Faculdade de Ciências da Universidade do Porto
Departamento de Ciência de Computadores

Estágio/Projeto (CC3050)

Fevereiro - Julho 2023

Frequency or Sequence, does it influence intrusion detection?

Autor:
Marco Gonçalves (202007512)

Orientador:
João Paulo da Conceição Soares

Docente:
Eduardo Resende Brandão Marques

Conteúdos

1 Introdução

Os Sistemas de Deteção de Intrusões (*Intrusion Detection Systems)* (IDS) são uma ferramenta imprescindível na segurança informática. Estes sistemas monitorizam e analisam o comportamento do tráfego de rede ou do sistema, para identificar atividades suspeitas ou maliciosas [1]. Face ao aumento exponencial do uso da Internet [2] e dos ataques cibernéticos que com ela vêm, diferentes maneiras de proteger os utilizadores de possíveis mal-feitores têm sido estudadas e implementadas (tais como *Firewalls* e *Antivírus*), no entanto, a maiorias destas soluções sofrem todas de um problema grande, pois requerem que o tipo de ataque seja normalmente conhecido, tendo bastantes dificuldades em lidar com ataques desconhecidos, ou que não foram considerados na sua conceção. Por estas razões, a implementação de um *IDS* eficaz tornou-se uma necessidade de enorme importância para qualquer organização, pelo que é uma área de estudo bastante desenvolvida, tanto por instituições académicas [3], como também por parte da indústria [4] [5] [6], com uma multitude de soluções diversas e abrangendo vários tipos de sistemas.

Este projeto centra-se na criação e investigação de diferentes modelos que um IDS pode usar para detetar intrusões, usando *Machine Learning* para detetar padrões comuns que possam ser considerados perigosos, permitindo assim detetar possíveis ataques, mesmo que não sejam conhecidos. Iremos investigar diversos modelos para averiguar as suas vantagens e desvantagens, assim como melhorar a precisão e a eficácia na deteção de ameaças. Os modelos a serem utilizados incluem *K-Nearest Neighbors (KNN)*, *Naive Bayes*, *Random Forest*, *Support Vector Machines (SVM)* e *Long Short Term Memory (LSTM)*. É importante notar que a utilização das diversas alternativas traz consigo certas vantagens e desvantagens que devem ser consideradas, dependendo da função desejada para o sistema.

Os diferentes modelos atuam nas chamadas de sistema (*syscalls*) que diferentes processos chamam durante a sua execução, visando detetar padrões que possam indicar possíveis ataques em tempo real.

Os primeiros quatro modelos (*KNN*, *Naive Bayes*, *Random Forest* e *SVM*) são baseados na frequência das *syscalls*, enquanto que o *LSTM* é um modelo baseado na sequência em si. Ambas as abordagens usam uma janela escolhida pelo utilizador. Iremos abordar com mais detalhe este assunto numa secção futura.

Para treinar os modelos, será usado o *ADFA-LD* [7], um conjunto de milhares de sequências de *syscalls*, algumas representado ataques e outras não, criado pela academia da Força de Defesa Australiana. Este *dataset* é considerado dos mais completos, sendo usado para treinar e verificar a grande maioria dos *IDS* (e os seus derivados [8]).

Esperamos que este relatório proporcione um entendimento profundo dos processos e desafios envolvidos na investigação de diferentes modelos a serem usados. Ao longo deste relatório, vamos discutir as etapas e considerações na implementação destes modelos e analisar os respetivos resultados obtidos.

1.1 Estrutura do documento

O propósito e estrutura deste documento estão estabelecidos para facilitar a compreensão completa do trabalho desenvolvido neste projeto. A seguir, apresentamos um resumo da estrutura do documento.

Na Secção 1, "Introdução", introduz o problema que estamos a resolver, fornecendo uma contextualização do campo da deteção de intrusões e sua relevância.

Na Secção 2, "Estado da Arte", é a revisão da literatura e técnicas atuais no campo de deteção de intrusões. Esta secção permite posicionar o nosso trabalho dentro do contexto da pesquisa atual. Na Secção 3, "Descrição do Trabalho", detalha o método utilizado para abordar o problema. Esta inclui informações sobre os dados utilizados, os algoritmos e modelos usados, assim como qualquer pré-processamento realizado nos dados.

Na Secção 4, "Resultados e Validação", são apresentados os resultados obtidos a partir dos modelos desenvolvidos, bem como uma comparação entre eles, avaliando a eficácia de cada um.

Por fim, na Secção 5, "Conclusão e Trabalho Futuro", resume os resultados do trabalho, além de sugerir direções possíveis para pesquisas futuras.

A leitura deste documento seguindo a ordem proposta irá proporcionar um entendimento claro e conciso do projeto, dos métodos empregados e dos resultados obtidos.

2 Estado da Arte

Neste segmento, exploramos o atual "Estado da Arte" na área de Sistemas de Deteção de Intrusões (*IDS*). O objetivo desta secção é estabelecer o contexto da pesquisa atual, destacando os avanços recentes, as tecnologias predominantes e as metodologias adotadas no domínio dos *IDS*. Ao longo desta discussão, destacamos os pontos fortes e fracos das técnicas existentes e identificamos as lacunas que ainda persistem, o que motivará o desenvolvimento das novas abordagens que serão apresentadas posteriormente neste estudo. Vamos começar este capítulo com uma pequena introdução aos *IDS*, seguidos da sua complexidade, finalizando com uma revisão dos dois principais tipos de *IDS*: Sistemas de Deteção de Intrusões baseados em **Assinatura** (*SIDS*) e Sistemas de Deteção de Intrusões baseados em **Anomalias** (*AIDS*).

2.1 Sistemas de Deteção de Intrusão

O primeiro conceito de *IDS* foi vagamente definido num artigo por *James Anderson* [9], no entanto, estes eram métodos irrealistas de serem implementados com a tecnologia da altura, pelo que no final da década de 80, com o início do crescimento explosivo da *Internet*, começou-se a desenvolver métodos mais relevantes, e com o investimento na sua investigação durante os últimos anos, vieram a tornar-se métodos sofisticados e eficientes, essenciais na segurança informática de variados setores.

Na Figura 1, descrevemos de maneira breve o funcionamento geral de um *IDS*. Estes são compostos por 4 componentes, sendo o primeiro destes o componente de **Captura de Eventos**. Este tem apenas a funcionalidade de ficar à espera de eventos como por exemplo, *syscalls*, eventos de teclado, rato e pacotes de rede, passando-os depois para os processos de **Análise de Eventos** e **Armazenamento**. Na parte da secção de análise, os eventos são analisados para investigar possíveis ataques. No caso de um ataque ser detetado, é ativada a camada de **Contramedida**, onde vários protocolos podem ser ativados para proteger o sistema, tais como gerar alarmes, reconfiguração de *firewalls* e até mesmo isolar o sistema atacado do resto da *network*, para prevenir que o ataque possivelmente se espalhe. O componente de **Armazenamento** é usado para guardar todos os eventos detetados e criados por ambas as camadas a ele ligadas, não só para manter informação para uma análise futura, como também para possivelmente usar esses

dados para posteriormente melhorar os modelos.

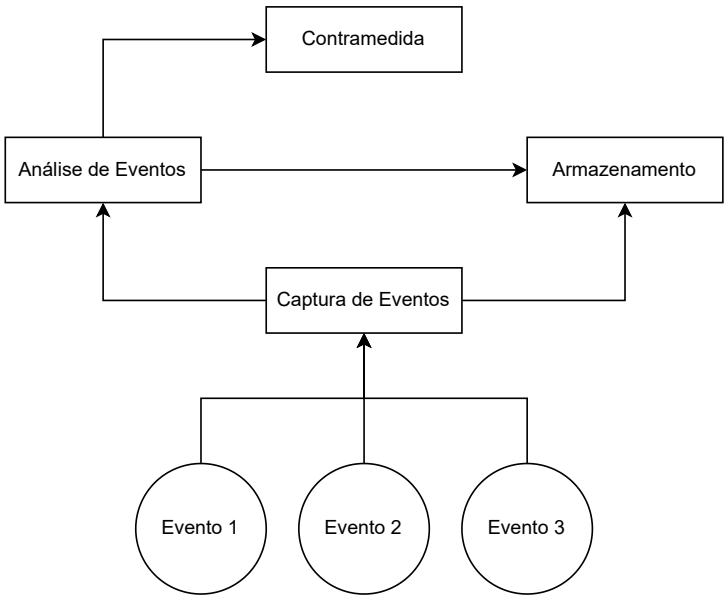

Figura 1: Funcionamento geral de um *IDS*

2.2 Complexidade e dificuldades dos *IDS*

A arquitetura e implementação de um *IDS* é complicada por uma multitude de razões, mas as mais problemáticas são a incorreta deteção de ataques e a não deteção dos mesmos. Estes dois problemas podem ser quantificados analisando os **falsos positivos** e **falsos negativos** que os modelos obtêm. Falsos negativos são a quantidade de vezes que o modelo classificou um dado evento como sendo inofensivo, quando este não o era. Falsos positivos, por outro lado, é quando o modelo classifica um evento como sendo um ataque, quando este é na realidade um uso benigno, podendo possivelmente correr processos e medidas que possam afetar o uso do utilizador. Por estas razões, queremos reduzir ao máximo ambos os valores, pois é claro que ambos afetam a efetividade dum *IDS*, por diferentes razões.

2.3 Técnicas principais para deteção de intrusões

Existem vários métodos para deteção de intrusões, no entanto, os mais comuns são deteção por **anomalias** e por **assinatura**. É importante notar que com a evolução da complexidade dos modelos, a maioria destas separações começam a ser cada vez menos relevantes, pois um *IDS* pode usar vários tipos de técnicas de deteção, por exemplo, usar *machine learning* para detetar ataques para os quais uma assinatura ainda não existe. Um *IDS* com esse comportamento seria efetivamente baseado em anomalias e assinaturas ao mesmo tempo.

2.3.1 Assinaturas

Os sistemas de deteção de intrusões baseados em assinaturas (*Signature-Based Intrusion Detection Systems*) (*SIDS*) são uma forma comum de *IDS* que utilizam um conjunto predefinido de regras ou padrões para identificar possíveis intrusões [1]. Estes padrões, conhecidos como "assinaturas", são geralmente derivados de ataques conhecidos e documentados e servem para identificar comportamentos anómalos ou maliciosos.

Devido a esta arquitetura, um sistema *SIDS* permite obter taxas de falsos positivos muito baixas (mas não nulas, pois podem existir comportamentos benignos que podem ser similares a assinaturas conhecidas), visto que geralmente só aciona alarmes quando existe algum padrão que corresponde com um ataque já conhecido, no entanto, isto também traz a sua maior fraqueza, pois estes sistemas não são geralmente capazes de detetar ataques não conhecidos, ou para os quais não foram registadas assinaturas. Este efeito pode ser mitigado com atualizações regulares à base de dados de assinaturas, assegurando-se que as atualizações vêm de uma fonte segura e confiável, no entanto, nunca teremos um sistema totalmente seguro, visto que ataques novos não vão, por definição, ter assinaturas registadas ainda.

2.3.2 Anomalias

Os sistemas de deteção de intrusões baseados em anomalias (*Anomaly-Based Intrusion Detection Systems*) (*AIDS*) representam uma abordagem alternativa aos *SIDS*. Em vez de depender de assinaturas de ataques conhecidos, os *AIDS* estabelecem um modelo de "comportamento normal" e depois identificam atividades que se desviam significativamente desse modelo como potenciais intrusões [1].

Estes modelos utilizam grandes quantidades de dados recolhidos, *machine learning*, e vários métodos estatísticos para estabelecer o que é considerado um comportamento normal, podendo até mesmo terem diferentes definições de normalidade para diferentes utilizadores e sistemas. Pela sua generalidade, *AIDS* são uma boa maneira de complementar as fraquezas dos *SIDS*, visto que estes não precisam de assinaturas conhecidas, podendo detetar ataques cujo um modelo puramente baseado em assinaturas não detetaria.

É importante notar, que uma das maiores desvantagens dos modelos *AIDS* é a definição de comportamento normal, visto que diferentes utilizadores podem ter comportamentos normais completamente diferentes, e até é completamente plausível que um dado utilizador tenha diferentes comportamentos dependendo da hora do dia, por exemplo, um utilizador em trabalho remoto terá geralmente um uso completamente diferente que o mesmo numa utilização fora do seu horário de trabalho. Com isto, vem também outra fraqueza dos sistemas *AIDS*, pois por norma, estes costumam ter uma taxa de falsos positivos maior que os *SIDS*, devido à sua natureza mais generalizada.

3 Descrição do Trabalho

Nesta secção, detalharemos o trabalho que foi realizado no contexto deste estudo. O objetivo é proporcionar uma visão completa e compreensível do processo de pesquisa e desenvolvimento adotado, bem como dos métodos utilizados para chegar às nossas conclusões. Descreveremos as etapas e decisões cruciais tomadas no decorrer do trabalho, com um foco particular na seleção e aplicação dos algoritmos de *Machine Learning*. Iniciaremos com uma explicação dos dados utilizados e como foram preparados para o uso neste estudo. Posteriormente, descreveremos as metodologias de treino e os algoritmos selecionados, explicando as razões para sua escolha e o processo de ajuste dos modelos.

3.1 Conjunto de dados usado - *ADFA-LD*

O *dataset* usado neste projeto, *ADFA-LD* foi criado pela Academia das Forças de Defesa Australianas (*Australian Defence Forces Academy*) (*ADFA*). Este conjunto foi criado usando as chamadas de sistema baseadas num sistema *Ubuntu 11.04*, e para brevidade, os nomes das chamadas foram substituídos pelo seu código correspondente, que pode ser confirmado no ficheiro *unistd.h* em sistemas *Linux*.

Os dados em estudo são organizados em três categorias distintas: Validação, Treino e Ataque. As categorias de Validação e Treino encapsulam dados provenientes de uma utilização normal do sistema, sem quaisquer sinais de atividade maliciosa. Por outro lado, a categoria de Ataque, como o próprio nome sugere, engloba dados recolhidos quando o sistema está sob o efeito de um ataque. É importante notar que os dados categorizados como "Ataque" estão ainda subdivididos de acordo com o tipo específico de ataque em curso, algo do qual falaremos mais à frente.

A tabela seguinte mostra a distribuição das 3 categorias neste conjunto de dados:

	Chamadas de Sistemas
Dados de Treino	308 007
Dados de Validação	2 122 085
Dados de Ataque	317 388
Total	2 747 550

Tabela 1: Distribuição das *syscalls* presentes em cada categoria do *ADFA-LD*

E na próxima tabela, mostramos a distribuição que cada ataque tem no conjunto de dados:

Ataque	***Payload***	**Vetor**	**Contagem**
Hydra-FTP	Força bruta da senha	FTP by Hydra	162
Hydra-SSH	Força bruta da senha	SSH Hydra	176
Adduser	Adicionar novo super utilizador	Executável envenenado do lado do cliente	91
Java-Meterpreter	Meterpreter baseado em Java	Exploração da vulnerabilidade Tikiwiki	124
Meterpreter	Linux Meterpreter payload	Executável envenenado do lado do cliente	75
Webshell	C100 webshell	Vulnerabilidade de inclusão de ficheiro remoto php	118

Tabela 2: Distribuição dos vários ataques presentes no *ADFA-LD*

3.2 Metodologia de treino

Nesta subsecção, vamos entrar em detalhes sobre os passos de pré-processamento que são fundamentais para a preparação do conjunto de dados, tornando-o adequado para a aplicação pelos diversos estudados. É importante entender que esta etapa de pré-processamento é absolutamente crucial, pois prepara o terreno para uma implementação eficiente dos modelos subsequentes.

O processo de pré-processamento começa com a tarefa de dividir o conjunto de dados em diversas janelas de tamanho fixo. Este passo é importante, pois permite analisar os dados em blocos mais pequenos, o que é especialmente relevante quando lidamos com sequências de dados que têm uma sequência temporal, o que será abordado mais tarde. Além disso, a divisão dos dados em janelas também possibilita a contagem da frequência de cada *syscall* de uma janela dentro do conjunto de dados. Esta contagem de frequência fornece uma perspetiva vital sobre os padrões de uso dos dados e é uma ferramenta útil para a deteção de anomalias ou comportamentos suspeitos.

Posteriormente, depois de dividir os dados em janelas e contar as frequências, passamos para a próxima etapa do processo de pré-processamento, que é a divisão dos dados em *subsets* distintos. Esta etapa permite a criação de novos cenários de teste de uma forma consistente, o que é fundamental para a avaliação rigorosa dos modelos. Ao dividir os dados em *subsets*, podemos garantir que cada modelo é testado e validado em diferentes cenários, permitindo uma análise robusta e completa do desempenho do modelo.

3.2.1 Pré-Processamento

Para o pré-processamento do *dataset*, decidimos concatenar os vários ficheiros apenas num ficheiro, para evitar várias chamadas ao sistema de abertura, levando assim a uma maior eficiência e rapidez nos processos seguintes. Depois disto, foi analisada cada entrada do ficheiro, e para cada entrada, dividimos em múltiplas linhas, cada uma contendo exatamente K *syscalls*, onde K é o tamanho da janela desejado. Este ficheiro será usado como base para todos os modelos, no entanto, o único modelo a usa-lo sem modificações é o *Long Short Term Memory*.

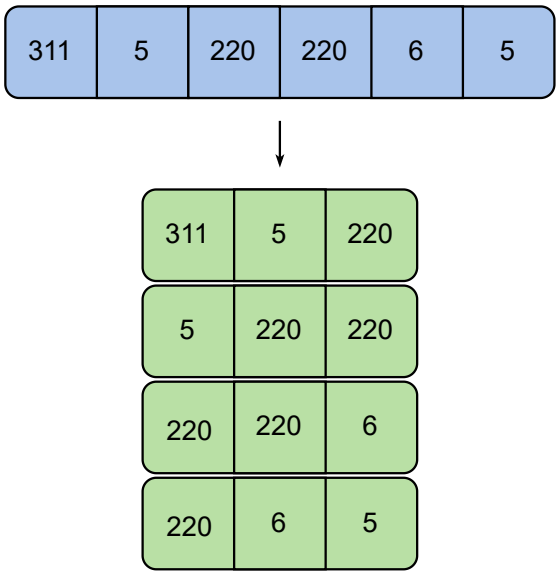

Figura 2: Janelas criadas para uma entrada com tamanho de janela 3

Após a fase de criação das janelas, iniciamos a construção de um novo ficheiro com o objetivo de contar as ocorrências de cada *syscall* dentro de cada janela criada. A frequência de cada *syscall* é armazenada na entrada correspondente da janela, comportando-se como um vetor.

No entanto, para a implementação eficaz desta técnica, é importante ter uma noção precisa da *syscall* de maior valor presente no sistema. Este conhecimento é essencial para assegurar a cobertura completa de todas as *syscalls* durante o processo de contagem de ocorrências. Assim, para obter esta informação, investigamos o arquivo *unistd.h*, que contém todas as *syscalls* existentes no sistema. Ao avaliar este ficheiro, somos capazes de identificar a *syscall* de maior valor, que neste caso específico, foi determinado como sendo 345.

Janela Original	5 1 3 1 2 2 1
Frequencia	0 3 2 1 0 1 0 0 0

Figura 3: Frequência de uma janela (assumindo que a syscall máxima é 8 para uma melhor leitura)

3.2.2 Divisão em *subsets*

Finalmente, depois do pré-processamento do *dataset* original, começamos a criar os *subsets* de treino, tanto para frequência como para sequencia. Para este efeito, são escolhidos 50% de entradas categorizadas como não sendo um ataque, e 50% como sendo um ataque. É importante notar que todos os ataques são escolhidos com a mesma percentagem, para evitar possíveis *biases* caso existam consideravelmente mais tipos de um ataque, do que de outros. Isto serve para aumentar a fidelidade dos resultados, visto que os dados são escolhidos de forma randomizada, podemos correr várias vezes os modelos com *subsets diferentes*.

3.3 Modelos treinados e resultados

Nesta subsecção, apresentaremos os modelos que foram treinados com nosso conjunto de dados processado. Discutiremos as especificidades de cada algoritmo, incluindo as suas características únicas, vantagens e limitações.

3.3.1 *K-Nearest Neighbors (KNN)*

O algoritmo de *KNN* é baseado numa ideia simples, em que uma qualquer observação ainda não rotulada é avaliada consoante os seus k vizinhos mais próximos, escolhendo a classe maioritária dos mesmos. Por esta razão, o *KNN*, apesar de criar um modelo muito rápido de treinar, tem uma grande desvantagem, pois possui baixa precisão para a classe minoritária, o que pode ser um problema dependendo do problema a resolver.

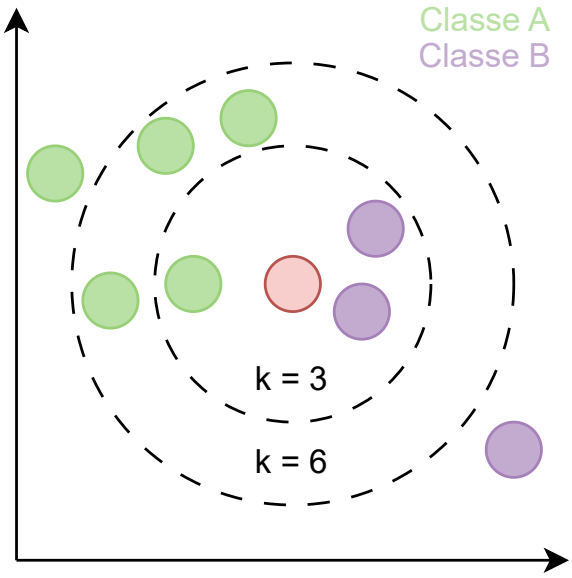

Figura 4: Exemplo de funcionamento do *KNN*

Como podemos ver pelo exemplo em cima, a escolha do k é extremamente importante e crucial a cada *dataset* especifico, pois dependendo do k, a nova observação vai ser da classe A, ou da classe B, para $k = 3$ e $k = 6$ respetivamente.

Ao aplicar o algoritmo *KNN* ao conjunto de dados de frequências, começamos por rotular cada ponto de dados de acordo com as informações fornecidas no conjunto dado. Este conjunto rotulado de pontos de dados forma o nosso espaço de características.

No treinamento do modelo, o *KNN* aprende a classificar uma nova observação baseando-se na proximidade desta observação a outras observações no espaço de características. A observação é classificada com a etiqueta da maioria dos seus K vizinhos mais próximos no espaço de características.

Para implementar o *KNN*, é crucial escolher um valor apropriado para K, que é o número de vizinhos a serem considerados. Este parâmetro é normalmente otimizado para minimizar a taxa de erro no conjunto de dados de validação.

3.3.2 *Naive Bayes*

O algoritmo de *Naive Bayes* é uma maneira simples e eficaz de lidar com grandes conjuntos de dados. O algoritmo começa por fazer a suposição "ingénua" (dai o nome) de que todas as caraterísticas são independentes, isto é, a existência de uma caraterística não afeta a presença das outras. Claramente, esta suposição parece muito restritiva, visto que existem vários casos em que isto não é verdade, por exemplo, num conjunto de dados médicos, a probabilidade de um paciente sofrer de uma doença cardíaca, está diretamente relacionada com o facto do paciente sofrer de obesidade ou não [10]. No entanto, em 2004, um estudo mostrou que existem razões teóricas para a otimalidade do *Naive Bayes*, dependendo de certas propriedades do conjunto de dados [11].

A ideia fundamental do algoritmo usa o teorema de *Bayes*:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Figura 5: Teorema de *Bayes*

Este teorema diz-nos a probabilidade de A acontecer sabendo que B aconteceu. No contexto do nosso projeto, o *Naive Bayes* iria primeiro ser treinado com dados já rotulados, formando assim as probabilidades para cada classe ("ataque" ou "não ataque"). Depois do processo de treino, quando o algoritmo recebe alguma ocorrência não rotulada, o algoritmo calcula as probabilidades de pertencer a cada classe, e usa a classe com maior percentagem para avaliar a ocorrência.

3.3.3 *Random Forest* e árvores de decisão

Para entendermos o funcionamento do *Random Forest* precisamos primeiro de entender uma componente crucial dele, Árvores de decisão.

Árvores de decisão são modelos matemáticos usados para classificar ocorrências, semelhante aos outros modelos vistos. No entanto, estas usam, como o nome indica, uma estrutura de árvore, o que as torna em modelos muito fáceis de compreender e estudar, mesmo para conjuntos de dados complexos e grandes.

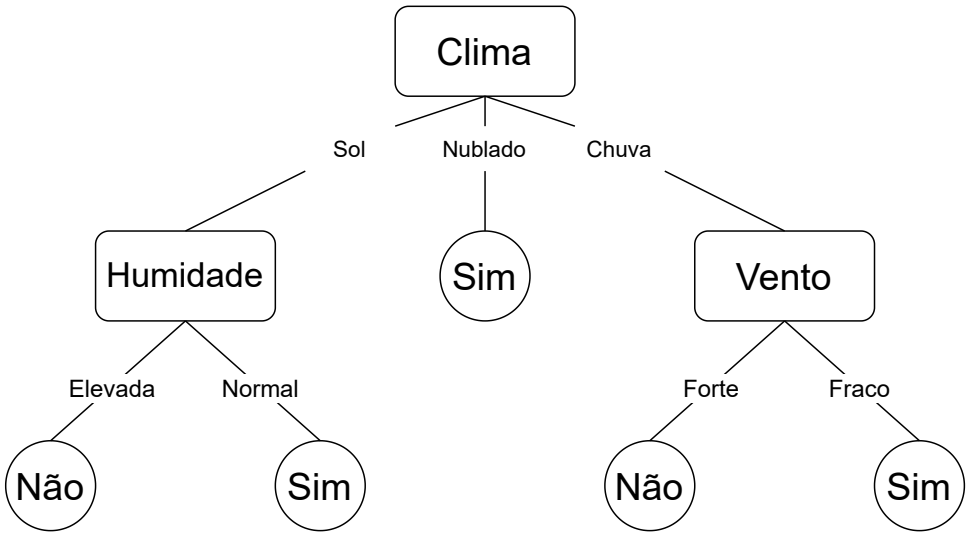

Figura 6: Exemplo de uma árvore de decisão

A Figura 6 mostra um exemplo de uma árvore de decisão que modela se num dado dia o utilizador irá ou não sair de casa para jogar *Badmington* (um exemplo clássico na área de classificação). Como podemos ver, o modelo é bastante simples, pois é o que intuitivamente a maioria das pessoas fazem no seu dia-a-dia. Isto pode ser generalizado para outros conjuntos de dados.

Random Forest

O algoritmo de *Random Forest* essencialmente usa várias árvores de decisão, onde cada árvore é treinada com diferentes partes do *dataset* escolhidas à sorte. Isto gera árvores diferentes, usando depois o *output* de todas as árvores, e escolhendo a classe maioritária para rotular a nossa nova observação.

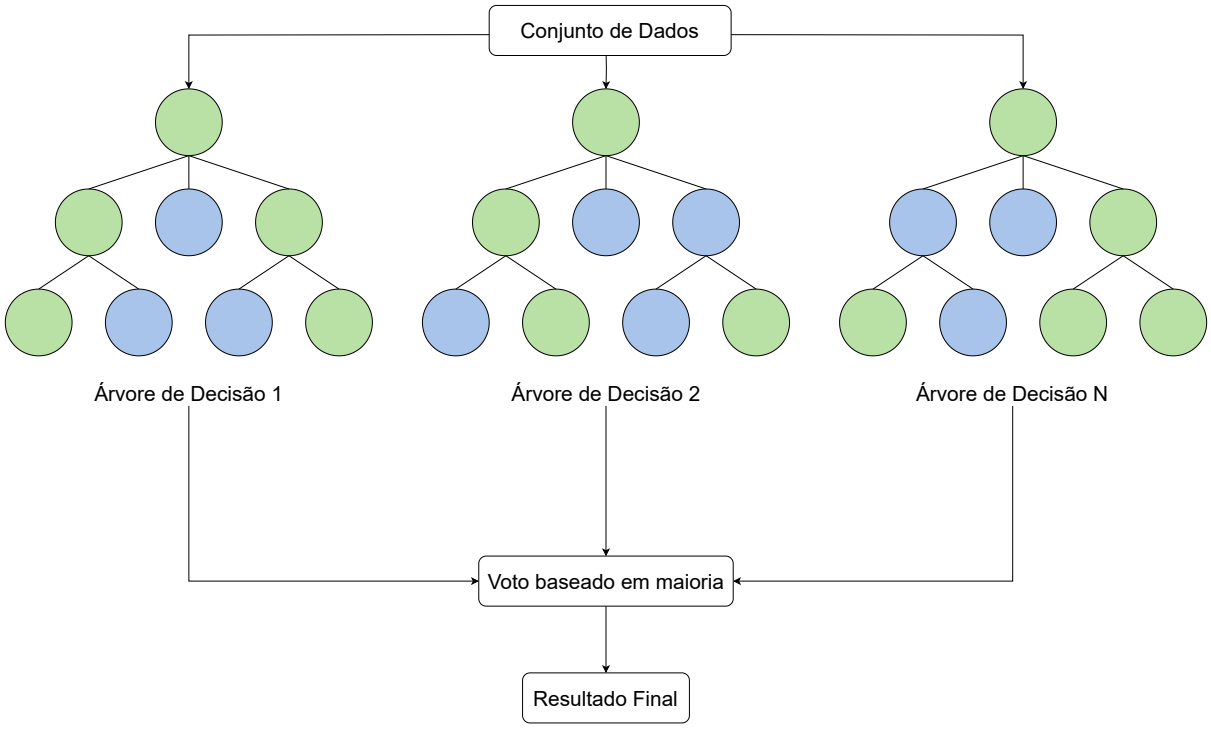

Figura 7: Exemplo de uma floresta gerada pelo *Random Forest*

3.3.4 *Support Vector Machines*

A ideia central de uma *SVM* é encontrar um hiperplano em N dimensões (onde N é o número de *features* do nosso conjunto de dados) que melhor divide as duas classes (é possível fazer com várias classes, mas essa abordagem é irrelevante para o nosso trabalho, pelo que não iremos falar delas). Para isto, maximiza-se a distância entre o hiperplano, e quaisquer dois pontos (chamada de **margem**).

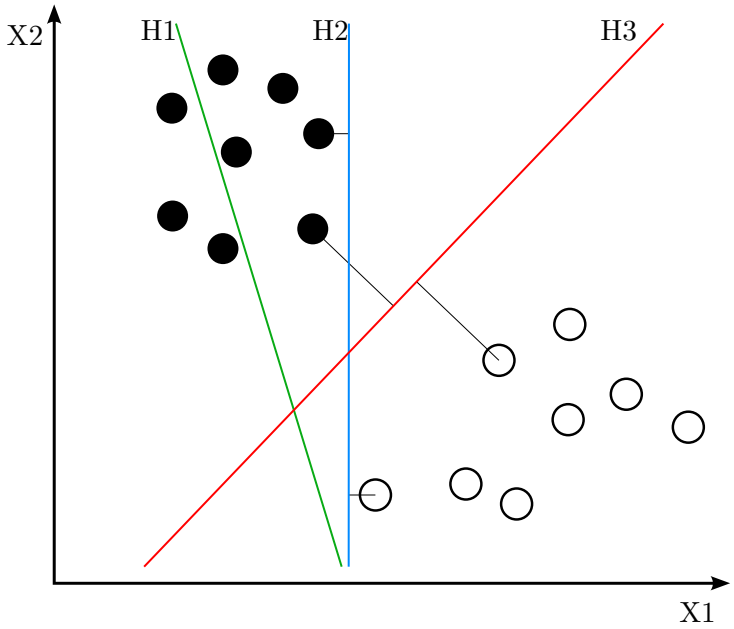

Figura 8: Exemplo de vários hiperplanos que separam as ocorrências

Como podemos ver, encontrar um hiperplano que separe as ocorrências todas é um problema computacionalmente difícil, em cima disso, ainda temos de ter a certeza que maximizamos a **margem**. Observando a Figura 8, o hiperplano *H1* estaria incorreto pois existem algumas observações da classe **preta** que seriam consideradas diferentes das outras. O hiperplano *H2*, apesar de separar corretamente as ocorrências todas, não maximiza a margem, o que poderia aumentar a chance de novas observações serem rotuladas de maneira errada. O hiperplano *H3* já corrige estes problemas todos, separando corretamente as classes, e ainda maximizando a **margem**, aumentando assim a probabilidade que novas ocorrências sejam corretamente identificadas.

3.3.5 *Recurring Neural Networks (RNN)* e *Long Short Term Memory (LSTM)*

As *Recurrent Neural Networks (RNN)* e a sua subcategoria, *Long Short Term Memory (LSTM)*, são algoritmos avançados de *deep learning* que compartilham muitas características com as redes neuronais convencionais, mas que incluem um elemento adicional vital: a capacidade de formar ciclos entre neurónios.

Esta diferença permite ao modelo reter uma espécie de "memória" de eventos anteriores, o que o torna especialmente potente para lidar com dados que têm uma sequência ou que exibem uma natureza temporal inerente aos dados [12]. Este aspeto das *RNNs* e *LSTMs* abre a possibilidade de explorar a dependência temporal nos dados, permitindo o reconhecimento de padrões sequenciais de eventos que ocorrem ao longo do tempo.

No contexto do nosso projeto, isto é particularmente útil. As nossas sequências de dados representam a utilização de um dado sistema operativo ao longo de um período de tempo. A estrutura sequencial inerente desses dados permite que um modelo baseado em *RNN* ou *LSTM* faça inferências valiosas que podem corresponder a certos padrões de comportamento que indicam um ataque.

Por exemplo, se o sistema registar várias tentativas de *login* mal sucedidas seguidas de uma tentativa bem sucedida, um algoritmo como o *LSTM* poderia interpretar isto como um ataque de força bruta. Este tipo de inferência temporal é uma força significativa das *RNNs* e *LSTMs* e destaca o seu potencial no campo da deteção de intrusões.

4 Resultados & Validação

Nesta secção, vamos apresentar e discutir os resultados obtidos pelos modelos após o treino com o nosso conjunto de dados processado. Exploraremos o desempenho de cada algoritmo em detalhe, analisando os critérios de avaliação selecionados.

4.1 Análise de Resultados

Durante este projeto foram apresentados vários algoritmos para deteção de intrusões, alguns baseados em frequências (*K-Nearest Neighbors*, *Random Forest*, *Naive Bayes*, *Support Vector Machines*), e *Long Short Term Memory* para deteção baseada em sequências. Vamos agora discutir a sua eficácia, assim como rapidez de deteção.

4.1.1 *K-Nearest Neighbors*

	W = 2 Precision Recall	W = 3 Precision Recall	W = 4 Precision Recall	W = 5 Precision Recall	W = 6 Precision Recall	W = 7 Precision Recall	W = 10 Precision Recall
Media Ponderada	0.63 0.65	0.69 0.73	0.76 0.79	0.81 0.83	0.84 0.85	0.86 0.87	0.89 0.90
Tempo de Treino (seg)	0.001	0.002	0.005	0.006	0.009	0.012	0.05
Tempo de Teste (seg)	0.0159	0.168	0.678	1.921	3.115	5.410	12.794

Tabela 3: *Precision* e *Recall* correspondente a cada teste na respetiva janela do algoritmo *KNN*

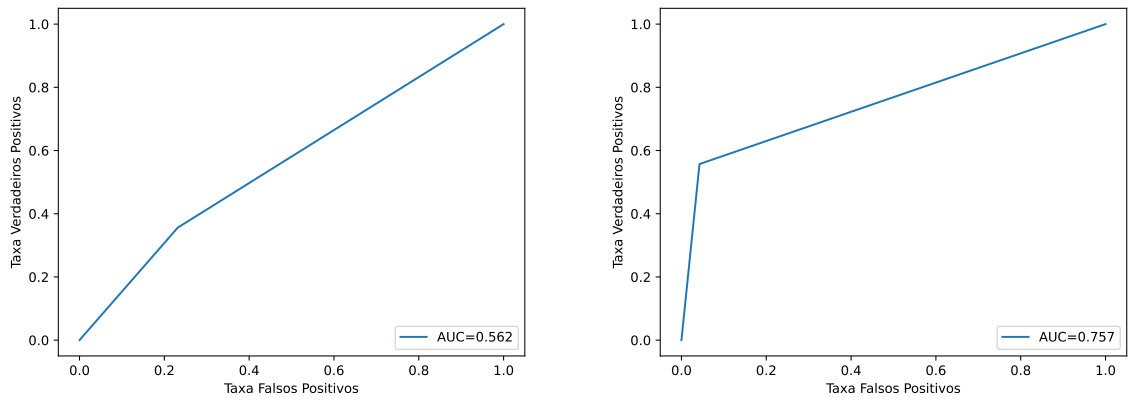

Figura 9: Curva de ROC para uma janela de tamanho 2 e 10, respetivamente, utilizando o algoritmo *K-Nearest Neighbors*

Como podemos ver, à medida que a janela aumenta, tanto a *Precision* como o *Recall* aumentam consideravelmente, chegando mesmo a rondar os 90% em ambos quando a janela tem um tamanho de 10. Isto indica que aumentar a janela ajuda consideravelmente, no entanto, a partir de janelas de tamanho 10, os dados começam a ter aumentos cada vez menores, até estagnarem,

por exemplo, com uma janela de tamanho 11, os resultados mantêm-se iguais para ambas as caraterísticas, só aumentando em 0.1 quando temos uma janela de tamanho 15, no entanto, como pode ser visto pela tabela anterior, o tempo de treino e teste demora consideravelmente mais consoante o tamanho da janela, pelo que decidimos não incluir estes resultados.

4.1.2 *Naive Bayes*

	W = 2 Precision Recall	W = 3 Precision Recall	W = 4 Precision Recall	W = 5 Precision Recall	W = 6 Precision Recall	W = 7 Precision Recall	W = 10 Precision Recall
Media Ponderada	0.80 0.57	0.82 0.39	0.83 0.34	0.86 0.32	0.86 0.30	0.87 0.28	0.87 0.27
Tempo de Treino (seg)	0.003	0.011	0.028	0.056	0.061	0.085	0.120
Tempo de Teste (seg)	0.012	0.014	0.006	0.018	0.023	0.032	0.043

Tabela 4: *Precision* e *Recall* correspondente a cada teste na respetiva Janela do Algoritmo *Naive Bayes*

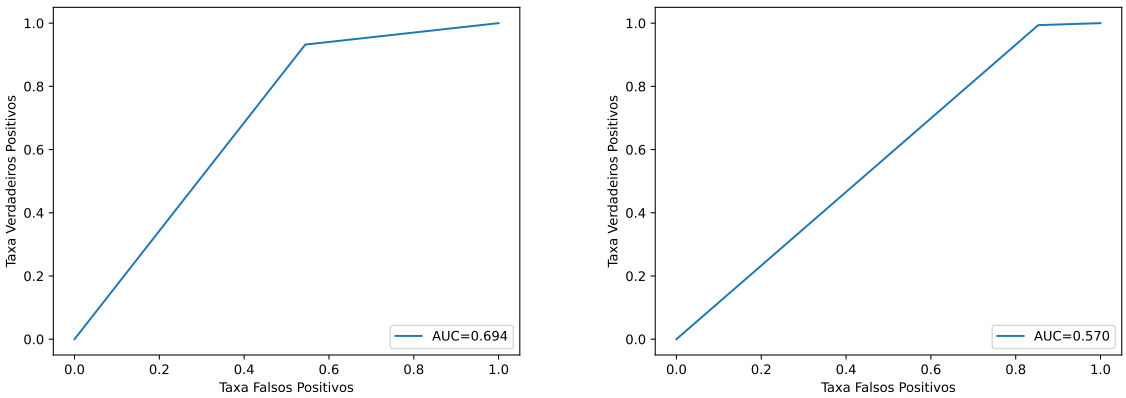

Figura 10: Curva de ROC para uma janela de tamanho 2 e 10, respetivamente, utilizando o algoritmo *Naive Bayes*

Podemos ver pela Figura 10 que este modelo não é muito adequado para o nosso *dataset*, pois o melhor valor que obtemos com ele é $AUC = 0.694$, o que é relativamente baixo para nos dar confiança nele. No entanto, este modelo é muito rápido tanto a treinar, como a detetar, devido à sua janela reduzida. Isto pode ser uma possível propriedade valiosa para aprofundar em futuros estudos.

4.1.3 *Random Forest*

	W = 2 Precision Recall	W = 3 Precision Recall	W = 4 Precision Recall	W = 5 Precision Recall	W = 6 Precision Recall	W = 7 Precision Recall	W = 10 Precision Recall
Media Ponderada	0.70 0.72	0.74 0.77	0.80 0.82	0.83 0.85	0.85 0.87	0.88 0.89	0.90 0.91
Tempo de Treino (seg)	0.172	0.469	1.018	1.803	2.688	3.653	6.001
Tempo de Teste (seg)	0.011	0.026	0.054	0.083	0.119	0.161	0.267

Tabela 5: *Precision* e *Recall* correspondente a cada teste na respetiva janela do algoritmo *Random Forest*

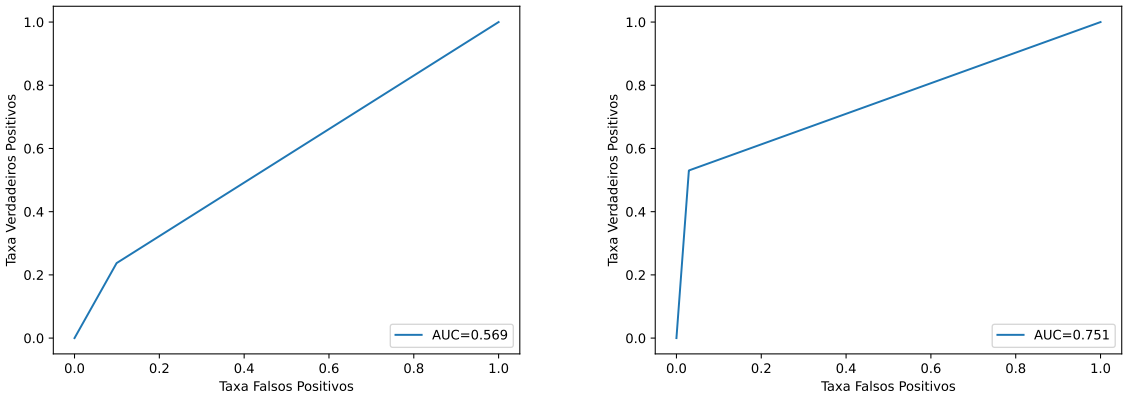

Figura 11: Curva de *ROC* para uma janela de tamanho 2 e 10, respetivamente, utilizando o algoritmo *Random Forest*

Como podemos ver pela tabela 5 e pela figura 11, o *Random Forest* obteve resultados consideravelmente melhores que o *Naive Bayes*, assemelhando-se ao *K-Nearest Neighbors* em termos de *AUC*. No entanto, é uma algoritmo mais demorado a treinar, mas mais rápido na fase de testes, o que poderá ser desejável no nosso caso.

4.1.4 *Support Vector Machines*

	W = 2		W = 3		W = 4		W = 5		W = 6		W = 7		W = 10	
	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
Media Ponderada	0.49	0.70	0.59	0.77	0.64	0.80	0.82	0.83	0.84	0.85	0.85	0.86	0.88	0.88
Tempo de Treino (seg)	0.039		0.398		2.897		8.317		17.854		28.176		62.249	
Tempo de Teste (seg)	0.031		0.293		1.147		2.911		6.366		10.734		25.448	

Tabela 6: *Precision* e *Recall* correspondente a cada teste na respetiva janela do algoritmo *SVM*

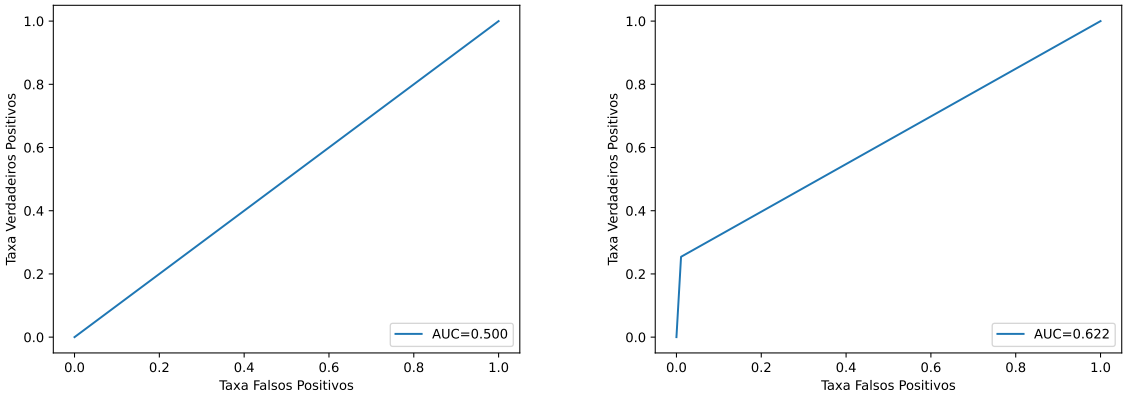

Figura 12: Curva de *ROC* para uma janela de tamanho 2 e 10, respetivamente, utilizando o algoritmo *Support Vector Machines*

Podemos ver pela tabela 6 e pela figura 12 que o algoritmo *SVM* não é muito indicado para este problema, pois, no seu pior, é como escolher uma classe completamente à sorte, e no seu melhor, tem um valor de *AUC* a rondar os 62%. Este modelo também é o que tem tempos de deteção piores, tendo quase o dobro do valor do segundo pior, *KNN*.

4.1.5 *Long Short Term Memory*

Para o nosso uso, treinamos o modelo usando 30 *epochs* (quantas vezes passamos pelo *dataset* de treino), e obtemos os seguintes resultados:

	W = 2		W = 3		W = 4		W = 5		W = 6		W = 7		W = 10	
	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
Media Ponderada	0.67	0.68	0.73	0.76	0.80	0.82	0.81	0.82	0.82	0.83	0.81	0.83	0.79	0.80
Tempo de Treino (seg)	16.448		81.218		219.074		408.457		609.939		807.457		1290.488	
Tempo de Teste (seg)	0.078		0.116		0.202		0.353		0.435		0.550		0.882	

Tabela 7: *Precision* e *Recall* correspondente a cada teste na respetiva janela do algoritmo *LSTM*

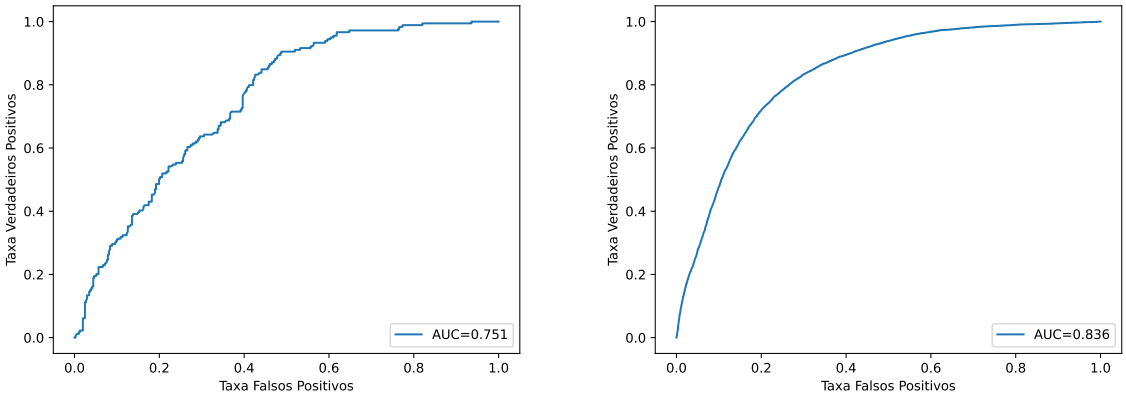

Figura 13: Curva de *ROC* para uma janela de tamanho 2 e 10, respetivamente, utilizando o algoritmo *Long Short Term Memory*

Avaliando os resultados previamente apresentados, é evidente que o algoritmo *LSTM* se destacou entre os outros modelos, demonstrando um desempenho superior. Este modelo atingiu um *AUC* de 83%, consideravelmente melhor em comparação com qualquer outro algoritmo explorado no âmbito deste projeto, mesmo o melhor modelo, baseado em sequências, que foi o *K-Nearest Neighbors*.

4.2 Comparação de Resultados

Tamanho da Janela	*K-Nearest Neighbors*	*Naive Bayes*	*Random Forest*	*Support Vector Machines*	*Long Short Term Memory*
2	56.2%	69.4%	56.9%	50.0%	75.1%
10	75.7%	57.0%	**75.1%**	62.2%	**83.6%**

Tabela 8: Comparação do valor de *AUC* para todos os modelos para tamanho de janela 2 e 10

Para o propósito de deteção de intrusões, é crucial que estas sejam detetadas o mais cedo possível para que outras medidas de proteção possam ser iniciadas, ou seja, o tempo que o modelo demora a detetar uma intrusão é quase tão importante como a sua eficácia. Por esta razão, seria possivelmente melhor usar *Random Forest* em vez do *K-Nearest Neighbors* visto que, como podemos ver na Tabela 8, a eficácia de ambos é similar, no entanto, o tempo de deteção do *KNN* era consideravelmente maior.

Os outros algoritmos, à exceção do *Long Short Term Memory*, tiveram uma eficácia consideravelmente pior, independentemente do tamanho da janela, pelo que não parecem promissores para a avaliação de resultados, portanto, iremos agora comparar o melhor algoritmo baseado em frequências, *Random Forest*, com o melhor algoritmo baseado em sequências, *Long Short Term Memory*.

	Random Forest $W = 10$	*Long Short Term Memory* $W = 10$
Tempo de Deteção (seg)	0.267	0.882
AUC	0.751	0.836

Tabela 9: *Tempo de Deteção* e AUC correspondente a uma janela de tamanho 10 para ambos *Random Forest* e *Long Short Term Memory*

Ainda que o modelo *Long Short Term Memory* tenha demonstrado um desempenho superior em termos de resultados de deteção, é importante notar que este vem acompanhado de um tempo de deteção marginalmente superior. Deste modo, poderia ser vantajoso adotar uma abordagem combinada que tire proveito das forças de ambos os modelos. Por exemplo, poderíamos utilizar inicialmente o modelo *Random Forest* para uma deteção rápida, embora menos precisa. Quando este modelo sinaliza uma possível intrusão, o *LSTM* entraria em ação para fornecer uma análise mais aprofundada e precisa, funcionando como uma forma de confirmação. Esta abordagem combinada poderia potencialmente compensar as limitações inerentes a cada modelo individual, resultando numa solução de deteção de intrusões mais robusta e eficiente.

5 Conclusão

Com o término deste trabalho, concluímos que a implementação de um Sistema de Deteção de Intrusões (*IDS*) utilizando *Machine Learning* oferece uma abordagem poderosa para a deteção de potenciais ataques, inclusive aqueles previamente desconhecidos. A abrangência dos modelos investigados, desde o *K-Nearest Neighbors (KNN)*, *Naive Bayes*, *Random Forest*, *Support Vector Machines (SVM)* até o *Long Short Term Memory (LSTM)*, permitiu uma avaliação detalhada das diferentes estratégias de modelagem possíveis.

Os modelos *Random Forest* e *LSTM* apresentaram desempenho superior aos demais, cada um com suas peculiaridades. O *Random Forest* mostrou-se eficiente em termos de rapidez na deteção de ataques, enquanto o *LSTM*, embora mais lento, apresentou maior precisão e uma melhor capacidade de lidar com séries temporais, destacando a importância das informações contextuais na deteção de intrusões.

Ainda que ambos os modelos tenham se destacado, a sugestão de um modelo híbrido, que aproveite a deteção rápida do *Random Forest* e a precisão superior do *LSTM*, pode ser uma alternativa atrativa. A combinação desses modelos poderia permitir uma primeira deteção rápida, seguida de uma análise mais detalhada para confirmar a presença de um ataque.

Este trabalho representa uma contribuição para a área de segurança, abordando a necessidade crítica de desenvolver sistemas de deteção de intrusões robustos e eficientes. Além disso, fornece uma análise aprofundada das potencialidades e limitações dos diferentes modelos de *Machine Learning* na deteção de intrusões, contribuindo para um melhor entendimento de como a tecnologia pode ser aplicada neste campo.

O caminho percorrido até aqui reforça a importância da evolução contínua na área de segurança cibernética. Conforme os ataques se tornam cada vez mais sofisticados, também devem evoluir as nossas estratégias de deteção e defesa. Nesse sentido, esperamos que as descobertas apresentadas neste projeto possam contribuir para futuras pesquisas, desenvolvimentos e aprimoramentos no campo da deteção de intrusões.

5.1 Trabalho Futuro

Para trabalhos futuros, há várias direções promissoras a serem exploradas com base no que foi descrito até agora. O objetivo é continuar a evolução constante das técnicas de deteção de intrusões para que possamos acompanhar e prever ameaças cada vez mais complexas.

Otimização de *LSTM*: Embora o modelo *LSTM* tenha demonstrado desempenho superior, está ainda longe de perfeito. Seria interessante tentar explorar métodos para similares aumentar a eficácia, por exemplo, modificação da arquitetura da rede, ajustando parâmetros, e incorporando técnicas de regularização [13], com o objetivo de otimizar ainda mais o desempenho do modelo.

Proteção híbrida usando vários modelos: Esta investigação revelou que diferentes modelos oferecem diferentes vantagens: o *Random Forest* destacou-se na deteção rápida, enquanto que o *LSTM* mostrou alta precisão. Um trabalho futuro valioso seria examinar como esses dois modelos, e possivelmente outros, poderiam ser combinados em um sistema híbrido ou uma abordagem em duas etapas para aproveitar os pontos fortes de cada modelo.

Atualização para novos tipos de ataques: Com a área de cibersegurança em constante mudança, novos tipos de ataques estão a ser desenvolvidos regularmente. Uma direção importante para o trabalho futuro seria a atualização e adaptação contínuas dos modelos para identificar e lidar com esses novos tipos de ataques.

Exploração de Outros Algoritmos de *Machine Learning*: Existem muitos outros algoritmos de *Machine Learning* que não foram explorados neste estudo e que podem ser úteis na deteção de intrusões. Seria valioso explorar o desempenho dessas outras abordagens em trabalhos futuros.

Refinamento da Classificação de Tipo de Ataque: No estado atual, o sistema é capaz de detetar quando um ataque está a ocorrer, mas não avaliamos a sua eficácia quando tentamos detetar o ataque em si. Seria interessante investigar a utilização destes modelos na deteção e reconhecimento dos ataques em especifico, pois poderia permitir que diferentes medidas preventivas sejam tomadas.

Em conclusão, há um vasto leque de possibilidades para avanços futuros no campo dos sistemas de deteção de intrusões. A área de segurança continua a ser um campo de grande importância, e é crucial que continuemos a aprimorar e expandir estas ferramentas e técnicas para lidar com as variadas ameaças. Com a continuação da pesquisa nessa direção, podemos esperar que os *IDS* se tornem ainda mais robustos e precisos, protegendo os nossos sistemas de uma gama cada vez maior de ameaças.

Bibliografia

[1] "Guide to Intrusion Detection and Prevention Systems (IDPS)". In: (2007).

[2] ITU. "Internet more affordable and widespread, but world's poorest still shut off from online opportunities". In: (2022).

[3] "Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey". In: (2019).

[4] SolarWinds. *Security Event Manager*. URL: `https://www.solarwinds.com/security-event-manager` (visited on 06/20/2023).

[5] Splunk. *Splunk*. URL: `https://www.splunk.com/` (visited on 06/20/2023).

[6] Ossec. *Ossec*. URL: `https://www.ossec.net/` (visited on 06/20/2023).

[7] Gideon Creech. "The ADFA Intrusion Detection Datasets". In: (2013).

[8] "Methods for Host-based Intrusion Detection with Deep Learning". In: (2021).

[9] "Computer security threat monitoring and surveillance." In: (1980).

[10] Tiffany M Powell-Wiley et al. "Obesity and cardiovascular disease: a scientific statement from the American Heart Association". In: *Circulation* 143.21 (2021), e984–e1010.

[11] "The Optimality of Naive Bayes". In: (2004).

[12] Tara N Sainath et al. "Convolutional, long short-term memory, fully connected deep neural networks". In: *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. Ieee. 2015, pp. 4580–4584.

[13] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. "Regularizing and optimizing LSTM language models". In: *arXiv preprint arXiv:1708.02182* (2017).