híbrido com capacidade de detetar não só ataques já conhecidos mas também ataques desconhecidos. Comparou-se os resultados obtidos dos algoritmos mais comuns na área de *machine learning*. Utilizou-se o *dataset UNSW-NB15* que contém nove tipos de ataques, os algoritmos *Random Forest* e *K-means Clustering*, *Support Vector Machine*, *Multilayer Perceptrons* e redes neuronais convolucionais.

Os resultados obtidos com o modelo proposto vai de encontro a resultados semelhantes encontrados na literatura, variando ligeiramente devido às técnicas utilizadas e são indicadores dos modelos de *machine learning* poderem detetar com alguma precisão ataques informáticos. As técnicas de *machine learning* têm o potencial de aumentar significativamente a precisão na identificação de intrusões em comparação com métodos tradicionais como por exemplo *proxy servers*, antivírus ou *firewalls*.

Palavras chaves: *Machine Learning*, *Random Forest*, Redes Neuronais Convolucionais, *K-Means*, *Support Vector Machine*

Abstract

Nowadays, technological devices are increasingly being used due to technological advances, and like everything else, they also show a negative side. Security threats such as cyber attacks and malicious intrusions have significantly increased over the decades for various reasons, such as information theft, extortion, or service disruption. Efficient and accurate detection of these threats is crucial to ensure the integrity, confidentiality, and availability of information systems.

To address these problems, detection, prevention, and protection tools have been created. More recently, attention has been given to machine learning models. Machine learning allows security systems to analyze large volumes of network data, identifying patterns and anomalous behaviors that may indicate intrusive activities. These tools also have the ability to learn from data and adapt to new types of attacks, becoming more effective over time.

The main objective of this dissertation is to create a hybrid model with the ability to detect not only known attacks but also unknown ones. The results of the most common machine learning algorithms in the field were compared. The UNSW-NB15 dataset, which contains nine types of attacks, was used along with the Random Forest and K-means Clustering algorithms, Support Vector Machine, Multi-layer Perceptrons, and convolutional neural networks.

The results obtained with the proposed model are in line with similar results found in the literature, varying slightly due to the techniques used and are indicators that machine learning models can detect cyber attacks with some precision. Machine learning techniques have the potential to significantly increase accuracy in identifying intrusions compared to traditional methods such as proxy servers, antivirus or firewalls.

Keywords: Machine Learning, Random Forest, Convolutional Neural Networks, K-Means, Support Vector Machine

Conteúdo

Lista de Acrónimos

ANN	Rede Neuronal Artificial (*Artificial Neural Network*)
AODE	(*Average One Dependence Estimator*)
AUC	Área sob a Curva ROC (*Area Under the ROC Curve*)
BN	(*Bayesian Network*)
CNN	Rede Neuronal Convolucional (*Convolutional Neural Network*)
DGSOT	*Dynamic Growing Self-Organizing Tree*
DoS	*Denial of Service*
DT	Árvores de Decisão (*Decision Tree*)
FTP	Protocolo de transferência de ficheiro (*File Transfer Protocol*)
ICS	Sistema de controlo industrial (*Industrial Control System*)
IDS	Sistema de Deteção de Intrusões (*Intrusion Detection System*)
IRC	*Internet Relay Chat*
KNN	K-vizinho mais próximo (*K-Nearest Neighbour*)
ML	Aprendizagem Automatizada (*Machine Learning*)
MLP	(*Multilayer Perceptron*)
NB	(*Naive Bayes*)
RF	*Random Forest*
RNN	Rede Neuronal Recorrente (*Recurrent Neural Network*)

ROC	*Receiver Operating Characteristic*
R2L	*Remote to Local*
SMOTE	*Synthethic Minority Oversampling TEchnique*
SNMP	Protocolo de gestão de rede simples (*Simple Network Management Protocol*)
SVM	Máquina de Vetor de Suporte (*Support Vector Machine*)
U2R	*User to Root*

Lista de Figuras

Lista de Tabelas

1 Introdução

1.1 Motivação e contexto

A motivação para este estudo surge da crescente necessidade de proteger as redes de computadores contra ameaças cibernéticas e intrusões maliciosas. Com o avanço da tecnologia e a expansão das redes de comunicação, as organizações estão cada vez mais dependentes de sistemas de informação e, consequentemente, mais vulneráveis a ataques.

A cibersegurança, também conhecida como segurança digital, abrange todas as medidas adotadas para proteger informações importantes em computadores, redes, servidores e dispositivos móveis contra ameaças virtuais, como ataques cibernéticos, terrorismo cibernético, *phishing* e intrusões em redes de computadores. Nas últimas décadas, tem havido um aumento significativo nas intrusões em redes de computadores, devido à disponibilidade de ferramentas utilizadas para invasões com diversos propósitos, que vão desde extorsão, roubo ou ganho financeiro até sabotagem e desativação de infraestruturas.

Nesse contexto, investigadores e profissionais têm desenvolvido várias estratégias de prevenção e proteção, com uma atenção crescente aos modelos de *machine learning* (ML). O ML envolve o uso de algoritmos e técnicas estatísticas para ensinar um sistema computacional a realizar uma tarefa específica sem ser programado explicitamente para essa tarefa. Um modelo de ML é construído a partir de treinos, validações e testes usando amostras de dados. Esse modelo pode então ser aplicado para diferentes finalidades. O desempenho de um modelo de ML depende das características selecionadas dos dados utilizados no treino, assim como de diversos fatores, incluindo o algoritmo utilizado.

A eficiência de um modelo de ML é medida pela correta classificação, agrupamento ou previsão dos valores desejados. A precisão do modelo pode ser melhorada ao aumentar a quantidade de dados utilizados no treino ou ao aplicar transformações nos dados.

1.2 Objetivo

Esta dissertação tem como objetivo principal criar um modelo de *machine learning* híbrido com capacidade de detetar não apenas ataques conhecidos, mas também ataques desconhecidos de forma eficiente e precisa, utilizando os diferentes algoritmos e técnicas mais comuns. Começando por utilizar os algoritmos de aprendizagem supervisionada para treinar os ataques conhecidos e depois utilizar o *Kmeans* para agrupar e detetar ataques desconhecidos nos resultados classificados anteriormente como 'normal', isto é, tráfego normal. Ao alcançar este objetivo mais concreto, espera-se melhorar significativamente a capacidade de segurança de sistemas e redes, protegendo contra ameaças informáticas.

1.3 Estrutura do documento

A presente dissertação encontra-se estruturada da seguinte forma:

1. Capítulo introdutório com objetivo e motivação.
2. Revisão abrangente da literatura de algoritmos e técnicas de *machine learning*, *datasets*, tipos de ciberataques e métricas de avaliação para criar o modelo proposto.
3. Revisão e examinação de estudos, pesquisas e resultados nesta área para obter uma compreensão aprofundada do estado da arte.
4. Aplicação de ML na deteção de intrusões em redes de computadores. Implementação e treino de diferentes algoritmos utilizando o *dataset* escolhido. Essa metodologia inclui etapas como pré-processamento de dados, seleção de características (*features*) relevantes, escolha de algoritmos de ML e conjunto de dados (*dataset*). Consideração de diferentes hiperparâmetros e configurações para identificar os modelos mais eficazes na deteção de intrusões.
5. Análise e comparação do desempenho dos modelos em termos de precisão, exatidão (*accuracy*), *recall* e *f1-score*.
6. Conclusão do trabalho realizado e discussão do trabalho futuro.

2 Estado de Arte

Os investigadores tentam sempre encontrar os melhores métodos para obter um bom desempenho de um modelo de *machine learning* e neste capítulo apresenta-se alguns dos resultados obtidos durante os treinos e testes de combinações de várias técnicas, algoritmos, métricas e *datasets* referidas no capítulo anterior para detetar intrusões nas redes. Estes modelos baseiam-se em dois tipos de classificação: classificação binária e classificação multi-classe. Nas tabelas 2.1 e 2.2, encontram-se mais resumidos.

2.1 Classificação binária

A classificação binária é um tipo de classificação básica em que o objetivo é classificar em duas classes. As classes são normalmente representadas como 0 e 1, ou normal e ataque. Nesta secção, abordamos os trabalhos que utilizam a classificação binária para a deteção de intrusões nas redes.

Ren *et al.* (2019) [32], propuseram um IDS (*Intrusion Detection System*) utilizando uma otimização de dados híbridos que consiste em duas partes: *data sampling* e *feature selection*, denominada DO_IDS, com o objetivo de detetar anomalias. Durante *data sampling*, utilizaram a técnica *Isolation Forest (iForest)* para eliminação de *outliers*, o algoritmo genético (GA) para otimizar o rácio de amostragem e o classificador RF como critério de avaliação para obter o conjunto de dados de treino ideal. Na *feature selection*, utilizaram também GA e RF para obter o *subset* de *features* ideal. Após essas duas partes, um IDS baseado em RF é criado utilizando o *dataset* UNSW-NB15. Com o modelo proposto, obtiveram um *accuracy* de 0.928, precisão de 0.616, *recall* de 0.630 e *f1-score* de 0.623.

Mukrimah Nawir *et al.* (2018) [24], criaram um sistema de deteção de anomalias em rede usando o *dataset* UNSW-NB15. Para criar este modelo passaram por quatro etapas: preparação do *dataset*, treino e teste, construção do modelo de avaliação e avaliação das métricas de desempenho. Optaram pelos algoritmos NB (*Naive Bayes*), BN (*Bayesian Network*) e

AODE (*Average One Dependence Estimator*) para classificação. O algoritmo AODE foi o que obteve melhor *accuracy*, sendo este de 94.37%. Para os outros dois algoritmos, NB e BN, foi de 75.73% e 92.70%, respetivamente.

Nebrase Elmrabit *et al.* (2020) [9], testaram doze algoritmos de ML em termos da sua capacidade de detetar comportamentos anómalos numa rede. A avaliação foi feita em três *datasets*: UNSW-NB15, CICIDS 2017 e ICS (*Industrial Control System*) *cyberattack dataset*. Dos resultados obtidos, o RF foi o algoritmo que teve melhor *accuracy* em ambos os *datasets*, sendo estes 0.877, 0.999 e 0.928, respetivamente. O NB foi o algoritmo que obteve piores resultados.

Yin *et al.* (2017) [44], exploraram como modelar um sistema de deteção de intrusões baseado na aprendizagem profunda, e propuseram uma abordagem de aprendizagem profunda para a deteção de intrusões deteção de intrusões utilizando redes neuronais recorrentes (RNN-IDS). Mapearam as 41 *features* para 122 *features* como entrada e 2 nós de saída como classificação binária. Utilizaram 100 épocas em todos os treinos. Uma época pode ser definida como o número total de iterações de todos os dados de treino num ciclo para treinar o modelo de ML. Variaram os nós ocultos com 20, 60, 80, 120 e 140 e a taxa de aprendizagem com 0.01, 0.1 e 0.5. Obtiveram o seu melhor resultado, *accuracy* de 83.28%, com 80 nós ocultos e 0.1 de taxa de aprendizagem. Toda a abordagem foi feita com o *dataset* NSL-KDD.

Majjed Al-Qatf *et al.* (2018) [2], propuseram uma abordagem que foi utilizada para a aprendizagem de *features* e a redução da dimensão. Reduz consideravelmente o tempo de treino e de teste e melhora efetivamente a precisão da *accuracy* de SVM no que diz respeito a ataques. O seu modelo foi construído utilizando o mecanismo de AE (*Auto-Encoder*) esparso, que é um algoritmo de aprendizagem eficaz para reconstruir uma nova representação de *features* de forma não supervisionada. Após a fase de pré-treino, as novas *features* são introduzidas no algoritmo SVM para melhorar a sua capacidade de deteção de intrusões e a precisão da classificação. Com o modelo treinado, obtiveram uma *accuracy* de 84.96% usando o *dataset* NSL-KDD.

M A Jabbar *et al.* (2017) [15], propuseram um novo classificador *ensemble* para um sistema de deteção de intrusões. Esse novo modelo, RFAODE, é construído a partir de dois algoritmos conhecidos, RF e AODE. O desempenho do classificador proposto foi analisado com o *dataset* Kyoto, obtendo um *accuracy* de 90.51%.

Aksu *et al.* (2018) [1], de forma a detetar intrusões, utilizaram o *dataset* CICIDS 2017 e para a seleção das *features* usaram o algoritmo *Fisher Score*. Para classificação usaram

o SVM, KNN e DT, tendo obtido 0.5776, 0.9997 e 0.99 respetivamente. Com a redução das *features*, enquanto que o desempenho de KNN aumentou e o de DT manteve igual, o desempenho de SVM reduziu.

2.2 Classificação multi-classe

Nesta secção, apresenta-se trabalhos cujo os resultados são classificados como multi-classe. Na classificação multi-classe, os resultados previstos são classificados e distribuídos em várias classes disjuntas.

Ujwala Ravale *et al.* (2015) [31], propuseram uma combinação de técnicas de aprendizagem supervisionada e aprendizagem não supervisionada para criar um modelo híbrido utilizando o *dataset* KDD Cup99. Numa primeira etapa, utilizaram o algoritmo *K-means* agrupando instâncias de dados semelhantes com base nos seus comportamentos. Na etapa final, utilizaram uma SVM para classificação e obtiveram um *accuracy* de 87.01%.

YiHan Xiao *et al.* (2019) [42], desenvolveram um modelo utilizando o CNN com o *dataset* KDD Cup99 para deteção de intrusões. Na CNN, utilizaram duas camadas convolucionais com dois *max pooling's* e duas *fully-connected's* , obtendo assim, um *accuracy* de 94.00%.

Muhammad Shakil Pervez e Dewan Md. Farid (2014) [29], fizeram vários testes usando o algoritmo SVM e o *dataset* NSL-KDD com o objetivo de melhorar a competência de classificação de intrusão com um conjunto significativamente reduzido de *features* de entrada dos dados de treino. Começaram com 41 *features* na qual obtiveram 82.37% de *accuracy*, 0.72 de precisão, 0.82 de *recall* e 0.77 de *f1-score*. Foram reduzindo as *features* até 3, obtendo 78.85% de *accuracy*. Durante a redução das *features*, conseguiram obter o seu melhor resultado que foi de 82.68% de *accuracy* com 14 *features*.

Khan *et al.* (2007) [17], propuseram um novo tipo de IDS usando SVM e agrupamento hierárquico. Para agrupamento utilizaram o DGSOT (*Dynamic Growing Self-Organizing Tree*). Este algoritmo é uma rede neuronal auto-organizável com estrutura em árvore e foi concebida para descobrir a estrutura hierárquica correta num *dataset* subjacente. Enquanto com o SVM puro que teve uma *accuracy* de 57.6%, o novo modelo deles obteve 69.8%. Ambos os treinos usaram o *dataset* DARPA98-99.

Saqr Mohammed Almansob e Santosh Shivajirao Lomte (2017) [4], com o *dataset* KDD Cup99, propuseram um modelo com a dimensão das *features* reduzidas utilizando a técnica

PCA (*Principal Component Analysis*) e o algoritmo NB como classificador. À medida que foram reduzindo as *features*, obtiveram melhores resultados, conseguindo obter uma alta taxa de *accuracy* e *recall*. Sendo estes 98.53% e 98.82% respetivamente. Reduziram das 41 *features* para 8 *features*.

Shah *et al.* (2021) [34], aplicaram uma abordagem usando o SVM e RF para criar um IDS. Esses algoritmos foram implementados para classificar o *dataset* NSL-KDD. Implementaram também uma abordagem de GA (*Genetic Algorithm*) para selecionar as melhores *features* do conjunto original. Com o SVM obtiveram 0.711 de *accuracy* e 0.865 com o RF.

Tabela 2.1: Trabalhos realizados por outros autores

Ref.	**Algoritmo(s)**	***Dataset***	**Resultado(s)**	**Classificação**	**Ano**
[32]	GA+RF	UNSW-NB15	*accuracy*=0.928 precisão=0.616 *recall*=0.630 *f1-score*=0.623	binária	2019
[24]	NB	UNSW-NB15	*accuracy*=75.73%	multi-classe	2018
	BN		*accuracy*=92.70%		
	AODE		*accuracy*=94.37%		
[31]	*K-means* e SVM	KDD Cup99	*accuracy*=87.01%	multi-classe	2015
[9]	RF	CICIDS 2017	*accuracy*=0.999	binária	2020
[1]	KNN	CICIDS 2017	*accuracy*=0.9997 precisão=0.9968 *recall*=0.9985 *f1-score*=0.9997	binária	2018
	SVM		*accuracy*=0.5776 precisão=0.5654 *recall*=0.8097 textitf1-score=0.6564		
	DT		*accuracy*=0.99 precisão=0.99 *recall*=0.99 textitf1-score=0.99		

Tabela 2.2: Trabalhos realizados por outros autores (continuação da tabela 3.1)

Ref.	**Algoritmo(s)**	***Dataset***	**Resultado(s)**	**Classificação**	**Ano**
[42]	CNN	KDD Cup99	*accuracy*=94.00%	multi-classe	2019
[29]	SVM	NSL-KDD	*accuracy*=82.37% precisão=78.85% *recall*=0.82 *f1-score*=0.77	multi-classe	2014
[17]	SVM+DGSOT	DARPA98-99	*accuracy*=69.8%	multi-classe	2007
[44]	RNN	NSL-KDD	*accuracy*=83.28%	binária	2017
[2]	AE esparso e SVM	NSL-KDD	*accuracy*=84.96%	binária	2018
[15]	RFAODE	Kyoto	*accuracy*=90.51%	binária	2017
[4]	NB	KDD Cup99	*accuracy*=98.53% *recall*=98.82%	multi-classe	2017
[34]	SVM+GA	NSL-KDD	*accuracy*=0.711 precisão=0.742 *recall*=0.493 *f1-score*=0.520	multi-classe	2021
	RF+GA		*accuracy*=0.865 precisão=0.69 *recall*=0.599 textitf1-score=0.570		

3 Conceitos e definições

3.1 Algoritmos e técnicas de *Machine Learning*

O ML, ou aprendizagem automatizada, consiste fundamentalmente na utilização de dados e algoritmos computacionais que recriam a aprendizagem humana a partir de exemplos e experiências (dados experimentais). Existem vários algoritmos de *machine learning* que podem ser divididos em três categorias: aprendizagem supervisionada (*supervised learning*) [6], aprendizagem não supervisionada (*unsupervised learning*) [8], e aprendizagem por reforço (*reinforcement learning*) [20]. Cada um destes tipos de algoritmo se distinguem pelo conjunto de dados disponíveis para treinar o algoritmo a tomar uma decisão. A partir da figura 3.1, é possível compreender melhor como ML está dividido e ainda, é possível ver quais os algoritmos mais comuns em cada uma das categorias.

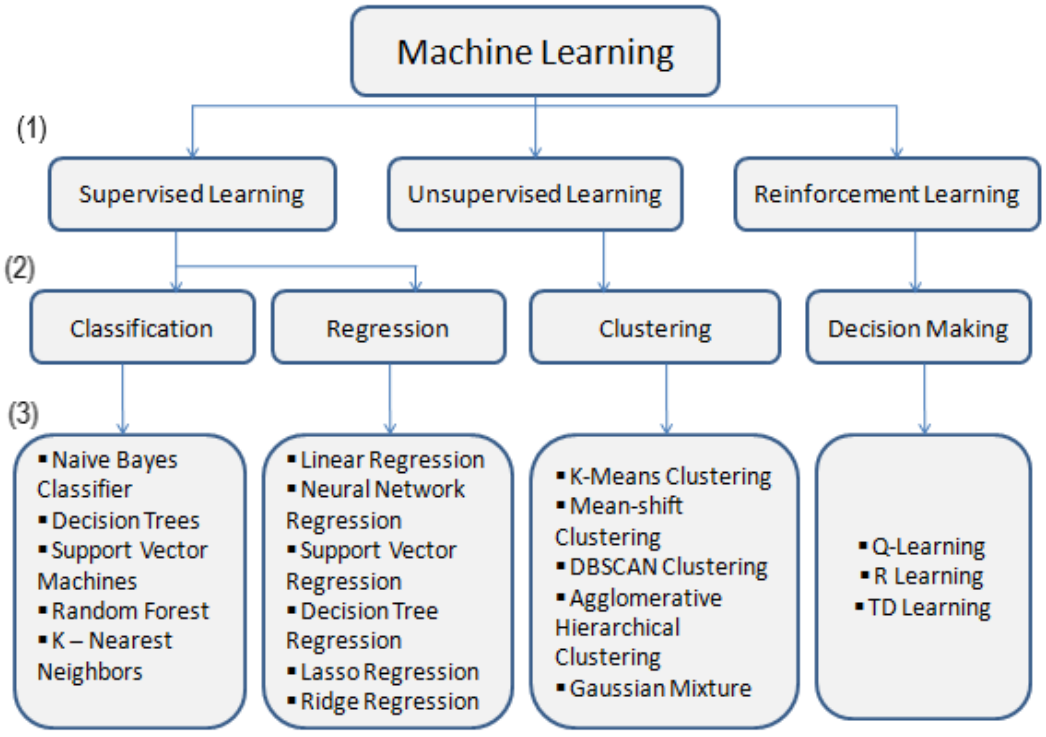


Figura 3.1: Categorias de algoritmos em ML: (1) Categoria; (2) Função; (3) Algoritmos

O *machine learning* é atualmente muito popular, com uma extensa aplicação em diversas áreas que vão do reconhecimento de imagens até a deteção de fraudes bancárias, passando por aplicações tão diversas como o reconhecimento de voz e carros autónomos. Na área da cibersegurança, o *machine learning* tem ganho nos últimos anos alguma notoriedade na deteção de situações de intrusão e ataques informáticos [5]. A seguir apresenta-se alguns conceitos e definições de algoritmos e técnicas de *machine learning* mais comuns e usados.

3.1.1 Algoritmo *K-Nearest Neighbour*

O algoritmo *K-Nearest Neighbour (KNN)*, em português K-vizinho mais próximo, é um método de aprendizagem supervisionado não paramétrico [45] que utiliza todo o conjunto de dados de treino como modelo, sem aprender explicitamente quaisquer parâmetros. O KNN funciona com base no princípio de que instâncias semelhantes são prováveis de ter *labels* ou valores semelhantes.

Um algoritmo não paramétrico é um tipo de algoritmo que não faz suposições sobre a distribuição subjacente dos dados. Diferente dos algoritmos paramétricos, que assumem uma forma específica para a distribuição dos dados (como a regressão linear), os algoritmos não paramétricos buscam estimar a distribuição de forma mais flexível, sem restrições prévias. É utilizado para classificações e regressões. Em ambos os casos, as entradas consistem nos k exemplos de treino mais próximos num conjunto de dados (*dataset*) e a saída depende do tipo de abordagem em que o algoritmo KNN é usado:

- na classificação, a saída é um membro da classe. Utiliza aproximações para classificar ou predições sobre o agrupamento de um ponto de dado individual.
- na regressão, aproxima a associação entre variáveis independentes e o resultado contínuo pela média das observações na mesma vizinhança.

Em conclusão, a regressão tenta prever o valor da variável de saída usando uma média local e a classificação tenta prever a classe à qual a variável de saída pertence, calculando a probabilidade local. Na figura 3.2 podemos ver um exemplo de aplicação do KNN para classificação.

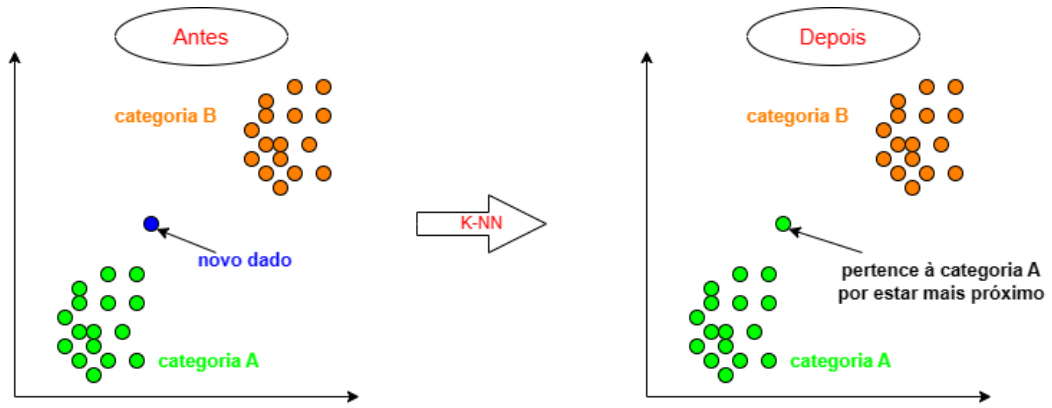


Figura 3.2: Exemplo de classificação usando um algoritmo KNN

3.1.2 *K-means Clustering*

K-means clustering (agrupamento) é um algoritmo de aprendizagem não supervisionada que descobre padrões de dados não conhecidos e agrupa-os em grupos cujos dados são semelhantes, como exemplificado na figura 3.3. O número de grupos define-se como k [19].

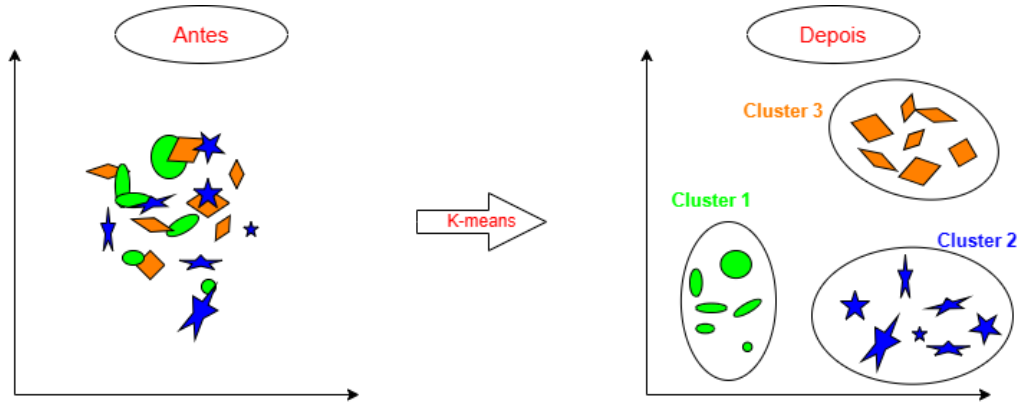


Figura 3.3: Exemplo de aplicação do *k-means clustering*

De forma resumida, o algoritmo *k-means clustering* pode ser descrito como:

1. escolhem-se aleatoriamente k pontos que se transformam em centroides nos *clusters*
2. movem-se esses centroides até ao ponto médio dos dados num *cluster*, isto é, o algoritmo calcula a média de todos os pontos num *cluster* (centroide) e move-os até ao local dessa média
3. o processo é repetido até não haver alterações (os dados não passam de um lado para o outro)

Na figura abaixo 3.4 podemos ver como dois grupos são separados com o algoritmo. No passo 1, é atribuído dois pontos aleatórios. De seguida esses pontos são movidos para o centro dos dados que é mostrado no passo 2. Nos passos 3, 4 e 5 são repetições dos passos anteriores e, até que, finalmente chega ao passo 6, quando os dados estão separados em dois grupos.

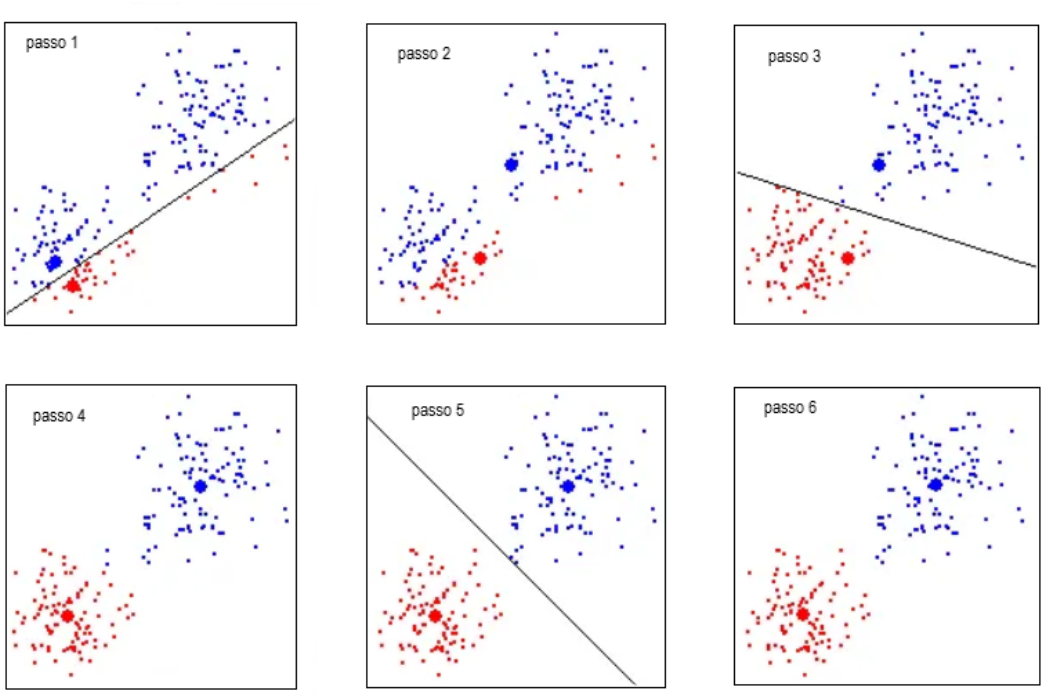


Figura 3.4: Aplicação do *k-means clustering* passo a passo

3.1.3 Árvores de decisão

Árvores de decisão, que na literatura inglesa comummente designada por *Decision Tree* (DT) é um tipo de aprendizagem supervisionada. É utilizado como um modelo preditivo para tirar conclusões sobre um conjunto de dados. Apresenta uma estrutura baseado numa árvore onde é desenhado de cima para baixo com a raiz no topo e dividindo-se em dois nós e, assim repetidamente. Cada nó faz uma pergunta para avaliar até chegar a uma conclusão, podendo ser vista um exemplo na figura 3.5.

As DT podem ser de classificação, com o objetivo de classificar os dados num *dataset*, ou de regressão, que podem prever continuamente os dados. Em ambos os casos, pode ocorrer um problema de *overfitting*. *Overfitting*, em português ajuste excessivo, é um comportamento de *machine learning* indesejável que ocorre quando um modelo é mais preciso para dados de treino do que para novos dados, isto é, adapta-se ao dados de treino e não consegue fazer generalizações dos novos dados. Nas DT, isto acontece devido ao número elevado de dados

Figura 3.5: Exemplo de uma árvore de decisão

para avaliar e para tentar resolver pode-se limitar o tamanho da árvore ou definir um valor mínimo em cada nó [18].

3.1.4 *Random Forest*

Random Forest ou *Random Decision Forest* (RF) é um método de aprendizagem *ensemble* para classificações e regressões. Diz-se que é um método de aprendizagem *ensemble* quando um algoritmo combina as previsões de dois ou mais modelos. À semelhança da DT, RF utiliza uma infinidade de árvores de decisão durante os treinos [21]. Para classificação, a saída da RF é a classe selecionada pela maioria das árvores, tal como exemplificado na figura 3.6. Para a regressão, é retornada a média ou a previsão média das árvores individuais, sendo um dos exemplos, a figura 3.7.

3.1.5 Máquinas de Vetores de Suporte

A Máquina de Vetores de Suporte (*Support Vector Machine*, SVM) é um algoritmo de aprendizagem supervisionada usado para problemas de classificação e regressão [30]. O SVM traça um hiperplano mais adequado que contém uma máxima margem possível que separa os dados em duas classes para que novos dados sejam previstos e pertencerem a uma. A figura 3.8 é um exemplo de aplicação do SVM. Para ser mais eficiente, usa uma técnica chamada *kernel trick* que é um método simples em que dados não lineares são projetados num espaço de dimensão superior (2D para 3D) para facilitar a classificação dos dados onde podem ser divididos linearmente por um plano. É possível visualizar um exemplo na figura 3.9.

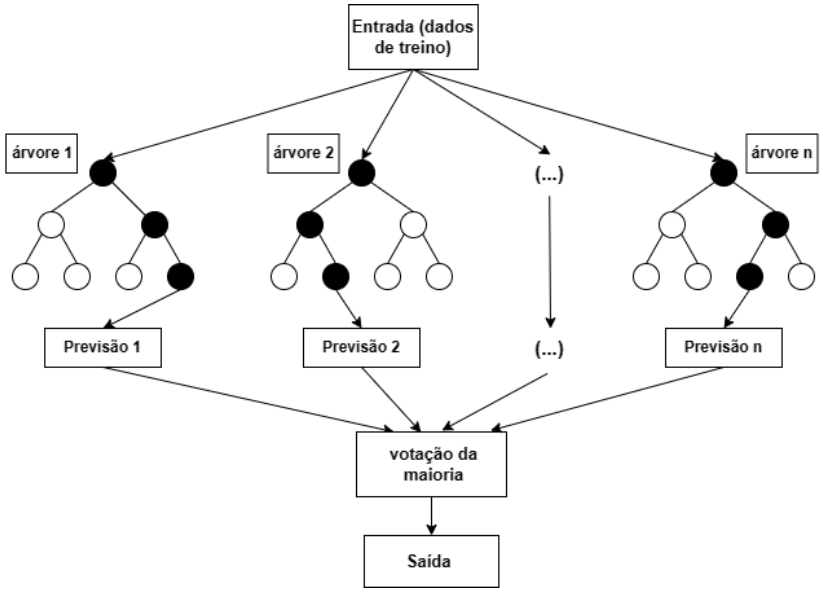


Figura 3.6: Exemplo de RF para classificação

3.1.6 Redes Neuronais Artificiais

Uma rede neuronal artificial (ANN) é um método em inteligência artificial que permite aos computadores a processar dados de uma maneira inspirada no cérebro humano [13]. Faz parte de um tipo de processo de *machine learning* chamado aprendizagem profunda (*deep learning*) que usa nós ou neurónios interconectados numa estrutura em camadas que se assemelha ao cérebro humano, como exemplificado na figura 3.10a. Uma rede neuronal artificial geralmente é composta por três tipos de camadas: camada de entrada, camadas ocultas (*hidden layers*) e camada de saída. Os dados de entrada são usados na camada de entrada, que repassa as informações para as camadas ocultas. As camadas ocultas processam as informações e passam-nas para a camada de saída, onde é gerada a resposta da rede. Um dos exemplos disso mesmo pode ser visto na figura 3.10b.

Existem vários tipos de redes neuronais, cada uma com a sua própria arquitetura e aplicação específica. Aqui estão alguns dos tipos mais comuns:

- ***Perceptron*** [16] **:** É a rede neuronal mais simples que existe. Introduzido no final da década de 1950 pelo psicólogo *Frank Rosenblatt*. É composta por um único neurónio e é usado principalmente para problemas de classificação binária. Recebe múltiplas entradas, aplica pesos a essas entradas e produz uma saída com base numa função de ativação específica (funções de ativação que têm como saída 0 ou 1, tais como *sigmoid*, tangente hiperbólica, *step function*, etc.). Na figura abaixo 3.11, encontra-se uma rede neuronal artificial com *perceptron*.

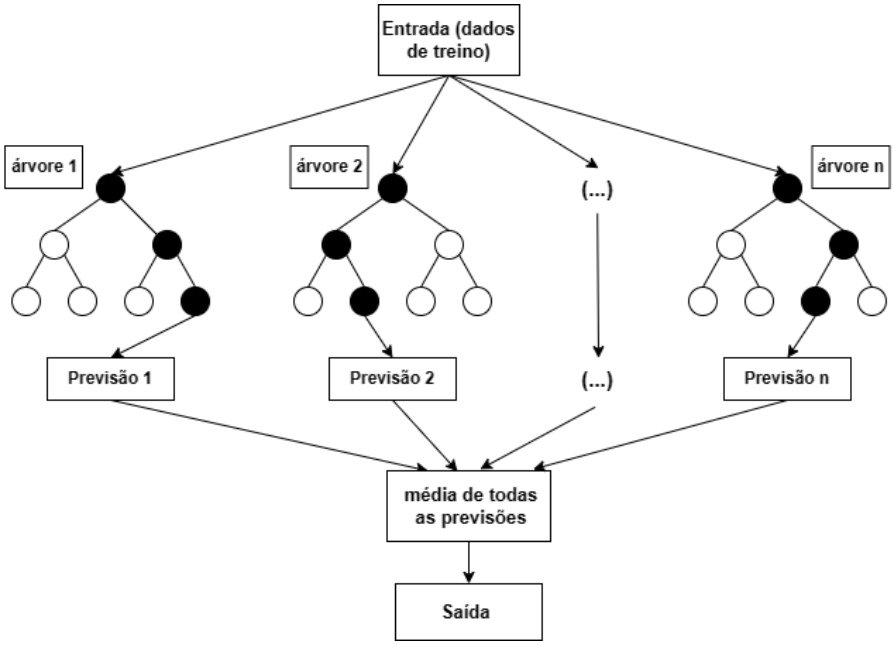


Figura 3.7: Exemplo de RF para regressão

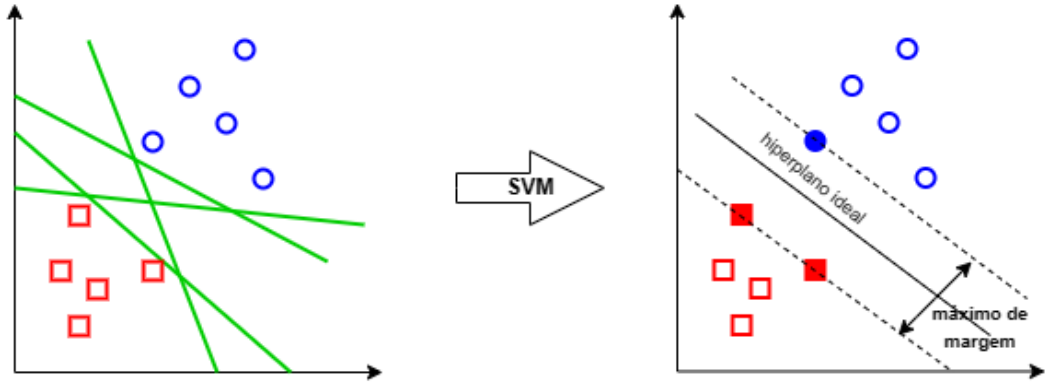


Figura 3.8: Exemplo de aplicação do SVM

- ***Multilayer Perceptron* (MLP) [25] :** O MLP é uma arquitetura de rede neuronal artificial composta por várias camadas de neurónios, em que, cada neurónio é semelhante a um *perceptron*. Inclui também uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída, figura 3.12. É uma arquitetura flexível e poderosa, capaz de aprender e modelar relações complexas nos dados. É amplamente utilizado para problemas de classificação e regressão, como reconhecimento de padrões, classificação de imagens, processamento de linguagem natural, previsão de séries temporais, entre outros.

- ***Convolutional Neural Network* (CNN) [41] :** As CNNs são uma classe de redes neuronais artificiais projetadas principalmente para processar dados de formato estruturado, como imagens e vídeos. São bastante eficazes em tarefas de visão de

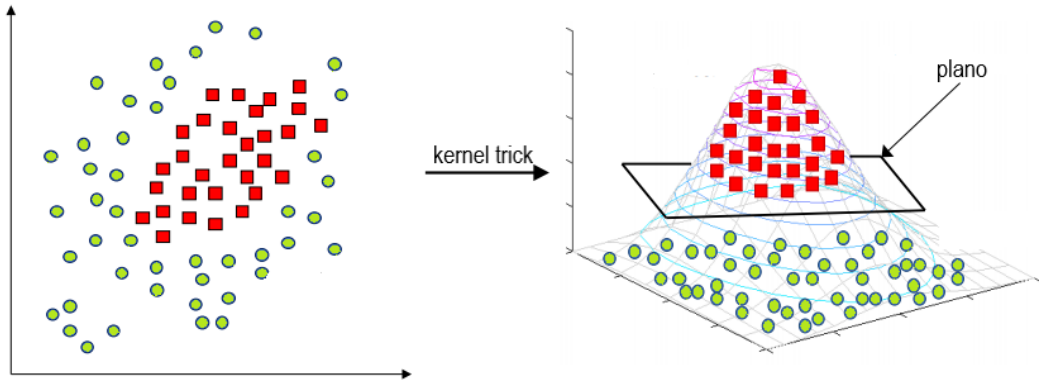


Figura 3.9: Aplicação da técnica *kernel trick*

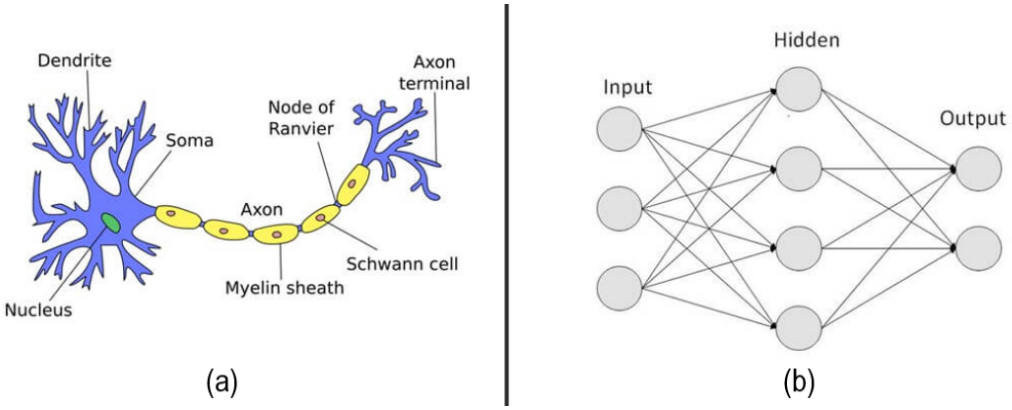


Figura 3.10: Redes Neuronais: (a) Rede neuronal humano; (b) Rede neuronal artificial

computadores, como classificação de imagens, deteção de objetos, segmentação e reconhecimento facial. A principal característica das CNNs é a utilização de camadas de convolução, que são responsáveis por realizar operações de convolução nas entradas, permitindo que a rede extraia características locais, como bordas, texturas e padrões, de forma eficiente. Na figura 3.13 encontra-se um exemplo de uma CNN. Além das camadas de convolução, as CNNs também podem incluir camadas de *max pooling*, que reduzem a dimensão dos mapas de características, mantendo as informações mais relevantes, como se pode ver na figura 3.14. Essas camadas ajudam a tornar a rede mais robusta a pequenas variações na posição dos objetos nas imagens. Durante as operações convolucionais, utilizam a técnica denominada de *padding*, representada na figura 3.15, que ajuda a manter a dimensão dos mapas de características reduzindo a perda de informação. Normalmente acrescenta-se zeros à borda do mapa de características.

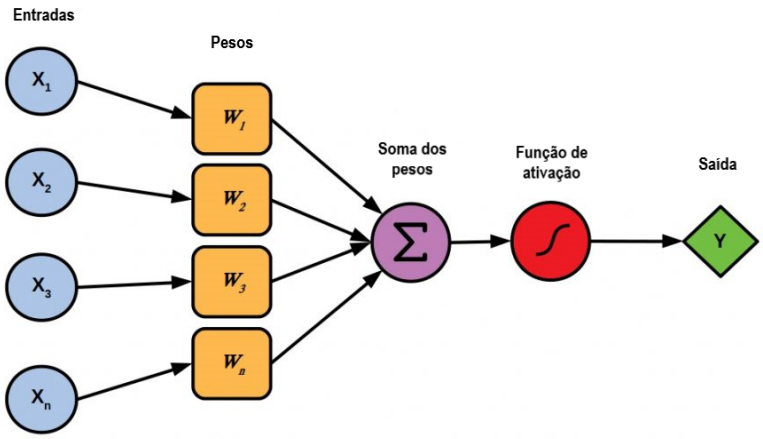


Figura 3.11: Rede neuronal artificial com *perceptron*

- ***Recurrent Neural Network* (RNN)** [12] **:** As RNNs são uma classe de redes neuronais artificiais projetadas para processar dados sequenciais, nos quais a ordem das informações é importante, como séries temporais, modelos de previsão e modelos de linguagem. São especialmente adequadas para lidar com dados com dependências temporais e estruturas sequenciais. A principal característica das RNNs é a capacidade de manter informações de estados anteriores e utilizá-las para processar informações subsequentes. Isso é alcançado através da introdução de conexões recorrentes, onde os neurónios de uma camada têm conexões que retornam a eles mesmos ou a neurónios de camadas anteriores. Essas conexões permitem que as RNNs tenham uma "memória" interna que mantém informações sobre os estados anteriores da sequência. Na figura 3.16 é apresentado um exemplo de uma RNN básica. Esta RNN, possui uma entrada X, uma camada oculta e uma saída. Em cada passo de tempo, a RNN recebe uma entrada, que pode ser um elemento da sequência, como uma palavra numa frase ou um valor numa série temporal. Na camada oculta, o estado oculto (ou estado da memória) é atualizado a cada passo de tempo e armazena informações sobre as entradas anteriores da sequência. Esse estado oculto num determinado passo de tempo é calculado com base na entrada atual e no estado oculto anterior. Após de ser calculado, o estado oculto é então usado para gerar a saída naquele passo de tempo. A saída pode ser usada para previsões ou classificações, dependendo da aplicação.

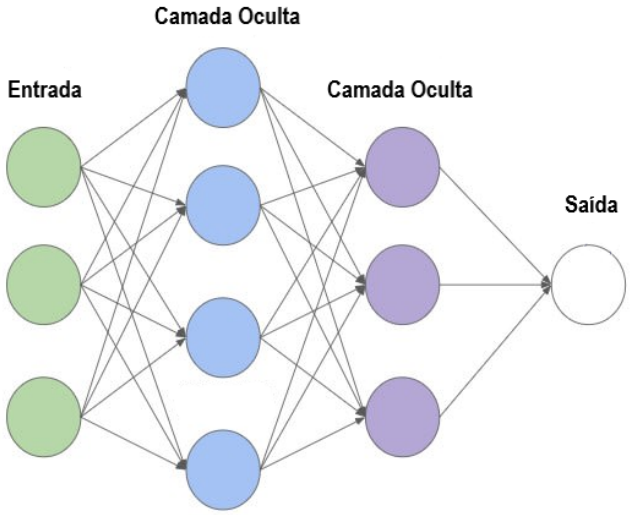


Figura 3.12: Rede neuronal artificial com *multilayer perceptron*

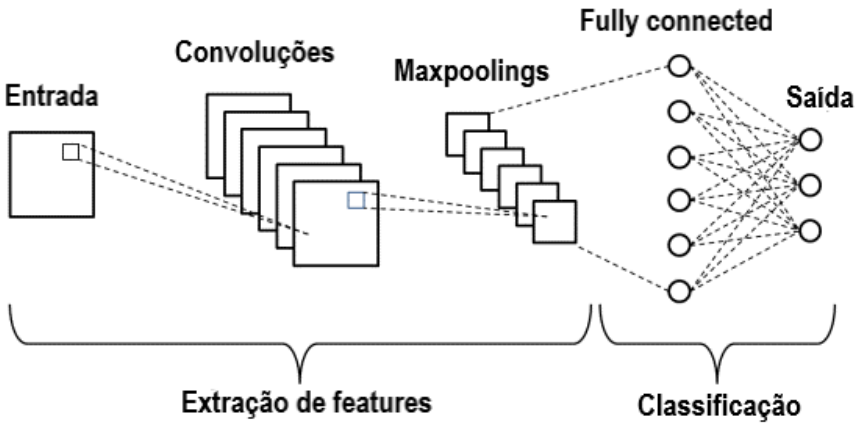


Figura 3.13: Exemplo de uma CNN

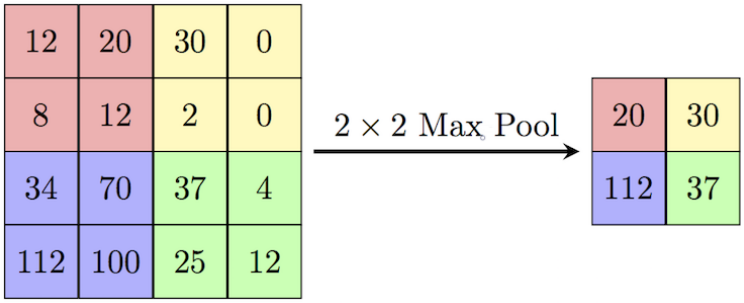


Figura 3.14: Aplicação de *max pooling* 2x2

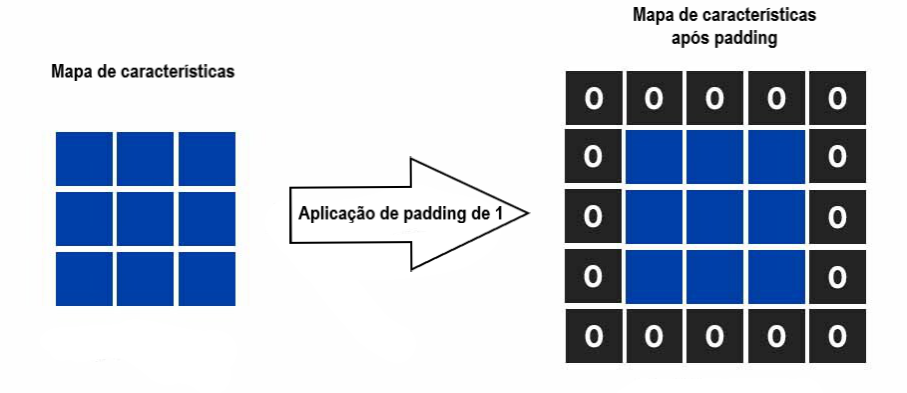


Figura 3.15: Aplicação de *padding* de 1

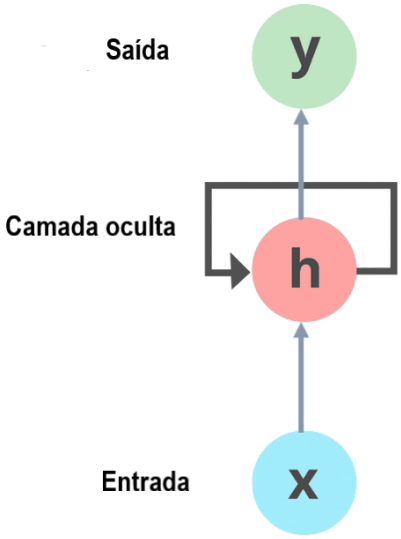


Figura 3.16: RNN simples

3.2 Tipos de ciberataque

Os ciberataques são ações maliciosas realizadas por indivíduos ou grupos com o objetivo de comprometer sistemas de computadores, redes ou dispositivos eletrónicos. Com o aumento de atividades *online*, o número e variedade de ciberataques também aumentou. Esses ataques podem ser direcionados a governos, empresas, organizações ou até mesmo a utilizadores individuais. Os ciberataques podem ter várias formas e níveis de sofisticação, e os atacantes geralmente têm motivações diferentes, que vão desde ganhos financeiros até sabotagem, espionagem ou simplesmente o prazer de causar danos. Existem diversos tipos de ciberataques, cada um com suas características e métodos específicos. Nas subsecções a seguir apresenta-se alguns tipos de ataque e os seus detalhes.

3.2.1 *Malware*

Malware (*Malicious Software*) refere-se a qualquer *software* ou programa especificamente concebido para danificar, explorar ou perturbar sistemas informáticos, redes ou dispositivos sem o conhecimento ou consentimento do utilizador [23]. Dependendo da intenção do atacante, o *malware* pode funcionar de forma diferente, executando várias funções como roubar informações sensíveis, obter acesso não autorizado a sistemas, causar danos aos dispositivos alvo, sequestrar, encriptar ou mesmo apagar dados. Existem vários tipos de *malware*, incluindo:

- **Vírus:** São programas que se podem replicar e infetar outros ficheiros ou sistemas, ligando-se a ficheiros hospedeiros. Os vírus podem causar vários tipos de danos, tais como corromper ou apagar ficheiros e perturbar a funcionalidade do sistema [39].

- ***Worms*:** Um *worm* ou rede pode consumir recursos de computador: memória, largura de banda, tempo de processo, etc., levando ao abrandamento do processamento do computador e mesmo ao bloqueio deste. São semelhantes aos vírus, mas podem propagar-se de forma independente, sem a necessidade de um ficheiro hospedeiro. Normalmente, exploram vulnerabilidades de segurança em redes informáticas para se propagarem e infetarem outros sistemas [36].

- ***Ransomware*:** O *ransomware* normalmente funciona encriptando os ficheiros no computador, ou mesmo o computador em si, com uma chave desconhecida do utilizador negando-lhes o acesso e depois exige um resgate em troca da chave de desencripta-

ção. Pode propagar-se através de vários meios, como anexos de correio eletrónico, descarregamentos maliciosos ou exploração de vulnerabilidades [26].

- ***Adware*:** O *adware* apresenta anúncios indesejados ou redireciona os utilizadores para sites de publicidade. Embora não seja tão prejudicial como outros tipos de *malware*, pode ser incómodo e intrusivo e afetar negativamente a experiência do utilizador [10].

- **Cavalos de Tróia:** Os cavalos de tróia são fragmentos de código malicioso dentro do software que parece ser fiável, isto é, disfarçam-se de software ou ficheiros legítimos para enganar os utilizadores e levá-los a executá-los ou transferi-los, abrindo o computador para acesso remoto. Uma vez ativados, os cavalos de tróia podem executar ações maliciosas, como roubar informações pessoais, fornecer acesso não autorizado a sistemas ou criar *backdoors* ("abrir portas") para outros *malwares* [46].

- ***Spyware*:** O *spyware* permite a recolha de informações sobre as atividades dos utilizadores sem o seu conhecimento. Pode registar as teclas premidas, capturar imagens de ecrã, seguir os hábitos de navegação na Web e transmitir os dados recolhidos para um servidor remoto [38].

3.2.2 *Phishing*

O *phishing* é um tipo de ataque cibernético em que um atacante se faz passar por uma entidade ou organização de confiança para enganar os indivíduos e levá-los a revelar informações sensíveis, como nomes de utilizador, palavras-passe, detalhes de cartões de crédito ou outras informações pessoais. Estes tipos de ataque ocorrem normalmente através de emails fraudulentos, mensagens instantâneas ou *links* para *websites* enganadores que imitam fontes legítimas [3].

3.2.3 DoS e DDoS

DoS (*Denial of Service*), em português, negação de serviço, é um tipo de ataque cibernético em que o atacante tenta perturbar ou desativar o funcionamento normal de um sistema informático, rede ou *website*, sobrecarregando-o com quantias de pedidos ilegítimos ou tráfego excessivo. As vítimas de ataques DoS são frequentemente servidores *Web* de organizações de alto nível, como bancos, empresas comerciais e de média ou organizações governamentais e comerciais. Embora estes ataques não resultem num roubo ou perda de informações importantes ou outros bens, podem custar à vítima muito tempo e dinheiro.

DDoS (*Distributed Denial of Service*) ocorrem quando vários sistemas de ataque organizam um ataque DoS sincronizado a um único alvo. A principal diferença é que, em vez de ser atacado a partir de um local, o alvo pode ser atacado a partir de vários locais ao mesmo tempo [11].

3.2.4 U2R

Um ataque U2R (*User to Root*) é um tipo de ataque de cibersegurança que envolve um utilizador não autorizado que obtém privilégios acrescidos ou acesso *root* num sistema informático ou numa rede. No contexto da segurança informática, *root* refere-se normalmente ao nível mais elevado de acesso ou privilégios administrativos em sistemas operativos. A obtenção de acesso *root* permite que um atacante tenha controlo total sobre o sistema, o que pode ser altamente perigoso e prejudicial [37].

3.2.5 R2L

Um ataque R2L (*Remote to Local*) é um tipo de ataque de cibersegurança em que um atacante tenta obter acesso não autorizado a um computador ou rede a partir de uma localização remota e, em seguida, aumentar os seus privilégios para obter acesso local ou controlo sobre o sistema alvo. Em contraste com os ataques U2R, que se concentram na elevação de privilégios quando um atacante tem acesso local a um sistema, os ataques R2L visam violar um sistema remotamente e, subsequentemente, obter controlo ou acesso como se fossem um utilizador local [37].

3.2.6 *Probe*

Probing attack, mais conhecido como *Probe*, é uma tentativa de recolher informações sobre uma rede de computadores com o objetivo aparente de contornar os seus controlos de segurança [40].

3.2.7 Força bruta

Força bruta, é um método de tentativa e erro utilizado na cibersegurança para obter acesso não autorizado a um sistema, conta ou dados encriptados. Envolve a tentativa sistemática de todas as combinações possíveis de palavras-passe ou chaves de encriptação até ser encontrada a correta. A maioria dos ataques de força bruta são automáticos, pelo que

a variedade de alvos, ou seja, de tipos de vítimas, é bastante elevada, incluindo contas de utilizador, serviços de rede, algoritmos de encriptação ou chaves criptográficas. A eficácia de um ataque de força bruta depende de fatores como a complexidade da palavra-passe ou da chave, os recursos disponíveis para o atacante e quaisquer contra-medidas em vigor para detetar ou atenuar tais ataques [7].

3.2.8 Ataque de injeção

Um ataque de injeção é um tipo de ataque de segurança em que código ou comandos maliciosos são inseridos numa aplicação ou sistema com a intenção de manipular o seu comportamento ou obter acesso não autorizado. Pode expor ou danificar dados e conduzir a DoS ou a um comprometimento completo de um servidor *Web*. Os ataques de injeção mais comuns são o *Cross-site scripting* e a injeção de SQL [43].

3.3 *Datasets* de acesso livre

Nesta secção encontra-se alguns tipos de *datasets* de acesso livre baseados em tráfegos de rede, tráfegos de *Internet*, tráfego de dispositivos, entre outros. Todos estes têm um objetivo comum, serem usados no estudo de deteção de intrusões de redes. Alguns desses *datasets* como DARPA98-99 e KDD99, estão perdendo a sua validade na literatura dia a dia. A seguir apresenta-se alguns dos *datasets* mais usados para deteção de intrusões.

3.3.1 DARPA98-99

DARPA (*Defense Advanced Research Projects Agency*) baseia-se no tráfego e análise de rede em que os dados de treino contêm sete semanas de ataques e os dados de teste contêm duas semanas de ataques. O *dataset* foi construído no laboratório *Lincoln* do Instituto Tecnológico de Massachusetts (MIT) para análise da segurança da rede e expôs os problemas associados à injeção artificial de ataques e tráfego benigno. Inclui atividades de *email*, *browsing*, FTP, Telnet, IRC e SNMP. Contém ataques como DoS, *Guess password*, *Buffer overflow*, FTP remoto, Syn flood, Nmap e Rootkit. Este *dataset* não representa o tráfego de rede do mundo real e contém irregularidades como a ausência de falsos positivos. Em outras palavras, o *dataset* pode não ser totalmente realista em termos de como os sistemas de segurança de rede lidam com ataques e tráfego benigno na prática, devido à falta de falsos positivos, que são uma parte importante da avaliação de sistemas de deteção de intrusões e

segurança cibernética em cenários do mundo real [14] [35].

3.3.2 KDD Cup99

Este *dataset* foi criado para a terceira competição *Knowledge Discovery and Data Mining* (KDD) *Cup* em 1999 que se concentrou na tarefa de deteção de intrusão em redes, especificamente a deteção de acessos não autorizados a redes de computadores. Foi criado por uma equipa de investigadores e peritos no domínio da segurança de redes. Contém uma grande coleção de dados de tráfego de rede (cerca de cinco milhões), simulando um ambiente de rede informática com vários tipos de ataques à rede, incluindo tráfego normal e de ataque, o que o torna adequado para o desenvolvimento e avaliação de algoritmos e sistemas de deteção de intrusões. Os ataques simulados podem ser agrupados em quatro grupos: U2R, R2L, DoS e Probe. Existem 41 *features* no *dataset*. Estas *features* incluem tráfego, conteúdo e *features* gerais [27].

3.3.3 NSL-KDD

NSL-KDD (*Network Socket Layer-Knowledge Discovery in Datasets*) é uma versão aprimorada e atualizada do conjunto de dados KDD Cup99 para deteção de intrusões em redes. Foi criado por investigadores da Universidade do Novo México e da Universidade do Texas, e é composto por registos de conexões de rede, assim como o KDD Cup99, mas passou por um processo de pré-processamento adicional para remover redundâncias e inconsistências presentes no conjunto de dados original. Uma das principais melhorias do NSL-KDD é a divisão mais equilibrada entre tráfego normal e tráfego de ataque, o que proporciona uma representação mais realista dos cenários de rede [33].

3.3.4 UNSW-NB15

Este *dataset* foi criado por investigadores da UNSW (*University of New South Wales*) Canberra (Austrália), em 2015, com a ferramenta *IXIA PerfectStorm* no *Cyber Range Lab* com um objetivo de simular o tráfego de rede do mundo real. Tem uma representação mais recente e realista e mais diversidade de tipos de ataques do que os *datasets* referidos anteriormente. A ferramenta tcpdump foi utilizada para capturar 100 GB de tráfego (ficheiros Pcap). O *dataset* UNSW-NB15 contém 49 *features* e 9 tipos de ataque que são denominados de, Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms [22].

3.3.5 CICIDS 2017

CICIDS 2017 (*Canadian Institute for Cybersecurity Intrusion Detection System* 2017) é um conjunto de dados amplamente utilizado para pesquisa em deteção de intrusões em redes. Foi desenvolvido pelo Instituto Canadiano para Cibersegurança, localizado na *Universidade de New Brunswick*, no Canadá. O *dataset* é composto por dados de tráfego de rede que simulam uma ampla variedade de cenários de ataques e tráfego normal. Representa eventos de rede gerados por um total de 25 utilizadores. Os perfis dos utilizadores são definidos para incluir protocolos específicos, como os protocolos HTTP, HTTPS, FTP, SSH e E-Mail. Contém cerca de 3 milhões de dados, 83 *features* e 14 tipos de ataque [28].

3.4 Métricas de avaliação

Na área de ML, as métricas de avaliação servem para averiguar se um modelo tem um bom desempenho. A escolha da métrica adequada depende do problema específico e dos requisitos de avaliação do modelo. É importante considerar as características do *dataset* e o contexto do problema ao selecionar a métrica de avaliação mais apropriada. Antes de referir às métricas mais comummente utilizadas, apresenta-se a seguir as quatro definições que são utilizadas para calcular as métricas:

- ***True Positive* (TP):** Um verdadeiro positivo ocorre quando o modelo identifica um evento como malicioso, e ele é de facto malicioso. Por outras palavras, o modelo deteta corretamente uma intrusão ou ataque real.
- ***True Negative* (TN):** Um verdadeiro negativo acontece quando o modelo identifica corretamente um evento não malicioso como benigno ou normal. Neste caso, o sistema determina corretamente que o evento não é uma intrusão ou um ataque.
- ***Falso Positive* (FP):** Um falso positivo ocorre quando o modelo assinala incorretamente um evento benigno como malicioso. Significa que o sistema dispara um alarme ou comunica um evento como uma intrusão, mas na realidade trata-se de um falso alarme.
- ***False Negative* (FN):** Um falso negativo ocorre quando o modelo não consegue detetar uma intrusão ou ataque real. O sistema não deteta ou ignora o evento malicioso, classificando-o incorretamente como benigno ou normal. Isto quer dizer que o modelo falhou e houve uma intrusão ou ataque.

Tendo estas definições, a seguir mostra-se as métricas mais comummente utilizadas:

- ***Accuracy* (Acc):** É a métrica mais básica e representa a proporção de predições corretas em relação ao total de predições. É útil quando as classes são equilibradas, ou seja, têm aproximadamente a mesma quantidade de amostras. No entanto, a exatidão pode ser enganosa quando as classes são desequilibradas. Calcula-se a partir da fórmula 3.1.

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

- ***Precision*:** A precisão mede a capacidade do modelo em realizar corretamente previsões positivas. É a proporção de eventos que são previstos pelo algoritmo como intrusões e que são efetivamente intrusões. Calcula-se a partir da fórmula 3.2.

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

- ***Recall*:** O *recall* ou sensibilidade mede a capacidade do modelo em encontrar corretamente as instâncias positivas. É a proporção de intrusões efetivas que foram previstas como intrusões pelo algoritmo. Calcula-se a partir da fórmula 3.3.

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

- ***F1-Score*:** É a média harmónica entre precisão e *recall*. É uma métrica útil quando se deseja levar em consideração tanto os falsos positivos quanto os falsos negativos. Calcula-se a partir da fórmula 3.4.

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.4}$$

- ***Area Under the ROC Curve* (AUC-ROC):** Área sob a curva ROC é uma métrica usada em problemas de classificação binária para avaliar o desempenho geral do modelo. A curva ROC (*Receiver Operating Characteristic*) é um gráfico que mostra a taxa de verdadeiros positivos (TPR) em função da taxa de falsos positivos (FPR) em diferentes pontos de corte. Quanto maior a área sob a curva (AUC), melhor o desempenho do modelo. Podemos ver um exemplo de um gráfico AUC na figura 3.17.

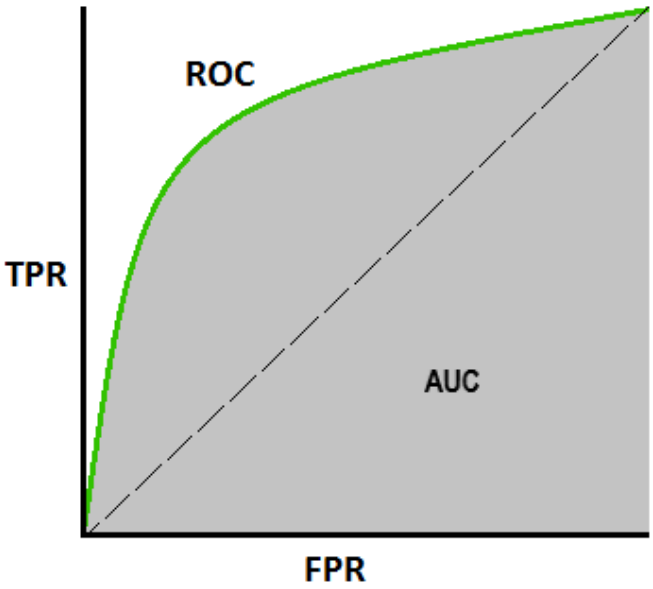


Figura 3.17: Exemplo de um gráfico AUC

- **Matriz de confusão:** A matriz de confusão é uma tabela que resume o desempenho de um modelo de classificação, mostrando a contagem de previsões corretas e incorretas para cada classe do problema. É uma ferramenta útil para avaliar a qualidade das previsões feitas pelo modelo. Na figura abaixo, figura 3.18, encontra-se um exemplo de uma matriz de confusão com 5 classes, na qual foram previstas corretamente 11000 saídas como classe 0, 7400 como classe 1, 5200 como classe 2, 17 como classe 3 e 2000 como classe 4. Os outros valores foram saídas previstas de forma incorreta.

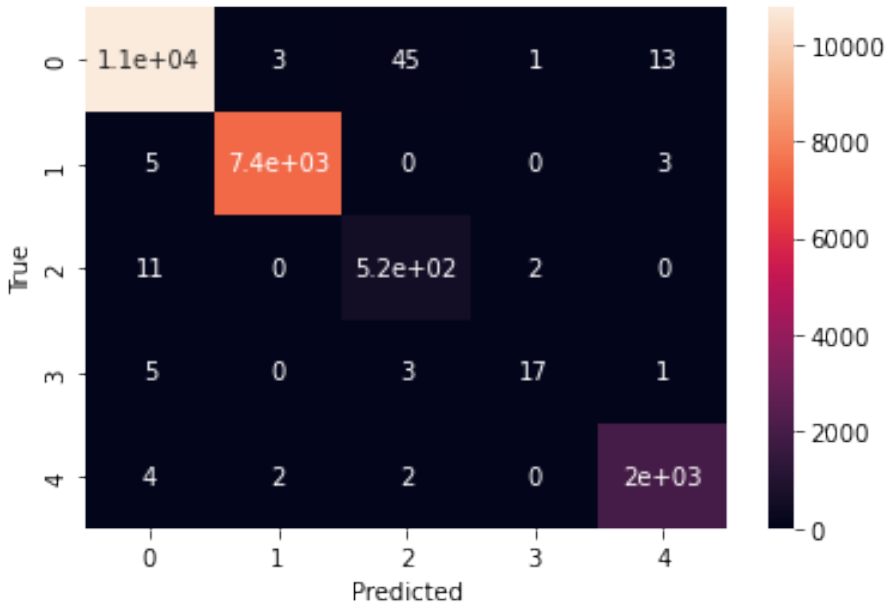


Figura 3.18: Exemplo de uma matriz de confusão

4 Implementação e resultados obtidos

4.1 Descrição do *setup* utilizado

Para a implementação deste trabalho foi escolhido a plataforma *Jupyter Lab*, versão 3.2.1 da *Anaconda Navigator* com a linguagem *Python*. A linguagem *python* contém muitas bibliotecas, como o *sklearn* e *tensorflow* que ajudam muito nestes tipos de trabalho. O computador usado foi um *HP Pavilion* com o processador *Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz*. Nas secções a seguir encontra-se o *dataset* escolhido com detalhe e o *pipeline*.

4.2 *Dataset*

O *dataset* escolhido para o trabalho foi o UNSW-NB15, que está dividido em dois: *dataset* de treino e *dataset* de teste. Tal como referido na subsecção 3.3.4, este *dataset* é um dos mais utilizados para treinar um modelo de ML de modo a detetar intrusões nas redes. Foi criado através da utilização de uma ferramenta *IXIA PerfectStorm* para extrair atividades híbridas de ataques do tráfego de rede. Foi utilizada uma ferramenta denominada de *tcpdump* para capturar 100 GB de tráfego de rede (ficheiros pcap). Cada ficheiro pcap contém 1000 MB para facilitar a análise dos pacotes. Utilizaram várias técnicas para gerar 49 *features* com as suas respetivas classes *label*. Nas tabelas abaixo, encontram-se 6 grupos de *features*: *flow features* (tabela 4.1), *basic features* (tabela 4.2), *content features* (tabela 4.3), *time features* (tabela 4.4), *additional generated features* (tabelas 4.5 e 4.6) e *labeled features* (tabela 4.7).

Após a obtenção de todos os dados, estes foram divididos em dois *datasets*: um *dataset* de treino, que possui 82.332 dados, e um de teste, que possui 175.341 dados. Cada um deles contém as 49 *features*, tendo 10 classes na *feature* "*attack_cat*", uma de tipo normal e 9 tipos de ataque. Essas 10 classes são descritas nas tabelas 4.8 e 4.9.

Tabela 4.1: *Flow features*

Número	**Nome**	**Tipo**	**Descrição**
1	srcip	nominal	*Source IP address*
2	sport	inteiro	*Source port number*
3	dstip	nominal	*Destination IP address*
4	dsport	inteiro	*Destination port number*
5	proto	nominal	*Transaction protocol*

Tabela 4.2: *Basic features*

Número	**Nome**	**Tipo**	**Descrição**
6	state	nominal	*The state and its dependent protocol, e.g ACC,CLO, CON,ECO,ECR,FIN,INT,MAS,PAR,REQ, RST,TST,TXD,URH,URN, and (-)*
7	dur	float	*Record total duration*
8	sbytes	inteiro	*Source to destination bytes*
9	dbytes	inteiro	*Destination to source bytes*
10	sttl	inteiro	*Source to destination time to live*
11	dttl	inteiro	*Destination to source time to live*
12	sloss	inteiro	*Source packets retransmitted or dropped*
13	dloss	inteiro	*Destination packets retransmitted or dropped*
14	service	nominal	*http,ftp,smtp,ssh, dns,ftp-data,irc and (-)*
15	sload	float	*Source bits per second*
16	dload	float	*Destination bits per second*
17	spkts	inteiro	*Source to destination packet count*
18	dpkts	inteiro	*Destination to source packet count*

Tabela 4.3: *Content features*

Número	**Nome**	**Tipo**	**Descrição**
19	swin	inteiro	*Source TCP window advertisement value*
20	dwin	inteiro	*Destination TCP window advertisement value*
21	stcpb	inteiro	*Source TCP sequence number*
22	dtcpb	inteiro	*Destination TCP sequence number*
23	smeansz	inteiro	*Mean of the flow packet size transmitted by the src*
24	dmeansz	inteiro	*Mean of the flow packet size transmitted by the dst*
25	trans_depth	inteiro	*Represents the pipelined depth into the connection of http requestresponse transaction*
26	res_bdy_len	inteiro	*Actual uncompressed content size of the data transferred from the serverś http service*

Tabela 4.4: *Times features*

Número	**Nome**	**Tipo**	**Descrição**
27	sjit	float	*Source jitter (mSec)*
28	djit	float	*Destination jitter (mSec)*
29	stime	timestamp	*Record start time*
30	ltime	timestamp	*Record last time*
31	sintpkt	float	*Source inter-packet arrival time (mSec)*
32	dintpkt	float	*Destination inter-packet arrival time (mSec)*
33	tcprtt	float	*The sum of 'synack' and 'ackdat' of the TCP*
34	synack	float	*The time between the SYN and the SYN_ ACK packets of the TCP*
35	ackdat	float	*The time between the SYN_ ACK and the ACK packets of the TCP*

Tabela 4.5: *Additional generated features*

Número	**Nome**	**Tipo**	**Descrição**
36	is_sm_ips_ports	binário	*If source (1) equals to destination (3)IP addresses and port numbers (2)(4) are equal, this variable takes value 1 else 0*
37	ct_state_ttl	inteiro	*Number for each state (6) according to specific range of values for source/destination time to live (10) (11)*
38	ct_flw_http_mthd	inteiro	*Number of flows that has methods such as Get and Post in http service*
39	is_ftp_login	binário	*If the ftp session is accessed by user and password then 1 else 0*
40	ct_ftp_cmd	inteiro	*Number of flows that has a command in ftp session*
41	ct_srv_src	inteiro	*Number of connections that contain the same service (14) and source address (1) in 100 connections according to the last time (26)*
42	ct_srv_dst	inteiro	*Number of connections that contain the same service (14) and destination address (3) in 100 connections according to the last time (26)*
43	ct_dst_ltm	inteiro	*Number of connections of the same destination address (3) in 100 connections according to the last time (26)*
44	ct_src_ltm	inteiro	*Number of connections of the same source address (1) in 100 connections according to the last time (26)*
45	ct_src_dport_ltm	inteiro	*Number of connections of the same source address (1) and the destination port (4) in 100 connections according to the last time (26)*

Tabela 4.6: *Additional generated features* (continuação da tabela 4.5)

Número	**Nome**	**Tipo**	**Descrição**
46	ct_dst_sport_ltm	inteiro	*Number of connections of the same destination address (3) and the destination port (2) in 100 connections according to the last time (26)*
47	ct_dst_src_ltm	inteiro	*Number of connections of the same source (1) and the destination (3) address in 100 connections according to the last time (26)*

Tabela 4.7: *Labeled features*

Número	**Nome**	**Tipo**	**Descrição**
48	attack_cat	nominal	*The name of each attack category. In this dataset, nine categories (e.g.,Fuzzers,Analysis,Backdoors,DoS, Exploits,Generic,Reconnaissance,Shellcode and Worms)*
49	label	binário	*0 for normal and 1 for attack records*

Tabela 4.8: Classes da *feature "attack_ cat"*

Tipo	**Descrição**	**Nº de dados**		**Total**
		Treino	***Teste***	
Normal	Dados de tráfego normais.	37,000	56,000	93,000
Fuzzer	É um ataque com tentativa de fazer com que um programa ou rede seja suspenso, alimentando-o com dados gerados aleatoriamente.	6,062	18,184	24,246
Reconnaissance	É um ataque que recolhe informações sobre uma rede de computadores para escapar aos controlos de segurança.	3,496	10,491	13,987
Shellcode	É um malware em que o atacante penetra num pequeno pedaço de código a partir de uma *shell* para controlar a máquina comprometida.	378	1,133	1,511

Tabela 4.9: Classes da *feature "attack_ cat"* (continuação da tabela 4.8)

Tipo	**Descrição**	**Nº de dados**		**Total**
		Treino	***Teste***	
Analysis	É um tipo de variedade de intrusões que penetram nas aplicações Web através de portas (ex: *port scans*), emails (ex: spam) e *Web scripts* (ex: ficheiros HTML).	677	2,000	2,677
Backdoor	É uma técnica de contornar uma autenticação normal, assegurando o acesso remoto não autorizado a um dispositivo.	583	1,746	2,329
DoS	Um ataque malicioso que tenta desativar o funcionamento normal de um serviço ou rede a utilizadores, sobrecarregando-o com quantias de pedidos ilegítimos ou tráfegos excessivos.	4,089	12,264	16,353
Exploit	É uma sequência de instruções que tira proveito o conhecimento de uma falha ou erro de segurança num sistema ou rede e assim, explorando a sua vulnerabilidade.	11,132	33,393	44,525
Generic	É uma técnica funciona contra todos os cifradores de bloco (com um dado tamanho de bloco e de chave), sem ter em conta a configuração do cifrador de bloco.	18,871	40,000	58,871
Worm	É um tipo de malware que pode propagar-se de forma independente, sem a necessidade de um ficheiro hospedeiro. Normalmente, exploram vulnerabilidades de segurança em redes informáticas para se propagarem e infetarem outros sistemas.	44	130	174

4.3 *Pipeline*

Após a escolha da plataforma, linguagem de programação e *dataset*, definiu-se um *pipeline* para treinar os modelos com classificação binária e classificação multi-classe. Na figura 4.1, é um esquema de como foi conduzido o trabalho.

1. Carregamento dos *datasets* para um *Panda Dataframe*
2. Pré-processamento de dados
3. Manipulação de dados
4. Treinos de modelos de *machine learning*
    - Treino com *Support Vector Machine*
    - Treino com *Multi-Layer Perceptron*
    - Treino com *Convolutional Neural Network*
    - Treino com *Random Forest+K-means*

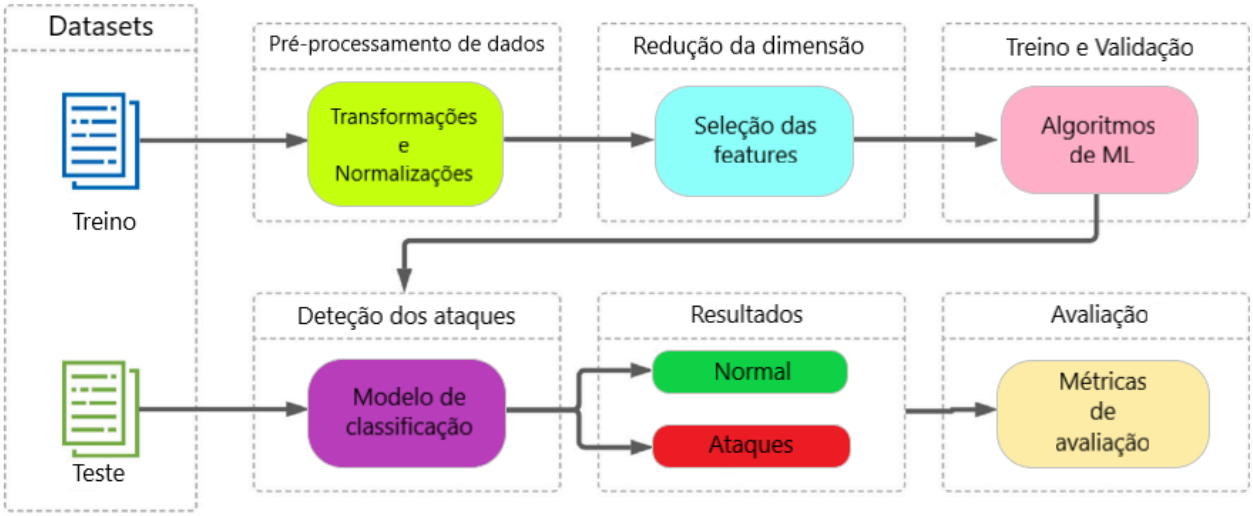


Figura 4.1: Esquema do trabalho

4.3.1 Carregamento dos *datasets* para um *Panda Dataframe*

Em ambas classificações, pegou-se nos *datasets* obtidos no formato CSV e transformou-se para um *Pandas Dataframe* facilitando a manipulação e análise dos dados. *Pandas* é uma biblioteca do Python para manipulação e análise de dados. Fornece uma poderosa estrutura de dados chamada *DataFrame*, que é semelhante a uma tabela ou a uma folha de cálculo.

4.3.2 Pré-processamento de dados

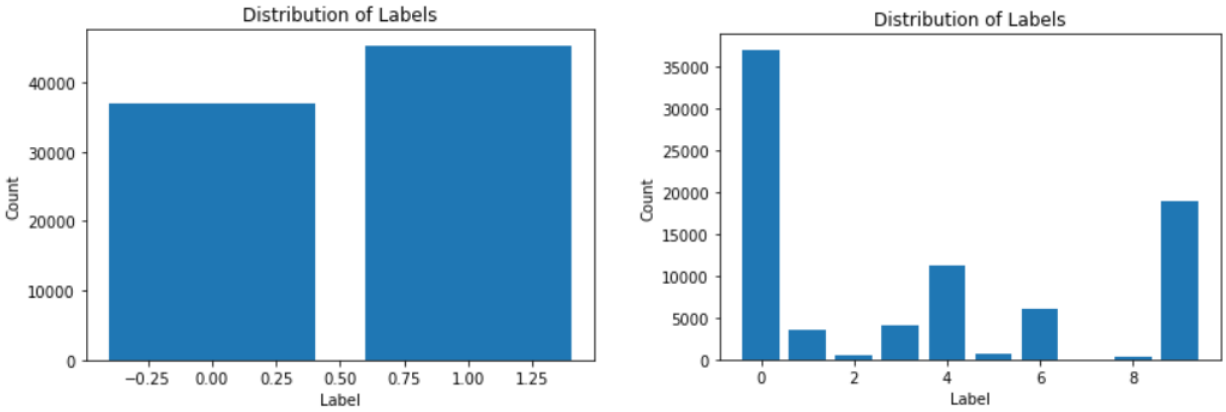
Durante o processamento dos dados, em primeiro lugar, procurou-se pelos valores em falta nos *datasets*, em segundo lugar, eliminar os dados duplicados e redundantes e os valores únicos e em terceiro lugar, eliminar as *features* desnecessárias. No caso da classificação binária eliminou-se as *features id*, *attack_ cat* e *rate* e na classificação multi-classe eliminou-se as *features id*, *label* e *rate*, pois são *features* desnecessárias para o treino. Se não resolvermos esses problemas quando estamos a lidar com um *dataset*, pode levar a uma má precisão ou exatidão no resultado final, e no pior dos casos, leva a um *overfitting* que é um problema muito comum em todos os treinos de ML e que ninguém o deseja.

Após esses passos, converteram-se as *features* do tipo categórico (referidas nas tabelas 4.1-4.7) em *features* numéricas, pois os algoritmos de ML só requerem entradas numéricas. Na classificação multi-classe, foi feita a conversão da *feature* "*attack_ cat*" da seguinte forma:

- Normal -> 0
- *Reconnaissance* -> 1
- *Backdoor* -> 2
- *DoS* -> 3
- *Exploits* -> 4
- *Analysis* -> 5
- *Fuzzers* -> 6
- *Worms* -> 7
- *Shellcode* -> 8
- *Generic* -> 9

4.3.3 Manipulação de dados

Nesta etapa, começou-se por guardar em duas variáveis (X e y) os dados de treino desejados. A variável X apresenta-se na forma (82332, 41) na qual 82332 são os dados e 41 são as *features*. A variável y apresenta-se na forma (82332, 2) para classificação binária e (82332, 10) para classificação multi-classe sendo 82332 os dados de treino. Os valores 2 e 10 dizem respeito ao número de classes que queremos classificar. Nas figuras 4.2 e 4.3 podemos ver como os dados estão distribuídos.



(a) Classificação binária

(b) Classificação multi-classe

Figura 4.2: Distribuição dos dados em gráfico

```
counter = Counter(y)
print(counter)
```

```
Counter({1: 45332, 0: 37000})
```

(a) Classificação binária

```
counter = Counter(y)
print(counter)
```

```
Counter({0: 37000, 9: 18871, 4: 11132, 6: 6062, 3: 4089, 1: 3496, 5: 677, 2: 583, 8: 378, 7: 44})
```

(b) Classificação multi-classe

Figura 4.3: Distribuição dos dados em quantidade

A seguir, eliminaram-se as *features* cujo dados têm poucas variâncias ficando com 82332 dados e 35 *features*. De modo a equilibrar o *dataset*, isto é, incrementar ou decrementar as classes minoritárias ou classes maioritárias, foi feito um *oversampling* e um *undersampling* utilizando as técnicas SMOTE (*Synthethic Minority Oversampling TEchnique*) e RandomUnderSampler respetivamente. A técnica SMOTE aumenta de forma sintética o número de dados das classes minoritárias (classes que têm menos dados como o 7 e o 8) e a técnica *RandomUnderSampler*, contrário do SMOTE, decrementa a classe majoritária (classe 0). Após de aplicar essas técnicas, a quantidade de dados aumenta para 370.000, ficando cada classe com 37.000. Essas técnicas foram apenas usadas durante a classificação multi-classe. No gráfico 4.4 podemos ver a distribuição desses dados.

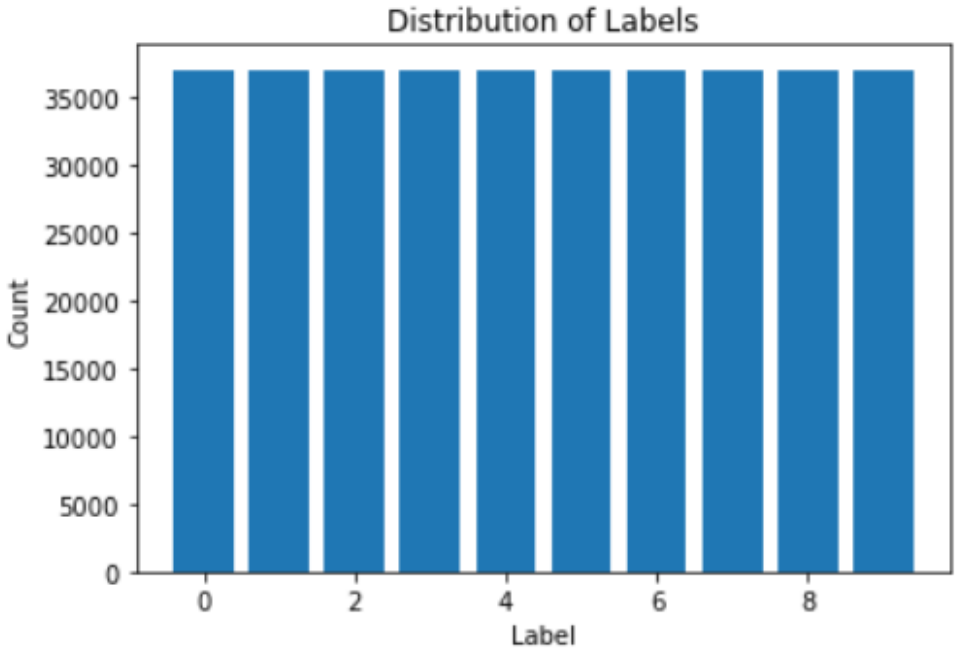


Figura 4.4: Nova distribuição dos dados em gráfico da classificação multi-classe

A técnica *SelectKBest* foi utilizada para escolher as melhores *features* das 35 existentes. Foi aplicado com o k=30 e o hiperparâmetro "mutual_info_classif" que calcula a informação mútua entre cada *feature* e a variável alvo (classes a classificar) para determinar a sua importância. A seguir foi feita uma normalização dos dados utilizando a técnica *StandardScaler* que transforma os dados para ter média zero e desvio padrão igual a 1. Desta forma, torna todos os dados comparáveis e facilita a análise e ajuste do modelo. Finalmente, antes de aplicar algoritmos de ML, foi feita uma divisão dos dados de treino em 20/80, isto é, 20% para teste e 80% para treino. Podemos ver essa divisão na figura 4.5.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2

print(X_train.shape)
print(X_test.shape)
```

```
(65865, 30)
(16467, 30)
```

(a) Classificação binária

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2

print(X_train.shape)
print(X_test.shape)
```

```
(296000, 30)
(74000, 30)
```

(b) Classificação multi-classe

Figura 4.5: Divisão dos dados para treino e teste

4.3.4 Treino com SVM

Após terem sido definidos os *datasets* de treino e de teste, utilizou-se o classificador SVM da biblioteca '*sklearn*' para classificar as duas e as 10 classes desejadas. Os hiperparâmetros escolhidos foram: kernel='rbf', C=3 e gamma=10. Com estes hiperparâmetros obteve-se os resultados descritos na figura 4.6. As figuras 4.7 e 4.8 são as matrizes de confusão obtidas utilizando o algoritmo SVM.

```
Accuracy:  0.9305277221108884
Precision:  0.9645081678223058
Recall:  0.9069510443142889
AUC:  0.9331196310813828
F1:  0.9348445153206515
```

(a) Classificação binária

```
Accuracy:  0.7905372432456412
Precision:  0.8113980571202323
Recall:  0.7905372432456412
F1:  0.7947135286739074
```

(b) Classificação multi-classe

Figura 4.6: Resultados de SVM

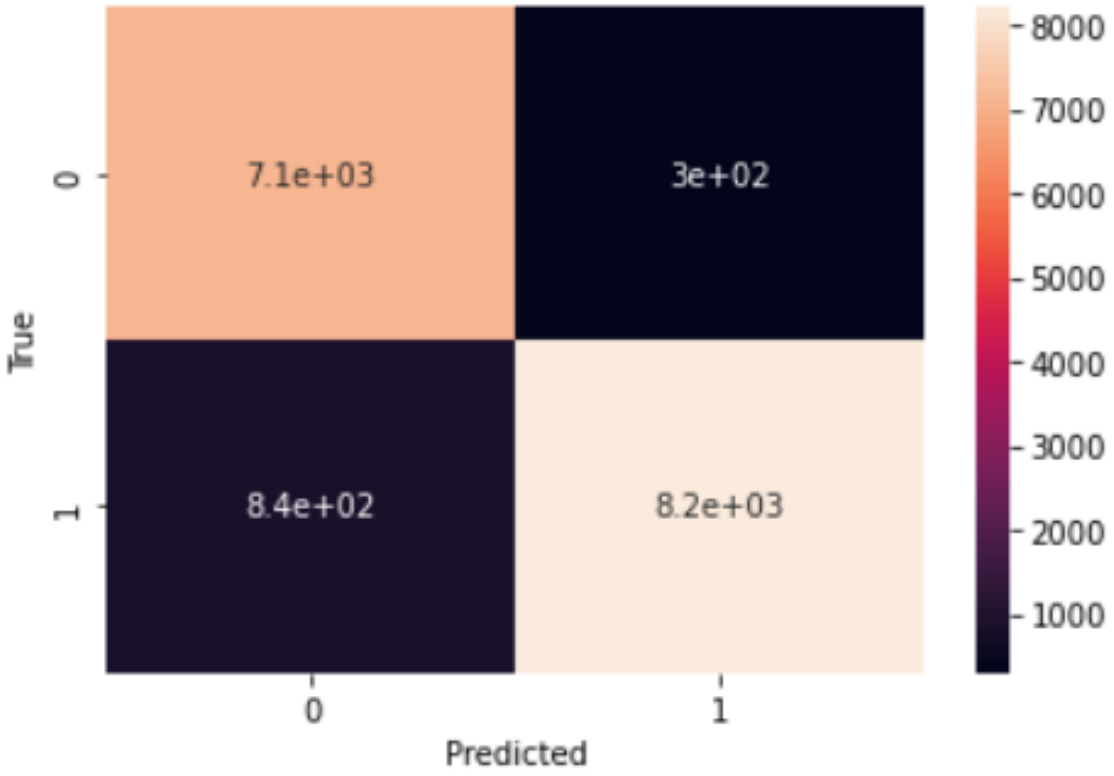


Figura 4.7: Matriz de confusão com SVM em classificação binária

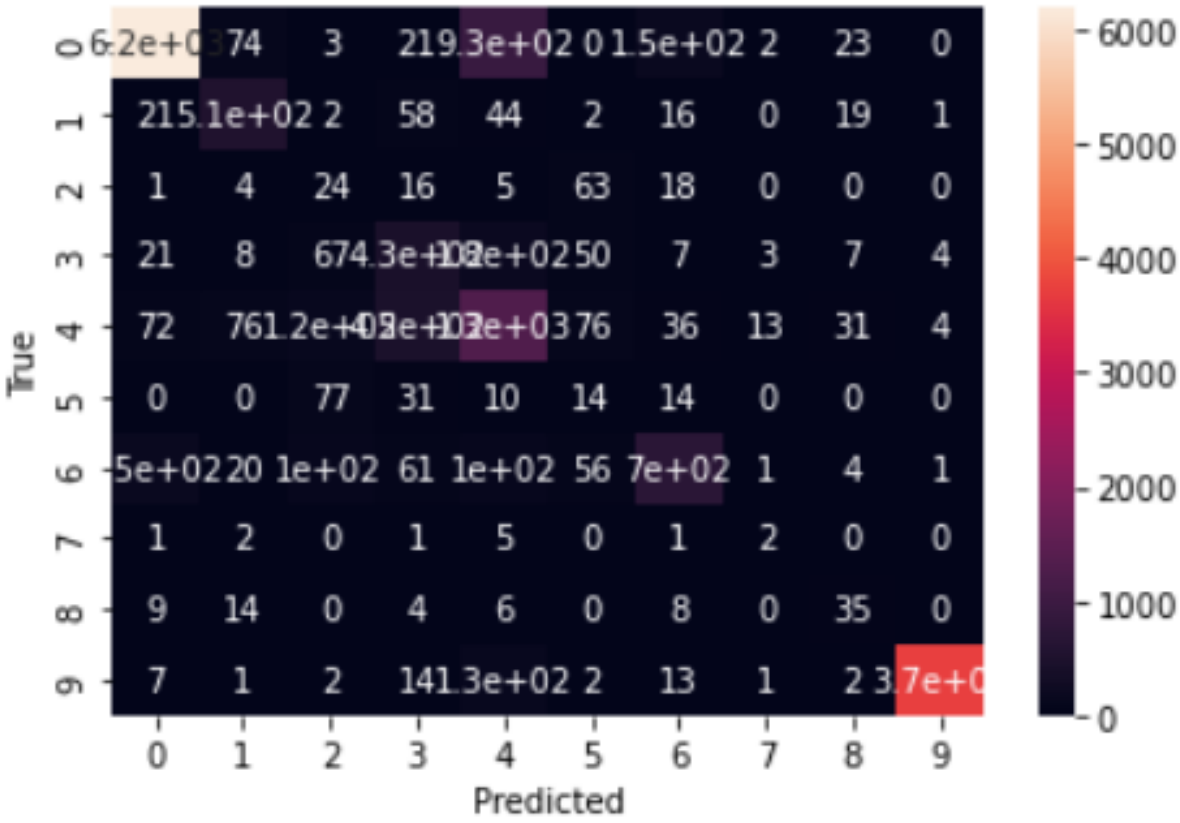


Figura 4.8: Matriz de confusão com SVM em classificação multi-classe

4.3.5 Treino com MLP

Para poder treinar com MLP (*Multi-Layer Perceptron*), foi definida uma rede tendo cerca de 2.9 milhões de parâmetros para treinar em ambas classificações. A figura 4.9 e 4.10 são a arquitetura e a tabela sumariada do MLP.

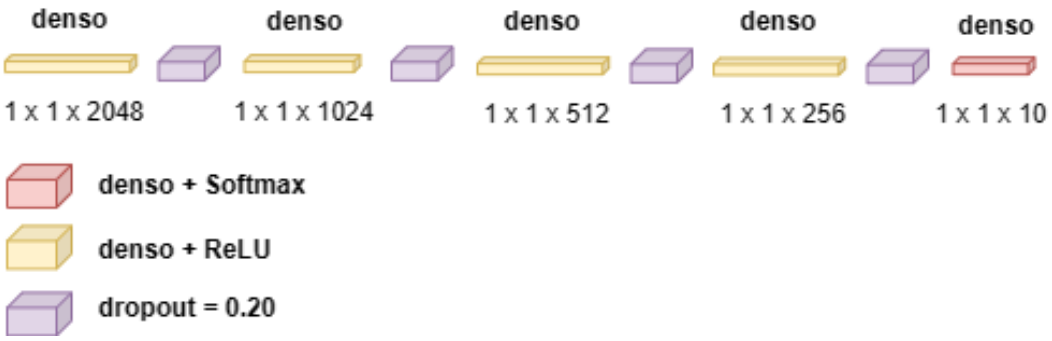


Figura 4.9: Arquitetura do MLP

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 2048)	63488
dropout (Dropout)	(None, 2048)	0
dense_1 (Dense)	(None, 1024)	2098176
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 512)	524800
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 256)	131328
dropout_3 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 10)	2570

Total params: 2,820,362
Trainable params: 2,820,362
Non-trainable params: 0

Figura 4.10: Tabela sumariada da arquitetura

Após definir a rede, escolheu-se o otimizador Adam com os hiperparâmetros learning_rate=0.0001, beta_1=0.9 e beta_2=0.999. Ambos os treinos foram feitos com 30 épocas e um *batch_size*=32. Os resultados obtidos de *accuracy* em binário e multi-classe foram de 96.1% e 81.8%, respetivamente. Os gráficos abaixo, gráficos 4.11 e 4.12, mostram o treino e validação da *accuracy* e da perda durante 30 épocas.

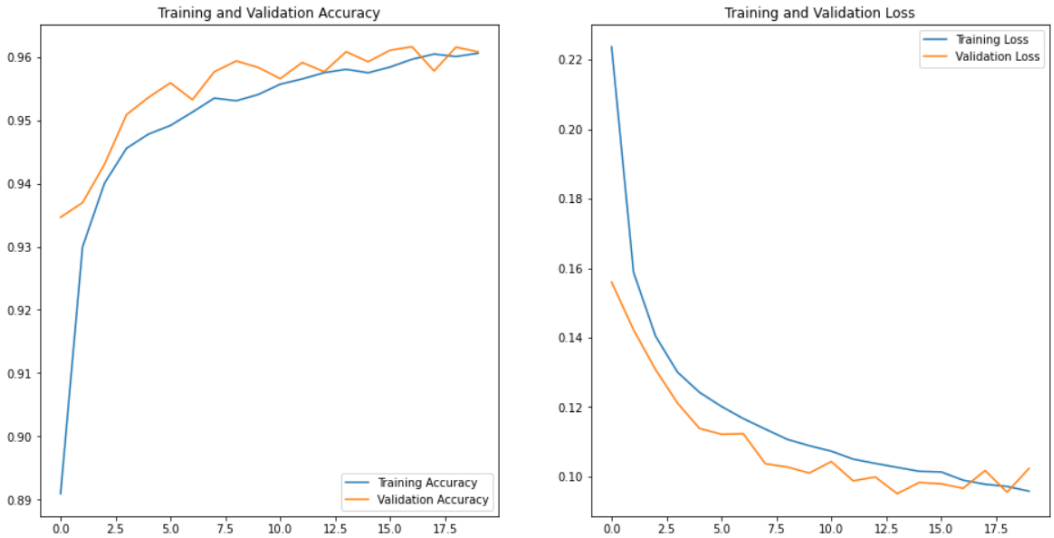


Figura 4.11: Gráfico do treino e validação da *accuracy* e da perda em binário

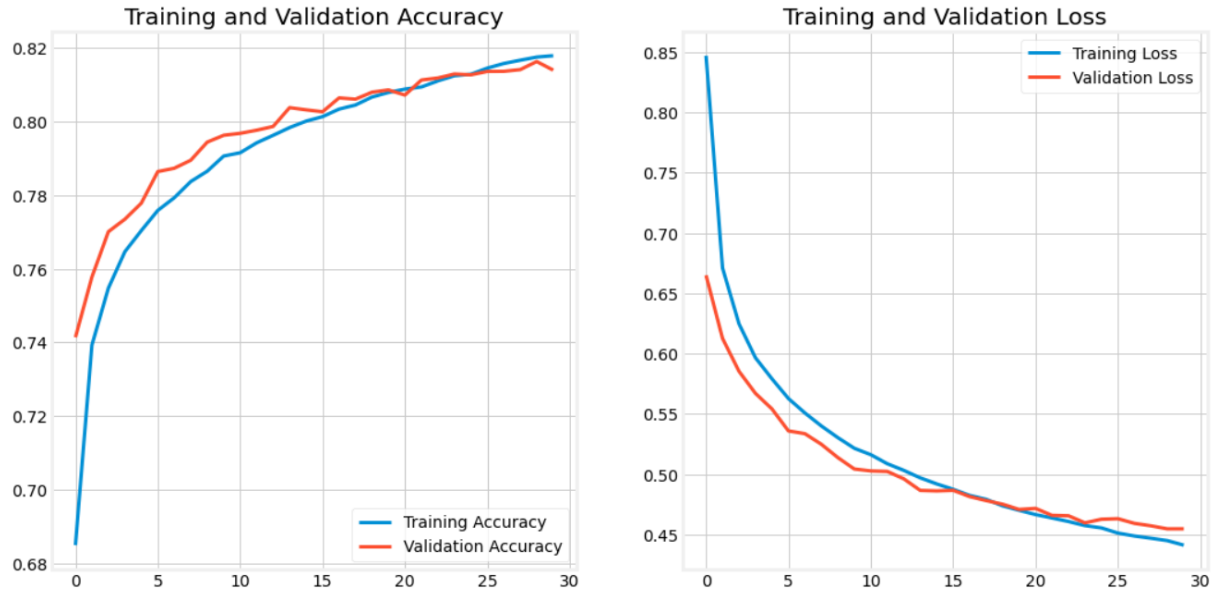


Figura 4.12: Gráfico do treino e validação da *accuracy* e da perda em multi-classe

4.3.6 Treino com CNN

Semelhante ao treino com MLP, para os treinos com CNN também foi preciso definir uma rede. Esta rede origina 874 mil parâmetros treináveis e 1728 parâmetros não treináveis. A figura 4.13 é a arquitetura e a figura 4.14 é a tabela sumariada da arquitetura.

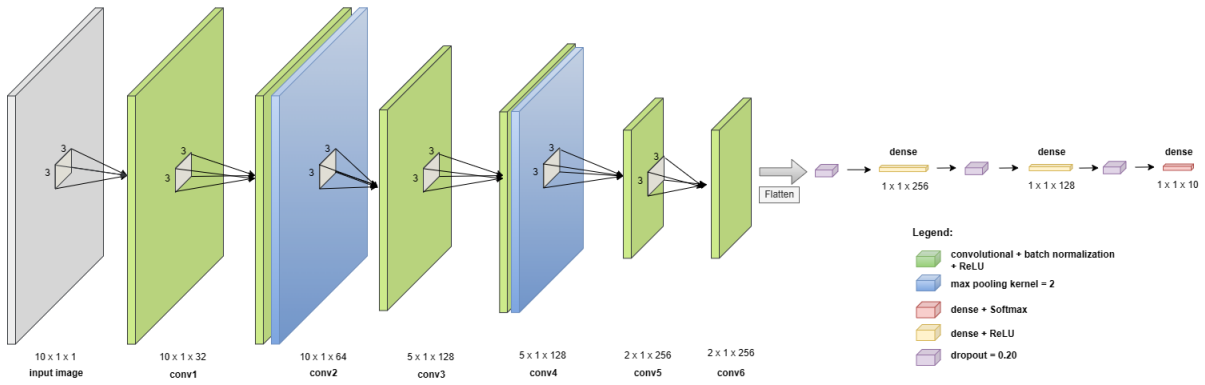


Figura 4.13: Arquitetura da CNN

Layer (type)	Output Shape	Param #
conv1d_6 (Conv1D)	(None, 30, 32)	128
conv1d_7 (Conv1D)	(None, 30, 32)	3104
batch_normalization_5 (BatchNormalization)	(None, 30, 32)	128
conv1d_8 (Conv1D)	(None, 30, 64)	6208
max_pooling1d_2 (MaxPooling1D)	(None, 15, 64)	0
batch_normalization_6 (BatchNormalization)	(None, 15, 64)	256
conv1d_9 (Conv1D)	(None, 15, 128)	24704
batch_normalization_7 (BatchNormalization)	(None, 15, 128)	512
conv1d_10 (Conv1D)	(None, 15, 128)	49280
max_pooling1d_3 (MaxPooling1D)	(None, 7, 128)	0
batch_normalization_8 (BatchNormalization)	(None, 7, 128)	512
conv1d_11 (Conv1D)	(None, 7, 256)	98560
batch_normalization_9 (BatchNormalization)	(None, 7, 256)	1024
conv1d_12 (Conv1D)	(None, 7, 256)	196864
batch_normalization_10 (BatchNormalization)	(None, 7, 256)	1024
flatten_1 (Flatten)	(None, 1792)	0
dropout_3 (Dropout)	(None, 1792)	0
dense_3 (Dense)	(None, 256)	459008
dropout_4 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 128)	32896
dropout_5 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 10)	1290

Total params: 875,498
Trainable params: 873,770
Non-trainable params: 1,728

Figura 4.14: Tabela sumariada da arquitetura de CNN

A rede também foi treinada utilizando o otimizador Adam e os mesmos hiperparâmetros (learning_rate=0.0001, beta_1=0.9 e beta_2=0.999), as 30 épocas e *batch_size*=32. Os resultados originados foram de 97.4% e 83% de *accuracy* em binário e multi-classe, respetivamente. Os gráficos 4.15 e 4.16 mostram os treinos e validações da *accuracy* e da perda durante 30 épocas.

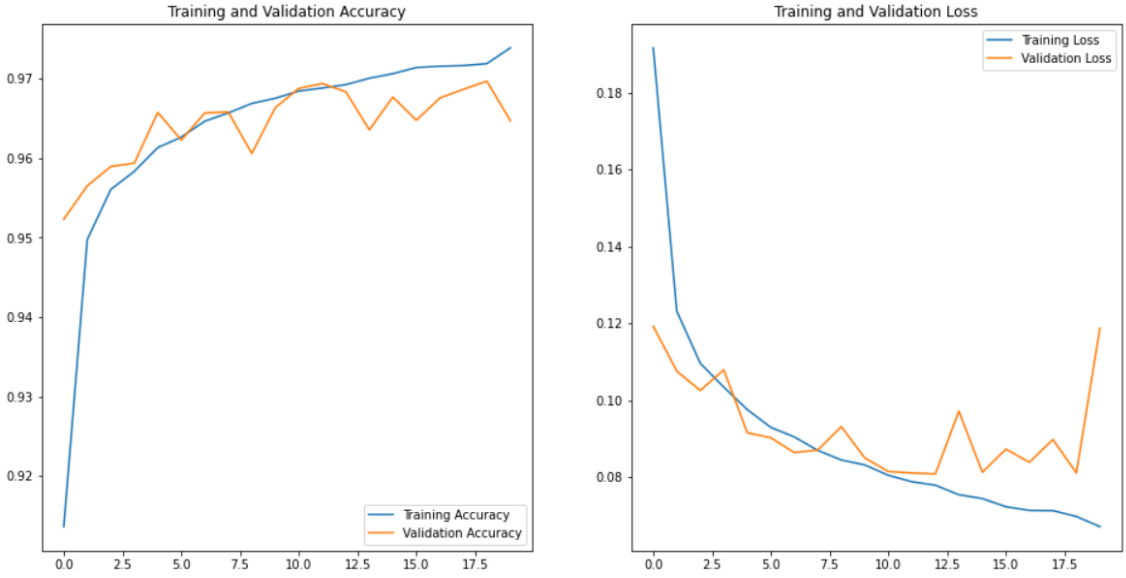


Figura 4.15: Gráfico do treino e validação da *accuracy* e da perda em binário

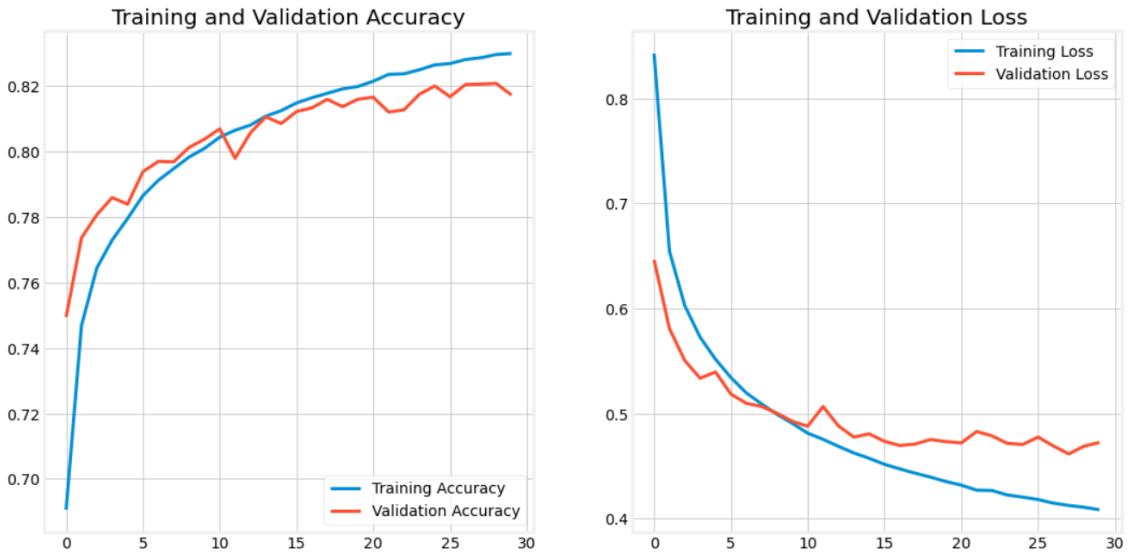


Figura 4.16: Gráfico do treino e validação da *accuracy* e da perda em multi-classe

4.3.7 Treino com RF + K-means

Neste treino, foi utilizado o RandomForestClassifier do '*sklearn*' com os hiperparâmetros n_estimators=225, min_samples_split=2, min_samples_leaf=1, max_features="sqrt", max_depth=250, bootstrap=True, n_jobs=-1. Após ter definido o classificador, utilizou-se uma técnica denominada de *cross-validation* para averiguar o modelo. Estes testes foram feitos com 10 vezes dando os seguintes valores, figura 4.17.

```
Cross-Validation Scores: [0.97943649 0.97573699 0.97637416 0.97667524 0.97926752 0.97549383
 0.97898873 0.97878849 0.97960775 0.97527532]
Average Score: 0.9775644519147259
```

(a) Cross-validation em binário

```
Cross-Validation Scores: [0.81938031 0.82392484 0.8211025  0.81919923 0.81705174 0.81500664
 0.82518122 0.82435961 0.81628677 0.81755995]
Average Score: 0.819905281020992
```

(b) Cross-validation em multi-classe

Figura 4.17: Valores obtidos com cross-validation

Depois de obter os valores, fez-se a previsão para ambas classificações e obteve-se os resultados mostrados nas figuras 4.18. As matrizes de confusão das figuras 4.19 e 4.20 servem para perceber melhor os resultados obtidos.

```
Accuracy:  0.9772271816359993
Precision:  0.9853401969561325
Recall:  0.9730356945518842
AUC:  0.9776879739947343
F1:  0.9791492910758965
```

(a) Classificação binária

```
Accuracy:  0.8671889233011477
Precision:  0.8916169247147822
Recall:  0.8671889233011477
F1:  0.8770321731100481
```

(b) Classificação multi-classe

Figura 4.18: Resultados de avaliação com RF

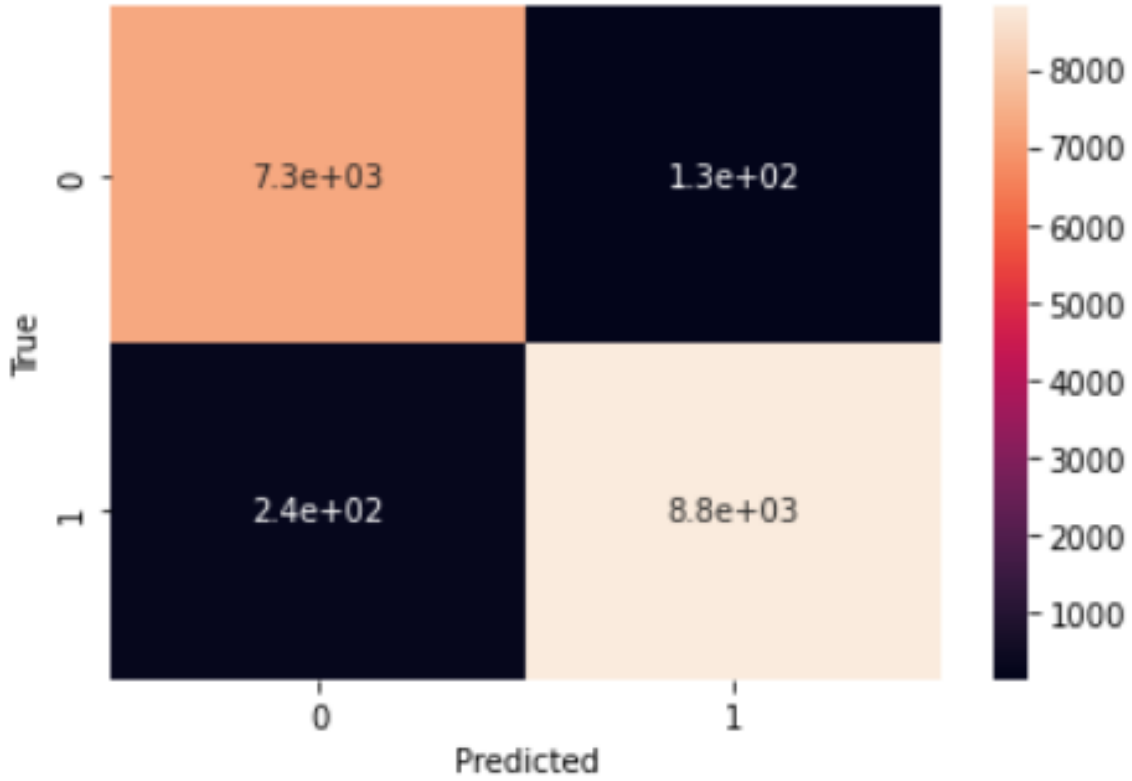


Figura 4.19: Matriz de confusão da RF em classificação binária

Na etapa seguinte (só para classificação multi-classe), selecionou-se os resultados classificados como 0, que são os tráfegos normais e das 30 *features* escolhidas para treinar, reduziu-se para as duas mais importantes. Esses valores foram guardados numa variável com o formato (4236, 2). Com os dados, utilizou-se o algoritmo KMeans para dividir em *clusters*. KMeans foi utilizado em conjunto com os hiperparâmetros n_init='auto', tol=0.0001, max_iter=300, random_state=32, n_clusters=19. Os hiperparâmetros foram obtidos a partir do RandomizedSearchCV.

Com estas condições todas, executou-se o algoritmo e originou-se o seguinte gráfico de *clusters*, gráfico 4.21, onde nos dois eixos (x e y) são as duas *features* selecionadas e os pontos são os dados.

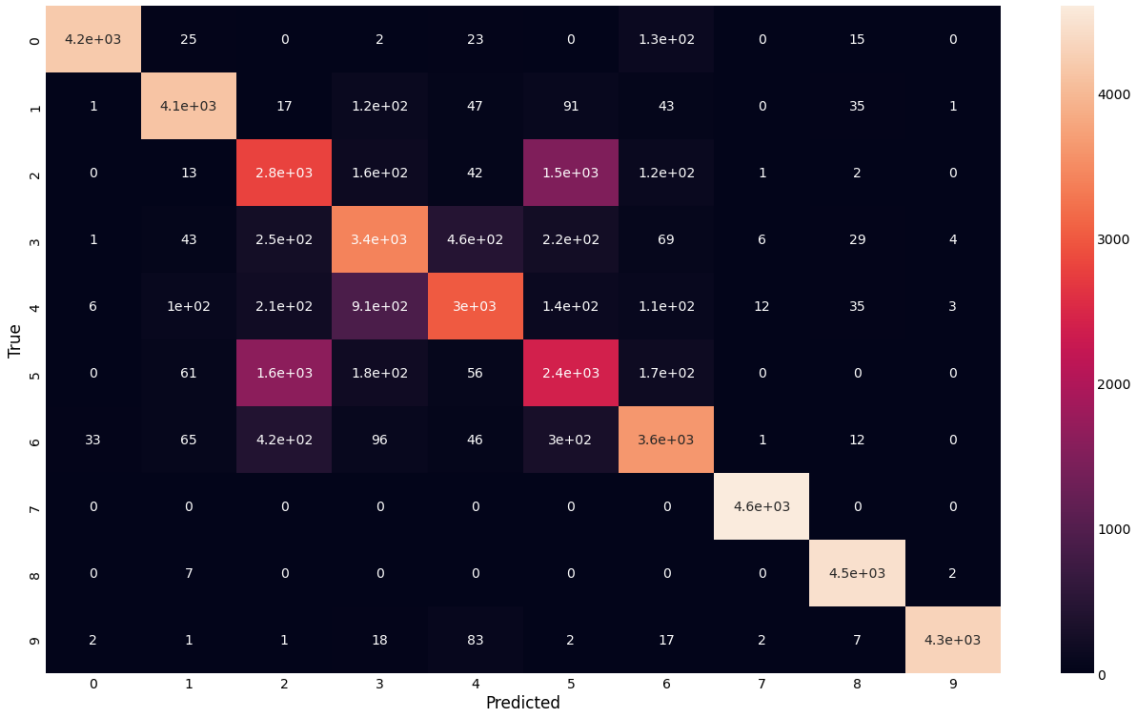


Figura 4.20: Matriz de confusão da RF em classificação multi-classe

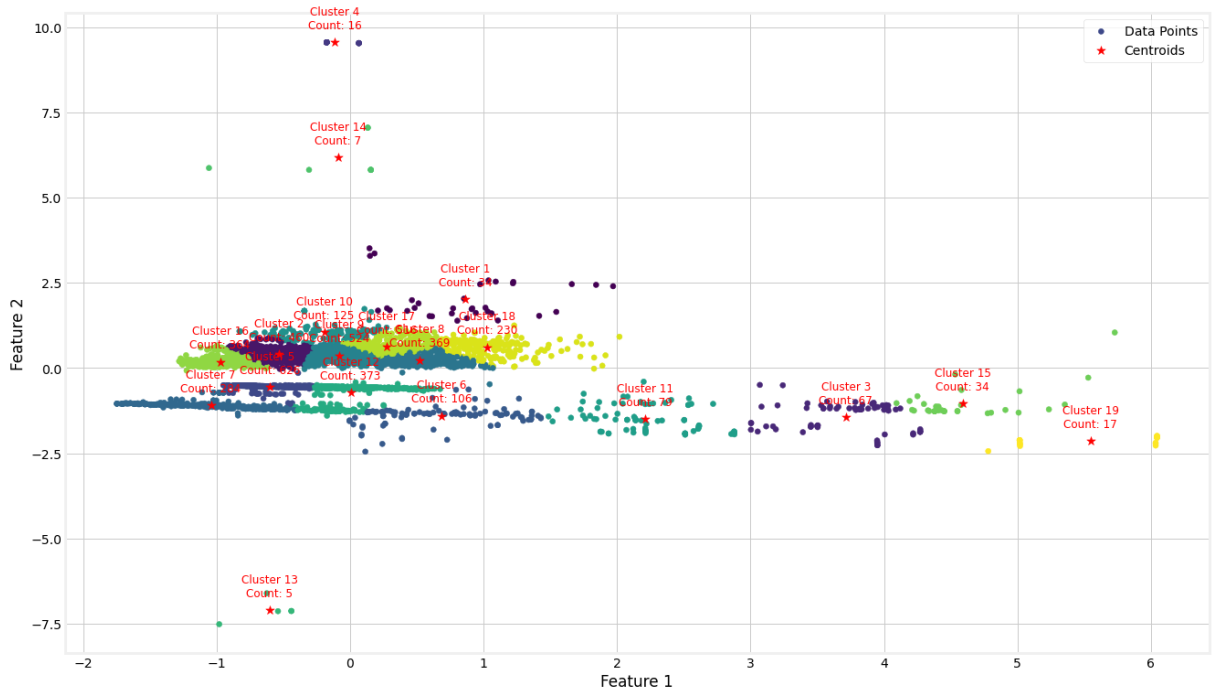


Figura 4.21: Gráfico dos *clusters*

De forma a facilitar a visualização e comparação dos resultados obtidos entre os vários algoritmos, apresenta-se a tabela 4.10.

Todos os hiperparâmetros apresentados neste capítulo, são resultados após várias tentativas com diferentes valores.

Tabela 4.10: Resultados obtidos dos algoritmos usados

Classificação	**Algoritmo**	***Accuracy***	***Precisão***	***Recall***	***AUC***	***F1-Score***
Binária	**SVM**	93.05%	96.45%	90.7%	93.31%	93.48%
	MLP	96.1%	-	-	-	-
	CNN	97.4%	-	-	-	-
	RF+Kmeans	97.72%	98.53%	97.3%	97.77%	97.41%
Multi-classe	**SVM**	79.05%	81.14%	79.05%	-	79.47%
	MLP	81.8%	-	-	-	-
	CNN	83%	-	-	-	-
	RF+Kmeans	86.72%	89.16%	86.72%	-	87.70%

5 Análise e discussão de resultados

Neste capítulo vamos analisar e discutir a implementação e os resultados obtidos no trabalho realizado.

Tal como referido no capítulo anterior, para a implementação deste trabalho foi escolhido a plataforma *Jupyter Lab* porque é uma plataforma muito simples e fácil de usar. Outro motivo é porque usa a linguagem *python* que é uma das linguagens mais utilizadas e contém várias bibliotecas que são muito úteis para este tipo de trabalho.

Escolheu-se o *dataset* UNSW-NB15 porque entre todas que se referiu na secção 3.3 é uma das melhores e apropriadas. Em comparação com o KDD Cup99 e NSL-KDD, o *dataset* UNSW-NB15 contém mais variedade de ataques, o que leva a que uma rede de computadores consiga detetar mais e em termos de equilíbrio, este tem mais entre as várias classes. Comparando com o CICIDS 2017, o *dataset* escolhido é mais simples de visualizar e compreender, pois o outro tem cerca de 3 milhões de dados e 83 *features*.

Durante a manipulação dos dados, foram utilizadas duas técnicas para seleção das *features*, *SelectKBest* e PCA. A partir da figura 5.1, podemos ver que quando é utilizado a técnica *SelectKBest*, os resultado originados são ligeiramente mais elevados.

```
Accuracy:  0.8187524954527305
Precision:  0.8258058997225101
Recall:  0.8187524954527305
F1:  0.8210534784873773
```

(a) Resultado com PCA

```
Accuracy:  0.8671889233011477
Precision:  0.8916169247147822
Recall:  0.8671889233011477
F1:  0.8770321731100481
```

(b) Resultado com SelectKBest

Figura 5.1: Comparação de resultados com as duas técnicas

PCA é mais utilizado quando queremos reduzir a dimensão das *features*, transformando as *features* originais num novo conjunto de *features* não correlacionados. *SelectKBest* é utilizado para seleção das *features*. Seleciona as k *features* mais importantes, mantendo as originais. Esse k foi definido como 30 porque durante os treinos, à medida que se reduzia,

os resultados eram piores.

Passando agora para os algoritmos, de acordo com a tabela 4.10, *Random Forest* foi o que obteve melhores resultados, tanto na classificação binária, como na classificação multi-classe. RF combina várias árvores de decisão para reduzir o *overfitting* e melhorar a generalização. Foi o algoritmo que demorou menos tempo a treinar. O SVM precisa de muito tempo para treinar e possui muita limitação quando usamos *datasets* grandes. É preciso definir bem os hiperparâmetros, pois pode levar com que o algoritmo seja computacionalmente caro, gastando muito tempo a encontrar os hiperplanos para separar os dados em diferentes classes.

Em relação ao MLP e CNN, estes dois algoritmos são precisos definir uma rede para poder treinar e isso pode levar muito tempo, pois para cada tipo de trabalho a rede varia, não existe uma que seja apropriada para todos os trabalhos. No trabalho realizado, as redes criadas foram precisas várias tentativas para chegar ao resultado obtido. As camadas densas, normalmente são utilizadas nas últimas camadas, pois estas retiram mais profundamente as características dos padrões que as camadas convolucionais obteram no início. Como a MLP só contém camadas densas, os resultados foram piores que a CNN. A partir dos gráficos 4.11, 4.12, 4.15 e 4.16, podemos ver que os treinos estão bem convergidos, não ocorrendo *overfitting*. Durante a convergência, ocorre algumas flutuações, isto é denominado de ruídos que podem ser pontos de dados que são significativamente diferentes do restante dos dados e não seguem os padrões esperados, variabilidade nas medições ou dados que não podem ser facilmente explicados ou controlados ou interferência ou flutuações aleatórias de fatores externos que afetam a qualidade dos dados. Uma das soluções podia ser a utilização de técnicas apropriadas de inicialização de peso para definir pesos iniciais de uma forma que reduza o risco de desaparecimento ou explosão de gradientes. Como este trabalho foi feito a partir de bibliotecas já fornecidas pela linguagem *python*, não foi possível experimentar alterar os valores dos pesos.

A figura a seguir, figura 5.2, é uma parte das previsões criadas pelo algoritmo RF, podendo comparar os valores obtidos com o *dataset* de teste fornecido.

	ct_src_ltm	ct_srv_dst	is_sm_ips_ports	attack_cat		SampleID	Class
135860	1	1	0	DoS	135860	135859	4
135861	1	2	0	Exploits	135861	135860	4
135862	18	17	0	Generic	135862	135861	9
135863	11	10	0	Generic	135863	135862	9
135864	3	5	0	Fuzzers	135864	135863	0
135865	34	34	0	Generic	135865	135864	9
135866	2	2	0	Fuzzers	135866	135865	0
135867	1	1	0	Reconnaissance	135867	135866	1
135868	18	17	0	Generic	135868	135867	9
135869	1	6	0	Fuzzers	135869	135868	0
135870	34	34	0	Generic	135870	135869	9
135871	16	24	0	Generic	135871	135870	9
135872	1	1	0	Exploits	135872	135871	4
135873	3	6	0	DoS	135873	135872	3
135874	1	1	0	Generic	135874	135873	3
135875	34	34	0	Generic	135875	135874	9
135876	34	33	0	Generic	135876	135875	9
135877	3	6	0	Generic	135877	135876	3
135878	1	1	0	Exploits	135878	135877	4
135879	1	1	0	Reconnaissance	135879	135878	1
135880	14	27	0	Generic	135880	135879	9
135881	17	16	0	Generic	135881	135880	9
135882	1	1	0	Exploits	135882	135881	9

Figura 5.2: Comparação das previsões criadas pelo RF

Do lado esquerdo são os dados do *dataset* de teste e do lado direito foi o ficheiro CSV criado. Os valores da coluna 'Class' estão referidos na subsecção 4.3.2. Comparando a coluna 'attack_cat' com a coluna 'Class' podemos ver que houve uns erros. Isto foi devido ao problema do desequilíbrio do *dataset*. Apesar de ter utilizado a técnica SMOTE, os valores criados são sintéticos, isto é, são baseados nos valores originais e por isso pode ter levado a um pouco de overfitting.

Para finalizar este capítulo, o modelo a ser utilizado para resolver o problema proposto será o modelo RF+K-means. Tal como analisado, foi o modelo que obteve melhores resultados em ambas as classificações.

6 Conclusão e Trabalho Futuro

6.1 Conclusão

A deteção de intrusões em redes de computadores é uma preocupação fundamental na segurança cibernética moderna. Este trabalho explorou o uso de ferramentas de *machine learning* como uma abordagem promissora para aprimorar a deteção de atividades maliciosas. Concluímos que as técnicas de *machine learning* têm o potencial de aumentar significativamente a precisão na identificação de intrusões em comparação com métodos tradicionais como por exemplo *proxy servers*, antivírus ou *firewalls*.

Ao longo deste trabalho, examinaram-se diferentes algoritmos de *machine learning*, técnicas de pré-processamento de dados e seleção de *features* que conduziram aos vários resultados apresentados nesta dissertação. Demonstrou-se que a escolha adequada desses elementos é crucial para o desempenho eficaz das ferramentas de *machine learning* na deteção de intrusões. Através dos resultados apresentados, o modelo RF+K-means foi o que obteve melhores resultados em ambas as classificações. A classificação binária, sendo a que tem menos classes para classificar, obteve melhores resultados. Pois, ao contrário da classificação multi-classe, esta só precisa de treinar e classificar se é um ataque ou não. Em comparação com os trabalhos dos outros autores, apesar do modelo criado não ser exatamente igual aos que existem, os resultados obtidos não diferem muito, havendo pouca variação devido ás técnicas utilizadas ou hiperparâmetros escolhidos.

Portanto, para concluir, pode-se afirmar que conseguiu-se atingir com sucesso o objetivo deste trabalho. As ferramentas de *machine learning* mostraram-se valiosas na deteção de intrusões em redes de computadores, oferecendo um aumento na capacidade de identificar ameaças. No entanto, é importante lembrar que essas ferramentas não são uma solução única e definitiva, e devem ser integradas numa estratégia de segurança mais ampla, incluindo medidas proativas de proteção e monitoramento contínuo. A segurança cibernética é uma área em constante evolução, e a aplicação eficaz de ferramentas de *machine learning*

requer vigilância e adaptação constantes para proteger as redes contra ameaças em constante mutação.

6.2 Trabalho Futuro

Para qualquer solução encontrada e posta em prática, há sempre um número crescente de novos problemas, e esta dissertação não é exceção. Para melhorar ainda mais os resultados obtidos, seria preciso investigar muitos mais algoritmos e técnicas e experimentar hiperparâmetros dentro dos algoritmos. Neste trabalho, apesar de ter chegado ao objetivo proposto, falta ainda testar o modelo criado numa rede de computador com tráfego real a tempo real e falando mais à frente, criar um sistema/aplicação com capacidade de realizar isso.

7 Bibliografia

[1] Doğukan Aksu, Serpil Üstebay, Muhammed Ali Aydin, and Tülin Atmaca. Intrusion detection with comparative analysis of supervised learning techniques and fisher score feature selection algorithm. In *Computer and Information Sciences: 32nd International Symposium, ISCIS 2018, Held at the 24th IFIP World Computer Congress, WCC 2018, Poznan, Poland, September 20-21, 2018, Proceedings 32*, pages 141–149. Springer, 2018.

[2] Majjed Al-Qatf, Yu Lasheng, Mohammed Al-Habib, and Kamal Al-Sabahi. Deep learning approach combining sparse autoencoder with svm for network intrusion detection. *Ieee Access*, 6:52843–52856, 2018.

[3] Zainab Alkhalil, Chaminda Hewage, Liqaa Nawaf, and Imtiaz Khan. Phishing attacks: A recent comprehensive study and a new anatomy. *Frontiers in Computer Science*, 3:563060, 2021.

[4] Saqr Mohammed Almansob and Santosh Shivajirao Lomte. Addressing challenges for intrusion detection system using naive bayes and pca algorithm. In *2017 2nd International Conference for Convergence in Technology (I2CT)*, pages 565–568. IEEE, 2017.

[5] Jadel Alsamiri and Khalid Alsubhi. Internet of things cyber attacks detection using machine learning. *International Journal of Advanced Computer Science and Applications*, 10(12), 2019.

[6] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised learning. *Machine learning techniques for multimedia: case studies on organization and retrieval*, pages 21–49, 2008.

[7] Konark Truptiben Dave. Brute-force attack 'seeking but distressing'. *Int. J. Innov. Eng. Technol. Brute-force*, 2(3):75–78, 2013.

[8] Peter Dayan, Maneesh Sahani, and Grégoire Deback. Unsupervised learning. *The MIT encyclopedia of the cognitive sciences*, pages 857–859, 1999.

[9] Nebrase Elmrabit, Feixiang Zhou, Fengyin Li, and Huiyu Zhou. Evaluation of machine learning algorithms for anomaly detection. In *2020 international conference on cyber security and protection of digital services (cyber security)*, pages 1–8. IEEE, 2020.

[10] Jun Gao, Li Li, Pingfan Kong, Tegawendé F Bissyandé, and Jacques Klein. Should you consider adware as malware in your study? In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 604–608. IEEE, 2019.

[11] Paolo Gasti, Gene Tsudik, Ersin Uzun, and Lixia Zhang. Dos and ddos in named data networking. In *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*, pages 1–7. IEEE, 2013.

[12] Stephen Grossberg. Recurrent neural networks. *Scholarpedia*, 8(2):1888, 2013.

[13] Neha Gupta et al. Artificial neural network. *Network and Complex Systems*, 3(1):24–28, 2013.

[14] JW Haines, RP Lippmann, DJ Fried, MA Zissman, E Tran, and SB Boswell. 1999 darpa intrusion detection evaluation: Design and procedures. *MIT Lincoln Laboratory Technigcal Report*, 1062:26, 2001.

[15] MA Jabbar, Rajanikanth Aluvalu, et al. Rfaode: A novel ensemble intrusion detection system. *Procedia computer science*, 115:226–234, 2017.

[16] Laveen N Kanal. Perceptron. In *Encyclopedia of Computer Science*, pages 1383–1385. 2003.

[17] Latifur Khan, Mamoun Awad, and Bhavani Thuraisingham. A new intrusion detection system using support vector machines and hierarchical clustering. *The VLDB journal*, 16:507–521, 2007.

[18] Sotiris B Kotsiantis. Decision trees: a recent overview. *Artificial Intelligence Review*, 39:261–283, 2013.

[19] Youguo Li and Haiyan Wu. A clustering method based on k-means algorithm. *Physics Procedia*, 25:1104–1109, 2012.

[20] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[21] Yanli Liu, Yourong Wang, and Jian Zhang. New machine learning algorithm: Random forest. In *Information Computing and Applications: Third International Conference, ICICA 2012, Chengde, China, September 14-16, 2012. Proceedings 3*, pages 246–252. Springer, 2012.

[22] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (MilCIS)*, pages 1–6. IEEE, 2015.

[23] Anitta Patience Namanya, Andrea Cullen, Irfan U Awan, and Jules Pagna Disso. The world of malware: An overview. In *2018 IEEE 6th international conference on future internet of things and cloud (FiCloud)*, pages 420–427. IEEE, 2018.

[24] Mukrimah Nawir, Amiza Amir, Ong Bi Lynn, Naimah Yaakob, and R Badlishah Ahmad. Performances of machine learning algorithms for binary classification of network anomaly detection system. In *Journal of Physics: Conference Series*, volume 1018, page 012015. IOP Publishing, 2018.

[25] Leonardo Noriega. Multilayer perceptron tutorial. *School of Computing. Staffordshire University*, 4(5):444, 2005.

[26] Gavin O'Gorman and Geoff McDonald. *Ransomware: A growing menace*. Symantec Corporation Arizona, AZ, USA, 2012.

[27] Atilla Özgür and Hamit Erdem. A review of kdd99 dataset usage in intrusion detection and machine learning between 2010 and 2015. 2016.

[28] Ranjit Panigrahi and Samarjeet Borah. A detailed analysis of cicids2017 dataset for designing intrusion detection systems. *International Journal of Engineering & Technology*, 7(3.24):479–482, 2018.

[29] Muhammad Shakil Pervez and Dewan Md Farid. Feature selection and intrusion classification in nsl-kdd cup 99 dataset employing svms. In *The 8th International Conference on Software, Knowledge, Information Management and Applications (SKIMA 2014)*, pages 1–6. IEEE, 2014.

[30] Derek A Pisner and David M Schnyer. Support vector machine. In *Machine learning*, pages 101–121. Elsevier, 2020.

[31] Ujwala Ravale, Nilesh Marathe, and Puja Padiya. Feature selection based hybrid anomaly intrusion detection system using k means and rbf kernel function. *Procedia Computer Science*, 45:428–435, 2015.

[32] Jiadong Ren, Jiawei Guo, Wang Qian, Huang Yuan, Xiaobing Hao, Hu Jingjing, et al. Building an effective intrusion detection system by using hybrid data optimization based on machine learning algorithms. *Security and communication networks*, 2019, 2019.

[33] Sathyanarayanan Revathi and A Malathi. A detailed analysis on nsl-kdd dataset using various machine learning techniques for intrusion detection. *International Journal of Engineering Research & Technology (IJERT)*, 2(12):1848–1853, 2013.

[34] Sandeep Shah, Pramita Sree Muhuri, Xiaohong Yuan, Kaushik Roy, and Prosenjit Chatterjee. Implementing a network intrusion detection system using semi-supervised support vector machine and random forest. In *Proceedings of the 2021 ACM southeast conference*, pages 180–184, 2021.

[35] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.

[36] Jagsir Singh and Jaswinder Singh. A survey on machine learning-based malware detection in executable files. *Journal of Systems Architecture*, 112:101861, 2021.

[37] Dimitris Sklavounos, Alexandros Leondakianakos, and Aloysius Edoh. Statistical process control method for cyber intrusion detection (ddos, u2r, r2l, probe). *International Journal of Cyber-Security and Digital Forensics*, 8(1):82–89, 2019.

[38] Thomas F Stafford and Andrew Urbaczewski. Spyware: The ghost in the machine. *The Communications of the Association for Information Systems*, 14(1):49, 2004.

[39] S Ravi Subramanya and Natraj Lakshminarasimhan. Computer viruses. *IEEE potentials*, 20(4):16–19, 2001.

[40] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*, pages 1–6. Ieee, 2009.

[41] Jianxin Wu. Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, 5(23):495, 2017.

[42] Yihan Xiao, Cheng Xing, Taining Zhang, and Zhongkai Zhao. An intrusion detection model based on feature reduction and convolutional neural networks. *IEEE Access*, 7:42210–42219, 2019.

[43] Ruibo Yan, Xi Xiao, Guangwu Hu, Sancheng Peng, and Yong Jiang. New deep learning method to detect code injection attacks on hybrid applications. *Journal of Systems and Software*, 137:67–77, 2018.

[44] Chuanlong Yin, Yuefei Zhu, Jinlong Fei, and Xinzheng He. A deep learning approach for intrusion detection using recurrent neural networks. *Ieee Access*, 5:21954–21961, 2017.

[45] Zhongheng Zhang. Introduction to machine learning: k-nearest neighbors. *Annals of translational medicine*, 4(11), 2016.

[46] ZHU Zhenfang. Study on computer trojan horse virus and its prevention. *International Journal of Engineering and Applied Sciences*, 2(8):257840, 2015.