ENGENHARIAS, TECNOLOGIA, GESTÃO E TURISMO
ENGINEERING, TECHNOLOGY, MANAGEMENT AND TOURISM
INGENIERÍA, TECNOLOGÍA, ADMINISTRACIÓN Y TURISMO

millenium

Millenium, 2(Edição Especial Nº20)

pt

MODELOS DE INTELIGÊNCIA ARTIFICIAL PARA ANÁLISE DE EVENTOS EM LOGS
ARTIFICIAL INTELLIGENCE MODELS FOR LOG EVENT ANALYSIS
MODELOS DE INTELIGENCIA ARTIFICIAL PARA EL ANÁLISIS DE EVENTOS DE LOGS

Paulo Castro[1] *https://orcid.org/0009-0007-0010-1783*
Fernando Santos[1] *https://orcid.org/0000-0003-1551-4111*
Pedro Lopes[1] *https://orcid.org/0000-0002-4644-5748*

[1] Instituto Politécnico de Viseu, Viseu, Portugal

Paulo Castro – estgl4262@estgl.ipv.pt | Fernando Santos - fsantos@estgl.ipv.pt | Pedro Lopes - plopes@estgl.ipv.pt

Autor Correspondente:
Paulo Castro
Rua Alexandre Herculano
5100-107– Lamego, Portugal
pauloftcastro@gmail.com

RECEBIDO: 07 de maio de 2025
REVISTO: 29 de julho de 2025
ACEITE: 17 de setembro de 2025
PUBLICADO: 16 de outubro de 2025

DOI: https://doi.org/10.29352/mill0220e.41569

RESUMO

Introdução: A análise de *logs* é considerada uma tarefa fundamental no âmbito da cibersegurança, dada a sua relevância na identificação de padrões e comportamentos anómalos e atividades potencialmente maliciosas em redes computacionais, auxiliando numa resposta preventiva e fundamentada.
Objetivo: Examinar e comparar modelos de Inteligência Artificial com potencial para a deteção de anomalias em eventos de *logs*, identificando os mais adequados ao contexto de cibersegurança de uma instituição de ensino superior.
Métodos: Foi conduzida uma revisão sistemática da literatura científica com análise comparativa, de modelos supervisionados e não supervisionados de *Machine Leaning* e *Deep Learning*. A análise considerou critérios como sensibilidade a padrões anômalos, exigências de recursos computacionais, entre outros critérios relevantes.
Resultados: A revisão permitiu identificar diversas abordagens com características e níveis distintos de aplicabilidade. As informações reunidas oferecem uma base útil para orientar em decisões futuras quanto à adoção destas soluções, considerando os desafios associados à análise de grandes volumes de dados.
Conclusão: O estudo fornece uma base sólida para orientar a seleção inicial de modelos de Inteligência Artificial para a análise de *logs* em cibersegurança. A próxima etapa da investigação consistirá na transição para a implementação prática destes modelos, avaliando o seu desempenho em ambiente operacional. Este processo permitirá validar as escolhas teóricas efetuadas e otimizar a sua aplicabilidade.

Palavras-chave: inteligência artificial; deteção de anomalias; logs; cibersegurança**;** machine learning

ABSTRACT

Introduction: In cybersecurity, log analysis plays a crucial role by identifying patterns, anomalies, and potentially malicious activities in computer networks, supporting proactive and informed responses.
Objective: To explore and compare the different models of Artificial Intelligence built around detecting anomalies in log events, mainly prioritizing their use in an institution's network.
Methods: This work is characterized as a systematic literature review with a comparative analysis. The analysis was done following a literature review, extended through supervised and unsupervised models of Machine Learning and Deep Learning, to consider several contingencies as their sensitivity to anomaly patterns or use of computational resources.
Results: The review depicted variability within models in their characteristics and applications, highlighting their versatility. This systematic analysis provides a baseline knowledge to guide decision makers in the future regarding obstacles in the analysis of substantial amounts of data.
Conclusion: This research establishes a solid basis for the initial selection of Artificial Intelligence models for log analysis in cybersecurity. The next phase of the investigation will involve the practical implementation of these models, evaluating their performance in an operational environment. This process will allow for the validation of the theoretical choices made and the optimization of their applicability.

Keywords: artificial intelligence; anomaly detection; logs; cybersecurity; machine learning

RESUMEN

Introducción: El análisis de *logs* es una tarea fundamental en ciberseguridad, dado su papel en la identificación de patrones anómalos y actividades maliciosas en redes informáticas, facilitando respuestas preventivas y fundamentadas.
Objetivo: Este artículo tiene como objetivo examinar y comparar diversos modelos de Inteligencia Artificial aplicados a la detección de anomalías en eventos de logs, con énfasis en su uso en la red informática de una institución de educación superior.
Métodos: Se llevó a cabo una revisión sistemática de la literatura científica con análisis comparativos, abarcando modelos supervisados y no supervisados de Machine Learning y Deep Learning. El análisis consideró criterios como sensibilidad a patrones anómalos, requisitos computacionales, entre otros aspectos relevantes.
Resultados: La revisión permitió identificar distintos enfoques con niveles variables de aplicabilidad. La información recopilada sirve como base para orientar decisiones futuras sobre la adopción de estas soluciones.
Conclusión: Este estúdio ofrece una base sólida para orientar la selección inicial de modelos de Inteligencia Artificial aplicados al análisis de *logs* en ciberseguridad. La siguiente fase consistirá en implementar estos modelos en un encuentro operativo para validar las elecciones teóricas y optimizar su aplicabilidad.

Palabras clave: inteligência artificial; detección de anomalias; logs; ciberseguridad; machine learning

DOI: https://doi.org/10.29352/mill0220e.41569

INTRODUÇÃO

No domínio da computação e da cibersegurança, os logs, ou registos de eventos, constituem uma fonte de dados primordial para a compreensão do comportamento de sistemas e aplicações. Fundamentalmente, os *logs* são registos cronológicos de eventos, gerados automaticamente por uma variedade de fontes digitais, incluindo sistemas operativos, aplicações de *software*, dispositivos de rede e infraestruturas digitais em geral. Estes registos podem ser compostos por uma única linha de texto ou por múltiplas linhas, capturando detalhes de interações e estados ao longo do tempo (Landauer et al., 2023).

A prática de documentar ocorrências, de forma detalhada, precede significativamente o surgimento do primeiro computador, encontrando-se presente ao longo da história da humanidade. Na navegação marítima, os capitães e navegadores mantinham registos detalhados sobre rotas, condições meteorológicas, entre outros acontecimentos a bordo. No setor da saúde, hospitais e médicos efetuavam registos sobre pacientes, sintomas e tratamentos.

Com o início da era dos computadores, por volta das décadas de 1950 e 1960, originaram-se paralelamente os *logs* digitais, que inicialmente eram projetados para manutenção de *hardware*. Em sistemas de computação modernos, estes registos digitais tornaram-se uma fonte rica e indispensável para diversas análises.

As fontes de eventos, como podemos verificar na Figura 1, normalmente armazenam as atividades de forma sequencial, ordenadas por tempo (Giradin & Brodbeck, 2002). Assim sendo, estes eventos são registos de uma interação realizada durante a execução de uma operação do sistema. Os registos podem fornecer dados tais como:

- Data e Hora – O instante preciso em que o evento ocorreu;
- Origem – A entidade (sistema, aplicação, serviço) que gerou o registo;
- Nível de Severidade – Classificação da criticidade do evento (“Informação”, “Aviso”, “Erro”, “Fatal”);
- Endereços de IP – Identificadores de rede ou dispositivos envolvidos em comunicações;
- Mensagens de Texto – Descrições pormenorizadas do evento, que podem variar de notificações concisas a detalhados rastreios de erros;
- Outros parâmetros que se mostrem relevantes, podem ser adicionados ao registo.

Para garantir o processamento consistente destes registos, os eventos são manipulados através de uma expressão regular (*regex*), que contém uma estrutura padrão, utilizada na identificação e extração de informações específicas. Este formato garante que o sistema consiga processar *logs* de maneira consistente. A mensagem em destaque inclui todo o conteúdo que possa estar inserido num registo (Astekin et al., 2018).

Type	Date	Time	Source	Category	Event	User	Computer
Information	1/21/2010	10:34:15 …	ESENT	General	100	N/A	VIDYAVASU
Error	1/18/2010	10:34:20 …	crypt32	None	8	N/A	VIDYAVASU
Error	1/18/2010	10:34:20 …	crypt32	None	8	N/A	VIDYAVASU
Error	1/18/2010	10:34:23 …	crypt32	None	8	N/A	VIDYAVASU
Error	1/18/2010	10:34:25 …	crypt32	None	8	N/A	VIDYAVASU
Error	1/18/2010	10:34:28 …	crypt32	None	8	N/A	VIDYAVASU
Error	1/18/2010	10:34:30 …	crypt32	None	8	N/A	VIDYAVASU
Information	1/18/2010	10:34:56 …	EAPOL	None	2002	N/A	VIDYAVASU
Information	1/18/2010	10:34:56 …	EAPOL	None	2003	N/A	VIDYAVASU
Information	1/18/2010	10:35:59 …	iPod Service	None	0	N/A	VIDYAVASU
Error	1/18/2010	10:36:24 …	crypt32	None	8	N/A	VIDYAVASU
Warning	1/15/2010	10:36:30 …	crypt32	None	6	N/A	VIDYAVASU
Error	1/15/2010	10:36:30 …	crypt32	None	8	N/A	VIDYAVASU
Information	1/18/2010	10:36:39 …	DesktopCentral	None	103	N/A	VIDYAVASU
Error	1/18/2010	10:36:39 …	crypt32	None	8	N/A	VIDYAVASU
Error	1/18/2010	10:37:31 …	crypt32	None	8	N/A	VIDYAVASU
Information	12/24/2009	10:38:15 …	ESENT	General	101	N/A	VIDYAVASU

Figura 1 – Fontes de Eventos

Fontes de eventos que fornecem descrições estruturadas, possuem entradas distintas e uniformes que descrevem cada evento para um determinado conjunto de características. No entanto, os *logs* não estruturados, não necessitam de entradas de eventos uniformes (Giradin & Brodbeck, 2002).

A cibersegurança é um assunto que ganhou uma enorme relevância ao longo dos anos. Com o aumento exponencial de invasões e ataques, consideram-se os *logs* como uma ferramenta fundamental na deteção e análise de possíveis ameaças. Cada vez mais, hackers exploram as falhas de segurança, comprometendo assim a autenticidade, a integridade, a confidencialidade e a disponibilidade de um sistema (Aung & Min, 2017). Assim sendo, a recolha de eventos de *logs* de entrada da rede e das demais aplicações determinam a existência de violações (Podlodowski & Kozlowski, 2019).

DOI: https://doi.org/10.29352/mill0220e.41569

Devido à natureza automatizada na produção de *logs* nos sistemas de computação, tornou-se impraticável efetuar uma inspeção manualmente. Para além de ser extremamente trabalhoso, revela-se inoportuno para responder a incidentes em tempo real (Yen & Moh, 2019). Deste modo, cria-se a necessidade de implementar técnicas de deteção de anomalias capazes de aprender automaticamente os modelos que representam o comportamento normal do sistema. Posteriormente, estas técnicas identificam e alertam atividades possivelmente adversas, que requerem a atenção de operadores humanos (Landauer et al., 2023).

1 MÉTODOS

Este estudo caracteriza-se como uma investigação de natureza qualitativa, conduzida sob a forma de revisão sistemática da literatura, com enfoque comparativo. A recolha de informação foi realizada em bases de dados científicas amplamente reconhecidas na área, nomeadamente IEEE Xplore, ACM Digital Library, ScienceDirect e SpringerLink, contemplando publicações de diversos períodos.

Foram incluídos trabalhos que descrevem e analisam modelos de *Machine Learning* (ML) ou *Deep Learning* (DL), privilegiando-se aqueles cujo conteúdo pudesse contribuir para a compreensão e avaliação das abordagens no contexto deste estudo.

Após a seleção inicial, cada artigo foi analisado segundo critérios como o tipo de modelo, a abordagem (supervisionada ou não supervisionada), a sensibilidade a padrões anómalos, a robustez face a ruído, os requisitos computacionais, a capacidade de generalização, o suporte a múltiplas classes e a adequação a diferentes tipos de dados. As características identificadas foram sintetizadas em tabelas, recorrendo a escalas qualitativas para facilitar a comparação direta entre abordagens.

O objetivo desta análise metodológica foi identificar, de forma fundamentada, os modelos com maior potencial para implementação prática no contexto da cibersegurança, considerando especificamente a deteção de anomalias em eventos de *logs*.

Neste enquadramento, os modelos de Inteligência Artificial (IA) possibilitam a aprendizagem de padrões representativos do comportamento normal de um sistema, permitindo a identificação de desvios que possam traduzir-se a falhas, comportamentos anómalos ou potenciais ameaças à integridade das infraestruturas. A aplicação de técnicas de IA contribuem para uma resposta rápida, precisa e informada a incidentes (Du et al., 2017).

1.1 Algoritmos de Machine Learning Supervisionado

Os algoritmos de ML são métodos computacionais que permitem que os sistemas aprendam padrões, a partir de dados e façam previsões ou tomadas de decisão, sem serem explicitamente programados para isso (Belcic & Stryker, n.d.). O *Machine Learning Supervisionado* é uma técnica que se divide em dois tipos de tarefas, sendo elas, classificação e regressão. A regressão associa-se à previsão de um valor contínuo, ou seja, o modelo tenta prever uma saída numérica através de dados de entrada ainda não observados. Em contraste, a tarefa de classificação envolve a previsão de valores categóricos, atribuindo a cada entrada uma classe específica entre um conjunto de categorias possíveis (El Mrabet et al., 2021).

1.1.1 Random Forest

O *Random Forest* é um método de *ensemble* que desempenha previsões a partir da agregação dos resultados de múltiplas árvores de decisão (Resende & Drummond, 2018). Este método é considerado particularmente adequado para situações com um número elevado de características (*features*), sendo capaz de lidar com grandes volumes de dados e manter uma previsão precisa, mesmo na presença de ruído e dados ausentes. A eficácia deste algoritmo reflete-se na sua estrutura constituída por diversos classificadores fracos, cuja combinação resulta num classificador forte. Este tipo de abordagem permite atenuar o problema de *overfitting,* algo comum em árvores de decisão individuais, aumentando a capacidade de generalização do modelo.

No entanto, num caso hipotético em que, num conjunto de características contenha apenas uma pequena porção de informação relevante, o desempenho do *Random Forest* pode ser afetado negativamente. A introdução de atributos irrelevantes tende a influenciar a construção das árvores, levando-as a ajustar-se ao ruído presente nos dados, o que pode comprometer todo o desempenho (Abellán et al., 2017; (Amaratunga et al., 2008; Aung & Min, 2017). Ainda assim, o algoritmo minimiza este risco por meio da técnica *bagging*, que será discutida adiante.

A construção de uma única árvore, dentro da floresta, inicia-se com a seleção de uma característica x e um limite t que dividem um conjunto de treino X, composto por N amostras pertencentes a duas classes e descritas por G características, em dois subconjuntos X_L e X_R. Esta divisão visa maximizar um critério de separação, normalmente medido pelo índice de *Gini* ou pela *Entropia* (Amaratunga et al., 2008).

$$\text{Gini}(S) = 1 - \sum_{i=1}^{C} p_i^2$$

$$\text{Entropia}(S) = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

DOI: https://doi.org/10.29352/mill0220e.41569

Onde p_i representa a proporção de elementos da classe i no conjunto S. C é o número total de classes. Este procedimento é iterativamente repetido para cada subconjunto, utilizando outras combinações (x, t), até que não seja possível efetuar mais nenhuma divisão.
Ao contrário da *Decision Tree,* o *Random Forest* utiliza a técnica de *bagging*, na qual cada árvore é treinada com uma amostra aleatória, com reposição de N instâncias. Estas amostras constituem o conjunto denominado *in-bag*, enquanto as instâncias não selecionadas durante o processo são encaminhadas para um conjunto *out-of-bag*, que eventualmente poderá ser utilizado na estimativa de erro do modelo sem a necessidade de uma validação cruzada.

$$\mathrm{Err}_{\mathrm{OOB}} = \frac{1}{N}\sum_{i=1}^{N} I\left(\hat{y}_{\mathrm{OOB}}^{(\mathrm{i})} \neq y^{(i)}\right)$$

Onde $y^{(i)}$ representa o rótulo verdadeiro da amostra i, $\hat{y}_{\mathrm{OOB}}^{(\mathrm{i})}$ é a previsão para a amostra i, feita apenas pelas árvores que não utilizaram essa amostra durante o treino e $I(\cdot)$ é a função indicadora, que retorna 1 se a previsão for incorreta e 0 se estiver correta.
Adicionalmente, na tentativa de reduzir a correlação entre árvores e aumentar a diversidade *ensemble*, considera-se, em cada nó, apenas um subconjunto aleatório de g características $G = \sqrt{G}$ é considerado para determinar a melhor divisão. Esta aleatoriedade adicional contribui para a consistência do algoritmo, ao melhorar a precisão em problemas complexos e de alta dimensionalidade.
A predição final para uma nova instância é obtida por votação maioritária entre as árvores, de acordo com a seguinte expressão (Amaratunga et al., 2008):

$$\hat{y} = \mathrm{mode}\{h_1(x), h_2(x), \ldots, h_R(x)\}$$

Onde $h_1(x)$ é a previsão de i-ésima árvore para a instância x, e R é o número total de árvores da floresta.
Constata-se que o *Random Forest* destaca-se por diversas vantagens, nomeadamente a sua robustez, a sua eficiência na gestão de conjunto de dados com elevada dimensionalidade, entre outras. Contudo, importa salientar que o algoritmo apresenta limitações. A aleatoriedade introduzida na seleção de atributos, embora indispensável na redução de correlação entre árvores, pode em certos casos resultar numa dependência excessiva das regras de construção, aumentando o risco de *overfitting* (Feng et al., 2020). Além disso, o desempenho do algoritmo tende a ser afetado quando aplicado a conjuntos de dados desequilibrados. Por fim, tanto o treino quanto a fase de inferência podem tornar-se computacionalmente exigentes, especialmente quando se utiliza um número alto de árvores, o que representa um desafio em ambientes com limitações de tempo e recursos computacionais.

1.1.2 Extreme Gradiente Boosting (XGBoost)

O *Extreme Gradiente Boosting* (XGBoost) é um algoritmo baseado em Árvores de decisão, sendo amplamente utilizado em tarefas de classificação e regressão (Himalaya Gohiya et al., 2018). O seu desenvolvimento foi focado no desempenho computacional, escalabilidade e capacidade de generalização (Abdiyeva-Aliyeva et al., 2022). Trata-se de uma técnica de *ensemble learning* baseada no algoritmo de *gradient boosting*, ou seja, permite envolver diversos modelos de aprendizagem que, em conjunto, melhoram o desempenho de cada um individualmente (Abdiyeva-Aliyeva et al., 2022) (Marinho & http://lattes.cnpq.br/5348870751897882, 2021). No decorrer deste processo, n modelos são treinados consecutivamente, sendo que cada novo modelo considera os erros cometidos pelo anterior. Deste modo, o processo consiste em minimizar a função de perda $L\big(y, F(x)\big)$, onde y caracteriza o valor real e $F(x)$ a previsão do modelo (Abdiyeva-Aliyeva et al., 2022).
Inicialmente, o modelo base realiza uma predição constante, e de seguida são calculados os erros de predição, indicando a diferença entres valores reais e os valores preditos. Através da otimização da função de perda é efetuado o processo de ajuste baseado no *gradient*, ao utilizar o algoritmo *gradient descent*. Uma nova árvore é construída em cada etapa, com o objetivo de prever os erros do modelo anterior, consoante a seguinte expressão:

$$\mathrm{F_m(x)} = \mathrm{F_{m-1}(x)} + \alpha_{\mathrm{m}} \cdot \mathrm{h_m(x,\, r_{m-1})}$$

Onde $F_m(x)$ representa o modelo atualizado na interação m, h_m é o modelo base treinado sobre os erros rm-1, e α_m e r_m são os coeficientes de regularização e de erro que minimizam a função de perda $L\big(y, F(x)\big)$. O coeficiente α_m é determinado através da seguinte equação:

$$\arg\alpha\min\sum_{i=1}^{m} L\big(y_i, F_{i-1}(x_i) + \alpha h_i(x_i, r_{i-1})\big)$$

DOI: https://doi.org/10.29352/mill0220e.41569

Contrariamente ao *bagging*, que constrói as árvores de maneira independente, o *boosting* reduz o viés do modelo, ou seja, ao aprender com os erros cometidos leva a um menor risco de *underfitting* e consequentemente a uma maior flexibilidade.
A cada divisão binária (nó) de atributos, é calculado o *Information Gain* que tem como função medir a qualidade da divisão.

$$IG = \text{similarity}_{\text{ramo1}} + \text{similarity}_{\text{ramo2}} - \text{similarity}_{\text{raiz}}$$

Com base nos erros obtidos, o *score* de similaridade de cada ramo é dado por:

$$\frac{\sum(y - \hat{y})^2}{P(1-P) + \lambda}$$

Em que P caracteriza a probabilidade prevista pelo modelo anterior e λ o parâmetro de regularização, utilizado para evitar o *overfitting*.
Para tarefas de classificação, o XGBoost utiliza funções de ativação como por exemplo a função *sigmoide* (Abdiyeva-Aliyeva et al., 2022).

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

De uma forma geral, a implementação do *gradient boosting* no XGBoost segue um fluxo lógico. O algoritmo começa por calcular a previsão média em problemas de regressão ou probabilidade constante no caso de tratar de um problema de classificação. De seguida, calcula os erros e constrói uma nova árvore a partir desses mesmos erros, combina as previsões com uma taxa de aprendizagem α e, por fim, repete este processo indefinidamente até atingir um número de árvores satisfatório.
Para além das vantagens mencionadas anteriormente, este algoritmo oferece uma ampla adaptabilidade na configuração de hiperparâmetros, possibilitando ajustar o modelo conforme as características específicas dentro do contexto do problema. Parâmetros como *max_depth*, *min_child_weight*, *gamma*, permitem controlar a complexidade da árvore e induzir a aleatoriedade durante o treino, favorecendo a generalização e a eficiência do modelo. No entanto, o XGBoost apresenta determinadas limitações, como a sua elevada complexidade computacional, tanto no treino quanto na fase de inferência, principalmente quando é utilizado diversas árvores ou quando o algoritmo é aplicado num volumoso conjunto de dados. A necessidade de um ajuste criterioso dos hiperparâmetros também pode representar uma barreira, já que o desempenho do modelo pode eventualmente variar consoante as configurações necessárias (Marinho & http://lattes.cnpq.br/5348870751897882, 2021).

1.1.3 Support Vector Machine (SVM)

O *Support Vector Machine* (SVM) é utilizado tanto para problemas de classificação como de regressão (Somvanshi et al., 2017). É amplamente usado em tarefas de classificação de padrões, que podem ser lineares ou não lineares (Pradhan, 2012). Em padrões linearmente separáveis, as amostras são facilmente divididas e, portanto, o algoritmo constrói um limite de decisão, designado por hiperplano. No caso dos padrões não lineares, os dados não são facilmente separáveis, o que leva o SVM a usar funções *kernel* (Somvanshi et al., 2017) (Pradhan, 2012). Estas funções mapeiam o espaço original num novo espaço para que os dados que originalmente não eram separáveis, possam ser separados sem calcular explicitamente todas as novas dimensões (Liu & Lang, 2019).
Na seleção do hiperplano, é indispensável resolver o problema da margem máxima. Esta margem define a distância entre o limite de decisão e os vetores de suporte (Somvanshi et al., 2017). Quanto maior a margem, o melhor será o processo de classificação. Assim sendo, os cálculos são realizados através da seguinte expressão.

$$aX + bY = C$$

Em padrões não linearmente separáveis, como já foi referido anteriormente, é necessário mapear os dados, para um novo espaço de maiores dimensões, através de uma função *kernel*.

$$\mathrm{K(x,\ y)} = \Phi(\mathrm{x}) \cdot \Phi(\mathrm{y})$$

Outro parâmetro de extrema importância é o parâmetro de complexidade C, que se refere à soma das distâncias de todos os pontos que se encontram do lado errado do hiper plano (Pradhan, 2012). Este valor não deverá ser muito elevado, uma vez que a sua tolerância ao erro será praticamente nula, podendo levar a *overfitting*. O valor C também não deverá ser muito pequeno, visto que o torna demasiado tolerante a erros, podendo assim levar a *underfitting*. Assim sendo, o valor de C não deverá ser nem muito grande nem muito pequeno, pois poderá influenciar negativamente o desempenho da classificação.

DOI: https://doi.org/10.29352/mill0220e.41569

De uma forma geral, o princípio do SVM num conjunto de amostras $\{(x_i, y_i)\}_{i=1}^{N}$, onde cada vetor $x_i \in R^d$ corresponde a uma amostra com d características, e cada classe $y_i \in \{-1, 1\}$ indica a classe a que a amostra corresponder e encontrar um hiperplano de separação que divida corretamente estas amostras consoante as suas classes. O hiperplano define-se pela seguinte equação:

$$w^T x + b = 0$$

Onde w representa o vetor normal ao hiper plano e b caracteriza o viés. Assim, a otimização do problema fica reduzido a um problema quadrático, em que o seu objetivo é maximizar a margem.

$$\min \Phi\,(w) = \frac{1}{2}|w|^2 = \frac{1}{2}(w, w)$$

Em que $|w|^2$ é o quadrado da norma do vetor w, que se responsabiliza por medir a largura da margem. O fator ½ é apenas uma convenção matemática, facilitando a derivada no processo de otimização, porém não altera a solução final.
Para garantir que os pontos sejam separados corretamente pelas classes, é indispensável impor uma restrição que assegure que todos os pontos de um lado do hiper plano pertençam à sua respetiva classe. A restrição é expressa pela seguinte equação:

$$\mathrm{y_i}(\mathrm{w} \cdot \mathrm{xi} + \mathrm{b}) \geq 1$$

Onde y_i é a classe da amostra i. Para uma amostra da classe +1 esteja do lado correto do hiper plano, o valor da equação deverá ser igual ou superior a 1. No caso de uma amostra da classe -1, o valor de equação deverá ser igual ou inferior a -1.
O *Support Vector Machine* é capaz de generalizar um problema, sendo o teor da aprendizagem estatística. A teoria da aprendizagem estatística é fornecer um enquadramento para estudar o problema, fazendo previsões e tomadas de decisões através de um conjunto de dados. Deste modo, a aprendizagem supervisionada é formulada da seguinte maneira:
Dado um conjunto de treino $\{(x_1, y_1), \ldots, (x_n, y_n)\} \subset R^n \times R$, onde x_1 e y_1 são pares de entrada e de saída, extraídos de uma distribuição de probabilidade desconhecida $P(x, y)$ e uma função de perda $V\big(y, f(x)\big)$ que mede o erro, o objetivo passa por encontrar uma função f que diminua a probabilidade de erros nos novos dados.

$$\min \int V\big(y, f(x)\big)\; P(x, y)\, dx\, dy$$

Uma das maiores vantagens do SVM reside na sua notável capacidade de generalização. Ao procurar maximizar a margem entre as classes, o modelo tende a reduzir significativamente a probabilidade de *overfitting*, sendo particularmente eficaz em conjuntos de dados com alto número de características e poucas amostras (Somvanshi et al., 2017). Porém, esta eficácia tem como contrapartida um elevado custo computacional durante o processo de treino. Em conjuntos de dados de grandes dimensões, o processo de otimização quadrática pode tornar-se extremamente exigente em termos de recursos computacionais (Schölkopf & Smola, 2002). Além disso, o facto do *Support Vector Machine* ter sido concebido para classificação binária, todo o tipo de problema que envolva mais de duas classes necessita de estratégias adicionais, como *one-vs-all*, onde treina-se um classificador para cada classe, tratando esta classe como positiva e todas as outras como negativas, ou *one-vs-one*, onde é treinado um classificador para cada par de classes possíveis e a predição é efetuada com base no voto maioritário. Estas abordagens, apesar de serem competentes, aumentam a complexidade computacional e o tempo de execução à medida que o número de classes cresce (Somvanshi et al., 2017).
Por fim, o modelo gerado por um SVM não é facilmente interpretável, sobretudo quando se utiliza um *kernel* não linear. Esta falta de interoperabilidade pode ser um obstáculo quando é exigido uma justificação das decisões do modelo (Schölkopf & Smola, 2002).

1.2 Algoritmos de Machine Learning Não Supervisionado

O *Machine Learning* Não Supervisionado tem como principal objetivo identificar grupos ou padrões no conjunto sem dispor de classificações de classe previamente definidos durante a fase de treino. Esta abordagem visa descobrir relações ou estruturas ocultas nos dados, de modo que certos padrões emergentes ocorram com maior frequência do que outros. Esta técnica divide-se em duas tarefas fundamentais, sendo elas, *clustering* e redução de dimensionalidade. A tarefa de *clustering* tem como objetivo agrupar os dados baseando-se nas suas semelhanças, organizando-os num número definido de grupos. Por sua vez, a redução de dimensionalidade, busca simplificar os dados, convertendo-os de um espaço com muitas variáveis para um espaço de menores dimensões, preservando, ao máximo, as características mais importantes da distribuição original. Esta transformação é útil para compressão de informação, remover atributos redundantes e melhorar o desempenho computacional (El Mrabet et al., 2021).

DOI: https://doi.org/10.29352/mill0220e.41569

1.2.1 Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

O algoritmo *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN), como o próprio nome indica, utiliza uma abordagem baseada em densidade, que permite identificar e agrupar conjuntos de dados com diferentes densidades em formatos esféricos (P. Singh & Meshram, 2018). No contexto deste algoritmo, o agrupamento é o processo de formação de *clusters* baseados nas regiões com alta e baixa densidade (Kulkarni & Burhanpurwala, 2024; H. V. Singh et al., 2022) . Por outras palavras, o agrupamento permite dividir conjuntos de dados em grupos variados com a similaridade *intracluster* maximizada e minimizada (H. V. Singh et al., 2022).

A técnica de agrupamento é um dos principais métodos de *data mining*. É um processo que busca padrões relevantes e anteriormente desconhecidos, porém potencialmente úteis, a partir de conjuntos de dados espaciais de grandes dimensões (Parilama et al., 2011). Deste modo, o funcionamento geral pode ser dividido em duas etapas. A primeira etapa procura um procedimento que realize uma estimativa de densidade de um ponto e identifique objetos que se encontrem em regiões densas. A segunda etapa é o processo de identificar grupos de observações que são atingíveis através de algumas observações principais. Na impossibilidade de adicionar mais pontos dentro de um *cluster*, estes pontos restantes são designados como *outliers* ou ruído (Kulkarni & Burhanpurwala, 2024).

A construção dos agrupamentos leva a cabo dois parâmetros fundamentais. Os Eps (épsilon) caracterizam o raio em torno de um ponto do qual considera-se os pontos vizinhos, determinados a densidade da região (H. V. Singh et al., 2022). O $MinPts$ refere-se à menor contagem de pontos necessária dentro do agrupamento, formando assim um *cluster*.

A vizinhança Eps de um ponto p, indicado por $N_{\mathrm{Eps}}(p)$ é definida como o conjunto de pontos $q \in D$, tal que a distância entre p e q é menor ou igual a Eps. Por outras palavras, trata-se da localização dos pontos a uma distância máxima de Eps do ponto p. Com base nesta definição, verifica-se num cluster, os pontos designados por pontos centrais (*core points*) que se situam inseridos dentro do *cluster* e os pontos de fronteira (*border points*) localizam-se nas extremidades do agrupamento.

Um ponto p é diretamente acessível por densidade através de um ponto q, relacionado aos parâmetros Eps e $MinPts$, apenas se ambas as condicionantes forem simultaneamente satisfeitas:

- $p \in N_{\mathrm{Eps}}(q) - p$ apresenta-se na vizinhança de q;
- $\left|N_{\mathrm{Eps}}(q)\right| \geq MinPts - q$ é um ponto central apenas se tiver uma densidade de pontos satisfatória.

Além disso, um ponto p é atingível por densidade através de um ponto q se existir uma cadeia de pontos $p_1, p_2, \ldots, p_n$, com $p_1 = q$ e $p_n = q$, tal que cada ponto $p_i + 1$ seja diretamente acessível por densidade a partir de p_i. Esta definição possibilita que pontos mais distantes, ainda conectados dentro de regiões densas, sejam considerados pertencentes ao mesmo agrupamento.

Dois pontos p e q são considerados conectados por densidade se existir um ponto intermediário o tal que p e q sejam atingíveis por densidade através do ponto o. Esta característica é fundamental para definir agrupamentos coerentes, mesmo quando os pontos p e q não estão diretamente acessíveis entre si.

Apoiado nestas definições, um agrupamento C, em relação a Eps e $MinPts$, é um subconjunto não vazio da base de dados D que satisfaz as seguintes propriedades:

- Maximalidade – Se $p \in C$ e q é acessível por densidade através de p, concluímos que $q \in C$;
- Conectividade – Todos os pares de pontos $p, q \in C$ estão conectados por densidade.

Os *outliers* são descritos por um conjunto de pontos não pertencentes a qualquer agrupamento identificado. De outro modo, todos os pontos $p \in D$ para os quais não existe nenhum agrupamento C_i tal que $p \in C_i$ [24].

Após concluir todo o fluxo lógico integrado no algoritmo DBSCAN para qualquer conjunto de dados, é obtido geralmente 3 tipos de pontos (H. V. Singh et al., 2022):

- Núcleo – na formação de diversos grupos, o núcleo é o ponto a partir do qual há pelo menos k pontos numa distância r do raio;
- Borda – Qualquer ponto que contenha um ou mais pontos centrais dentro de uma distância r do raio;
- Ruído – Pode ser qualquer ponto exceto um núcleo ou uma borda e que obrigatoriamente tenha pelo menos k número de pontos numa r distância do raio.

Deste modo, verificamos que este algoritmo tem um excelente desempenho quando se trata de ruído, tendo em conta a sua habilidade em ignorar pontos não pertencentes a qualquer região densa (H. V. Singh et al., 2022). Assim sendo, auxilia na extração de *insights* relevantes em conjuntos de dados complexos. No entanto, o DBSCAN é inadequado quando se trata de agrupar conjuntos volumosos de dados, devido ao elevado tempo de computação. O algoritmo também não possui a capacidade de identificar *clusters* caracterizados por diferentes densidades e limites parcialmente interligados (Kulkarni & Burhanpurwala, 2024).

DOI: https://doi.org/10.29352/mill0220e.41569

1.2.2 K-Means

O *K-Means* é um algoritmo, onde k refere-se ao número de *clusters* e *means* é a média dos atributos. É geralmente utilizado para agrupar grandes conjuntos de dados, semelhante ao que ocorre no DBSCAN, principalmente quando se trata de *data mining* e reconhecimento de padrões (Na et al., 2010) (Li & Wu, 2012). Além disso, a sua eficácia computacional e rápida convergência torna-o claramente adequado para aplicações de larga escala. Em contraste, demonstra limitações ao tratar todas as *features* de um conjunto de dados de igual forma, sem considerar os níveis de relevância (de Amorim, 2016). A sua dependência do valor de k, leva a que a escolha inadequada deste parâmetro possa comprometer a qualidade dos agrupamentos. Ademais, o *K-Means* revela sensibilidade na presença de *outliers*, que podem distorcer substancialmente a posição dos centroides e, consequentemente, afetar a formação de *clusters* (Ahmed et al., 2020).

O principal objetivo desta ferramenta é minimizar o erro quadrático, ao procurar k *clusters* que melhor retratam os dados. Deste modo, o algoritmo busca minimizar o índice de desempenho de acordo com a seguinte função:

$$J = \sum_{j=1}^{c} \sum_{X \in S_j} |X - M_j|^2$$

Onde J caracteriza a soma dos erros quadráticos, S_j retrata os pontos no *cluster j, e* M_j é o vetor médio, também conhecido como centroide. De acordo com a função anterior, o algoritmo pretende minimizar a distância entre os pontos de dados e seus respetivos centroides, resultando em *clusters* bem definidos.

Para cada classe S_j, o vetor médio M_j é calculado através da expressão:

$$M_j = \frac{1}{N_j} \sum_{X \in S_j} X$$

Onde N_j é o número de amostras no cluster S_j (Li & Wu, 2012).

Geralmente, quando falamos no processo de agrupamento, há três etapas envolvidas. Inicialmente, são escolhidos k pontos como centroides iniciais, que basicamente são o "centro" de um grupo de pontos de dados. De seguida, atribui-se ao centroide mais próximo cada ponto restante com base na distância euclidiana. Assim sendo, é formada uma classificação inicial de dados, onde a hipótese desta classificação não ser aceitável, os centroides são novamente calculados como a média dos pontos atribuídos a cada *cluster*. Estas etapas são repetidas até que os centroides permaneçam no mesmo local (Zhao et al., 2018).

1.2.3 Isolation Forest

O *Isolation Forest* é um algoritmo baseado em árvores, popularmente conhecido devido à sua eficácia na deteção de anomalias. Considera-se anomalias como padrões que não se identificam a um modelo bem definido de padrões normais. Assim sendo, a palavra *Isolation* refere-se ao processo de isolamento que este método realiza na separação de padrões incomuns dentro de um conjunto de dados (Chabchoub et al., 2022).

Ao contrário dos métodos referidos anteriormente, este algoritmo não calcula distâncias nem densidade e, portanto, o tempo de processamento e a exigência de memória são consideravelmente reduzidos (Chabchoub et al., 2022).

No processo de construção, o modelo cria várias árvores designadas por *Isolation Trees* (*iTrees*), ou em português, Árvores Isoladas. A construção de cada *iTree* é efetuada de forma recursiva, ou seja, em cada passo o algoritmo tende a escolher aleatoriamente uma *feature* e um valor de corte. Assim, divide os dados em dois subconjuntos, como ocorre numa árvore binária. O processo de divisão é concluído apenas quando uma amostra é isolada, ou seja, quando se torna o único elemento presente num subconjunto ou quando a árvore atinge a profundidade máxima previamente estabelecida. Desta forma, o modelo compreende que não será possível efetuar mais repartições e, consequentemente não haverá mais ganhos ao proceder em mais ramificações. É importante referir ainda que amostras que sejam isoladas nos primeiros níveis de ramificação, o percentual de serem classificadas como anômalas é elevada (Al Farizi et al., 2021).

No entanto, o *Isolation Forest* apresenta dificuldades em lidar com anomalias que apenas são detetáveis quando é considerado a combinação de múltiplas variáveis. Esta situação ocorre quando o algoritmo realiza divisões unidimensionais e, portanto, dificulta o processo de isolar pontos anômalos envolvidos por dados normais em diversas dimensões. Outra limitação importante referir, denomina-se de região fantasma. O algoritmo tende a atribuir pontuações artificialmente baixas a áreas onde não existe quaisquer dados reais ou padrões idênticos aos das regiões normais. Devido a uma limitação estrutural, esta abordagem executa apenas cortes paralelos aos eixos dos dados. Portanto, impede interações não lineares entre variáveis e, por conseguinte, há uma maior probabilidade de gerar falsos negativos (H. Xu et al., 2023).

Com base nos princípios descritos, o *Isolation Forest* fundamenta-se na premissa de que o isolamento de amostras anómalas é uma estratégia eficaz. De acordo com a árvore binária, cada árvore de isolamento é constituída por nós que desempenham funções distintas no processo de isolamento. Se T for um nó interno, este realiza um teste de divisão e possui dois nós filhos,

DOI: https://doi.org/10.29352/mill0220e.41569

nomeadamente a subárvores esquerda T_l e a subárvore direita T_r. Se T for um nó externo (folha), retrata o ponto em que o isolamento de uma amostra termina.
Para construir uma *iTree*, primeiramente é selecionado aleatoriamente um atributo A e um valor de divisão p, extraídos do conjunto de dados $D = \{d_1, d_2, \dots, d_n\}$. De seguida, cada amostra d_u é inserida à subárvore T_l no caso de $d_u(A) < p$, ou na subárvore de direita se $d_u(A) > p$. Este processo termina quando uma das condições referidas previamente for satisfeita.
A métrica principal utilizada é o comprimento de caminho $h_{(d)}$, que caracteriza o total de arestas percorridas desde a raiz até à folha onde o ponto d é isolado. O cálculo desta métrica é dado pela seguinte equação:

$$c(n) = 2H(n-1) - \big((2(n-1)/n)\big)$$

Onde $c(n)$ é a média de $h(d)$, $H(i)$ é o número harmónico e n retrata o número de folhas. A pontuação de anomalia s de uma instância d é dada pôr:

$$s(d,n) = 2 - \Big(E\big(h(d)/c(n)\big)\Big)$$

Onde $E\big(h(d)\big) \rightarrow$ caracteriza a média do comprimento do caminho para $h(d)$ numa coleção de árvores de isolamento.
Interpretamos $s(d,n)$ da seguinte maneira:

- Quando $E\big(h(d)\big) \rightarrow 0$, então $s \rightarrow 1$ – a amostra foi isolada rapidamente e, portanto, tem uma forte probabilidade de ser considerada uma anomalia;
- Quando $E\big(h(d)\big) \rightarrow c(n)$, então $s \rightarrow 0{,}5$ – a amostra segue o padrão de isolamento das restantes, indicando um comportamento normal;
- Quando $E\big(h(d)\big) \rightarrow n-1$, então $s \rightarrow 0{,}5$ – a amostra exigiu um percurso longo para ser isolado, revelando fortes indicações de normalidade (D. Xu et al., 2018).

1.3 Algoritmos de Deep Learning Supervisionado
O *Deep Learning* é um subcampo do *Machine Learning*, que simula o modo como o cérebro humano processa informações e adquire conhecimentos. Este paradigma baseia-se em modelos compostos por diversas camadas de representação, em que cada camada utiliza os dados processados pela camada antecedente com o objetivo de contruir abstrações cada vez mais complexas. Os modelos DL Supervisionados dependem de dados estruturados e categorizados para produzir resultados precisos (Holdsworth & Scapicchio, n.d.; Masolo, 2017).

1.3.1 LSTM
Long Short-Term Memory (LSTM) é um tipo de *Recurrent Neural Network* (RNN) que pretende superar o problema do *vanishing gradient*, que normalmente ocorre durante o processo de aprendizagem de dependências de longo prazo em sequências temporais (Van Houdt et al., 2020).
A arquitetura LSTM é composta por um conjunto de sub-redes com ligações recorrentes, conhecidas como *memory blocks* (Graves, 2012). Cada bloco é constituído por uma ou várias *memory cells* com ligações próprias e três unidades de *gates* multiplicativas, sendo elas, *input gate*, *output gate* e *forget gate* (Graves, 2012) (Chung & Shin, 2018). A *forget gate* inicialmente não estava presente na arquitetura deste algoritmo, tendo sido posteriormente introduzida para permitir que a rede aprendesse a "esquecer" informações irrelevantes armazenadas nas *memory cells* (Van Houdt et al., 2020). A *cell* tem a responsabilidade de transmitir valores de estado em intervalos de tempo variáveis e as três *gates* controlam o fluxo de informações através de operações, tais como, gravação, leitura e reinicialização (Chung & Shin, 2018; Van Houdt et al., 2020).
O processo de computação inicia-se pela atualização de *block input*, combinando a entrada atual $x^{(t)}$ e a saída $y^{(t-1)}$ na última iteração, através da seguinte expressão:

$$z^{(t)} = g\big(W_z x^{(t)} + R_z y^{(t-1)} + b_z\big)$$

Onde W_z e R_z são os pesos associados a $x^{(t)}$ e $y^{(t-1)}$, respetivamente, enquanto b_z caracteriza o vetor de peso de polarização. Esta mesma estrutura aplica-se às demais *gates*, substituindo os subscritos conforme seja necessário.
De seguida, a *input gate* unifica a entrada atual $x^{(t)}$, a saída $y^{(t-1)}$ e o valor da célula $c^{(t-1)}$ no passo de tempo anterior, mediante a seguinte fórmula:

$$i^{(t)} = \sigma\big(W_i x^{(t)} + R_i y^{(t-1)} + p_i \odot c^{(t-1)} + b_i\big)$$

DOI: https://doi.org/10.29352/mill0220e.41569

Onde ⊙ representa a multiplicação pontual de dois vetores.
Nas etapas anteriores, a camada LSTM decide quais informações devem ser mantidas nos estados das células da rede $c^{(t)}$. Isto envolve a seleção dos valores candidatos $z^{(t)}$ que podem ser adicionados aos estados das células, bem como a definição dos valores de ativação $i^{(t)}$ das *inputs gates*.
A *forget gate*, estabelece quais informações devem ser removidas dos estados anteriores de célula $c^{(t-1)}$. Para isso, os valores de ativação $f^{(t)}$ da *forget gate* no passo de tempo t, são calculados com base na entrada atual $x^{(t)}$, nas saídas $y^{(t-1)}$ e no estado $c^{(t-1)}$ das *memory cells* no passo de tempo anterior $(t-1)$, considerando também as conexões de *peephole* e os termos de polarização b_f da *forget gate*. É calculado a partir da seguinte equação:

$$f^{(t)} = \sigma\left(W_f x^{(t)} + R_f y^{(t-1)} + p_f \odot c^{(t-1)} + b_f\right)$$

De seguida, calcula-se o valor da *cell*, combinado com o *block input* $z^{(t)}$, a *input gate* $i^{(t)}$ e o *forget gate* $f^{(t)}$, juntamente com o valor da célula do passo tempo anterior. Calculado através da expressão:

$$c^{(t)} = z^{(t)} \odot i^{(t)} + c^{(t-1)} \odot f^{(t)}$$

Na sequência, o cálculo da *indoor gate*, unifica a entrada atual $x^{(t)}$, a saída $y^{(t-1)}$ e o valor da célula $c^{(t-1)}$ na última instância, conforme a seguinte função:

$$o^{(t)} = \sigma\left(W_o x^{(t)} + R_o y^{(t-1)} + p_o \odot c^{(t)} + b_o\right)$$

Por fim, a saída de bloco calcula o *block output* que combina o valor atual da célula $c^{(t)}$ com o valor da *output gate* atual da seguinte forma:

$$y^{(t)} = g\left(c^{(t)}\right) \odot o^{(t)}$$

No entanto, a escassa interoperabilidade deve-se à arquitetura complexa baseada em diversas camadas e operações não lineares do LSTM, funcionado como uma "caixa negra". Portanto, dificulta a compreensão e a explicação das decisões tomadas pelo modelo, comprometendo a transparência de resultados. Além disso, tal como acontece noutros tipos de redes neuronais, exige vários parâmetros de controlo, tais como, a quantidade de neurónios por camadas, o número de passos temporais, entre outros (Chung & Shin, 2018).
O *Gated Recurrent Unit* (GRU) segue uma arquitetura muito semelhante à do LSTM. Por sua vez, o GRU simplifica a estrutura ao fundir as *inputs gates* e *forget gates* numa única porta de atualização, e ao unificar o estado da célula com o estado oculto. Comparativamente com o LSTM, esta algoritmo exige menos números de parâmetros, tornando-o mais leve computacionalmente e destacando-se pela sua velocidade de treino e simplicidade (Cho et al., 2014; Mello, 2021).
Porém, perante o LSTM, o *Gated Recurrent Unit* apresenta uma capacidade de memória mais limitada por exigir menos parâmetros, o que poderá dificultar na captura de dependências de longo prazo em sequências complexas. O facto de o estado da célula estar fundido com o estado oculto, poderá ser desvantajoso no caso de ser necessário reter informações relevantes ao longo de sequências mais extensas (Cho et al., 2014; Hochreiter & Schmidhuber, 1997).

1.3.2 Bidirectional Encoder Representations from Transformers (BERT)

O *Bidirectional Encoder Representations from Transformers* (BERT), é um algoritmo *opensource*, baseado nos *Transformers*, para processamento de linguagem natural (PLN) (Hashemi-Pour & Lutkevich, n.d.). O princípio de funcionamento deste modelo consiste na fase de pré-treino e *fine tuning*, nas quais aprende representações de texto a partir de grandes volumes de dados textuais não supervisionados, podendo atingir um excelente desempenho relativamente a tarefas de PLN (Wang et al., 2024).
O BERT utiliza um *two-way Transformer*, ou seja, um transformador bidirecional como codificador, permitindo considerar simultaneamente a informação contextual. Desta forma, a relação entre palavras é compreendida facilmente, em relação a outros modelos que apenas utilizam codificadores unidirecionais. Para a aprendizagem de representação textual no pré-treino, este algoritmo utiliza duas tarefas, sendo elas, *Masked Language Model* (MLM) e *Next Sentence Prediction* (NSP). A tarefa MLM tem como objetivo forçar o modelo na captura de informação contextual e, assim, capturar representações contextuais mais informativas das palavras. O processo desta tarefa leva a que uma percentagem das palavras permaneça em estado oculto, levando o modelo a ser treinado na previsão das palavras ocultas, maximizando a função de verossimilhança, contribuindo na capacidade de entender os vários significados de diversas palavras em diferentes contextos (Wang et al., 2024; Zhang et al., 2021).
Na tarefa NSP, os dados são divididos em duas partes iguais. Numa metade dos pares de frases são contextualmente consecutivos e na outra metade não. Portanto, esta tarefa leva a que o algoritmo pré-treinado tenha a capacidade de identificar os pares de

DOI: https://doi.org/10.29352/mill0220e.41569

frases consecutivos (Zhang et al., 2021). Uma vez treinado, o BERT poderá ser utilizado em diversas tarefas de processamento de linguagem natural, como é o caso de análise de *logs* (Wang et al., 2024).
Na fase *fine-tuning*, ao adicionar apenas uma camada de saída ao modelo treinado previamente e combinar com dados categorizados para a aprendizagem supervisionada, com o intuito de realizar uma tarefa específica.
Dado que este método aprende a linguística comum, o número de parâmetros necessário a serem ajustados é bastante reduzido, refletindo num excelente desempenho, numa maior flexibilidade e versatilidade (Wang et al., 2024).

1.4 Algoritmos de Deep Learning Não Supervisionado
Os modelos de *Deep Learning* Não Supervisionado são capazes de identificar automaticamente padrões, atributos e relações relevantes através de dados não estruturados, sem a necessidade de intervenção humana direta. Além disso, estes sistemas possuem a capacidade de avaliar e melhorar continuamente as suas próprias saídas, promovendo o seu nível de precisão ao longo do tempo (Holdsworth & Scapicchio, n.d.).

1.4.1 LogBERT
O *Log Bidirectional Encoder Representations from Transformers*, é um algoritmo focado na deteção de anomalias em dados de *logs*, inspirado na arquitetura BERT e adaptado ao domínio de sistemas computacionais. Trata os *logs* como sequências de eventos, ao aprender automaticamente padrões sequenciais e dependências contextuais complexas. Através da adaptação do poder de representação semântica do BERT, o LogBERT simula o comportamento normal do sistema com base em grandes volumes de *logs* não categorizados. Durante o período de treino, qualquer desvio dos padrões aprendidos pode ser identificado como anomalia.
A arquitetura de LogBERT é constituída por três estágios, designadamente, o pré-processamento, a codificação contextual com camadas *Transformer* e o pré-treino a partir de duas tarefas principais, sendo elas, *Masked Log Key Prediction* (MLKP) e *Volume of Hypersphere Minimization* (VHM).
Cada evento e_i é convertido numa chave de *log*, mais concretamente designado pelo processo de Tokenização, e associado a um vetor de *embedding* $x_i = Embedding(e_i)$, formando uma sequência $S = [e_1, e_2, \dots, e_n]$. Estes *embeddings* são processados por camadas *encoder* do *Transformer*, dando resultado em representações contextuais $H = [h_1, h_2, \dots, h_n]$.
O MLKP, inspirado pelo MLM do algoritmo BERT, esta tarefa consiste em mascarar aleatoriamente algumas posições da sequência com um *token* especial (*MASK*) e prever a chave de *log* original. A previsão para cada posição mascarada $i \in M$ é dada por:

$$\hat{e}_i = arg \max_{e \in V} P(e|S_{\text{mask}})$$

Com a função de perda associada:

$$L_{\text{MLKP}} = -\sum_{i \in M} \log P(e_i|S_{\text{mask}})$$

O VHM atua como regularizador, ao comprimir representações normais dentro de uma hiperesfera de raio R e com centro c. A perda VHM é dada pela seguinte expressão:

$$L_{\text{VHM}} = R^2 + \frac{1}{n}\sum_{i=1}^{n} max(0, |z_i - c|^2 - R^2)$$

No período de treino, o modelo é otimizado com a combinação ponderada de ambas as perdas:

$$L_{\text{total}} = \lambda_1 L_{\text{MLKP}} + \lambda_2 L_{\text{VHM}}$$

Na fase de treino, o LogBert pode identificar anomalias com base no critério MLKP, no caso de a chave real e_i não estiver entre as k mais prováveis, a sequencia é considerada anómala:

$$Anomalia_{\text{MLKP}}(e_i) = \begin{cases} 1, se\ e_i \notin Top - k\big(P(\cdot\,|S_{\text{mask}})\big) \\ 0, caso\ contrário \end{cases}$$

Um outro critério a ter em conta, é a distância latente à hiperesfera. Se a distancia entre a representação z da sequência e o centro c for superior ao valor do raio, é então considerada uma anomalia:

$$Anomalia_{\text{VHM}}(S) = \begin{cases} 1, se\ \|z - c\|^2 > R^2 \\ 0, caso\ contrário \end{cases}$$

Assim sendo, o LogBERT apresenta vantagens como a contextualização bidirecional, ao considerar o contexto completo à esquerda e a direita de cada evento. A sua flexibilidade e generalização deve-se à sua capacidade de adaptar tipos de sistemas e formatos

DOI: https://doi.org/10.29352/mill0220e.41569

de *log*. No entanto, possui limitações como o custo computacional visto que o pré-treino necessita de grandes volumes de dados. A sua eficácia depende da qualidade da tokenização das chaves de *log* e do tamanho da janela de contexto escolhida. Além disso, é menos eficaz na deteção de anomalias cujo padrão sequencial se mantém, mas o conteúdo semântico é anómalo (Guo et al., 2021).

1.4.2 VAE

O *Variational Autoencoder* é um subconjunto dos *Autoencoders*, enquadrando-se no domínio dos modelos generativos baseados em variáveis latentes. Enquanto os *autoencoders* convencionais têm como principal tarefa comprimir os dados de entrada numa representação de dimensões reduzidas e seguidamente reconstruí-los, os VAEs introduzem uma abordagem probabilística que permite não apenas a reconstrução dos dados, como também gerar novas amostras realistas (Asperti et al., 2021; Bergmann & Stryker, 2024).

Diferentemente de um *autoencoder* tradicional, que codifica diretamente os dados de entrada de um vetor latente fixo, como por exemplo, uma imagem de proporções 28x28, o VAE aprende uma representação probabilística contínua do espaço latente. Assim sendo, o codificador deste algoritmo transforma os dados de entradas em dois vetores distintos, nomeadamente, μ que retrata a média e σ que caracteriza o desvio padrão de uma distribuição latente. Estes dois parâmetros definem uma distribuição de probabilidade através do espaço latente de forma contínua e estocástica (*Variational AutoEncoders,* n.d.).

Ao considerar x um vetor de variáveis observadas, proveniente de um sistema subjacente fixo, é dado um conjunto de dados $D = \{x^{(1)}, x^{(2)}, \dots, x^{(3)}\}$, que supõe-se que as amostras são independentes e identicamente distribuídas, segundo uma distribuição $p^*(x)$. O objetivo é aproximar esta distribuição com um modelo paramétrico $p_\theta(x)$:

$$p_\theta(x) \approx p^*(x)$$

A aprendizagem baseia-se na maximização da verossimilhança dos dados:

$$\log p_\theta(D) = \sum_{x \in D} \log p_\theta(x)$$

Com a introdução de variáveis latente z, o modelo passa a considerar a relação conjunta $p_\theta(x, z)$, sendo a marginal $p_\theta(x)$ obtido por integração:

$$p_\theta(x) = \int p_\theta(x, z) dz = \int p_\theta(z) p_\theta(x|z)\, dz$$

Neste contexto, $p_\theta(z)$ representa a distribuição a *priori* sobre o espaço latente, enquanto $p_\theta(x|z)$ caracteriza a distribuição condicional usada na reconstrução dos dados através de uma amostra z.

Como a integral marginal $p_\theta(x)$ é, em geral, intratável, recorre-se à inferência variacional para aproximar a distribuição posterior $p_\theta(z|x)$ com um modelo auxiliar $q_\theta(z|x)$, parametrizado por ϕ:

$$q_\theta(z|x) \approx p_\theta(z|x)$$

O objetivo do treino fundamenta-se na maximização do limite inferior da evidencia, designadamente, *Evidence Lower Bound* (ELBO), que fornece uma aproximação inferior ao logaritmo da verossimilhança marginal:

$$\log p_\theta(x) = \mathcal{L}_{\theta,\phi(x)} + D_{kL}(q_\theta(z|x) \parallel p_\theta(z|x))$$

Uma vez que a divergência de *Kullback-Leibler* (KL) é sempre positiva, $\mathcal{L}_{\theta,\phi(x)}$ funciona efetivamente com um limite inferior. O ELBO é expresso por:

$$\mathcal{L}_{\theta,\phi}(x) = E_{q\theta(z|x)}[\log p_\theta(z|x)] - D_{kL}(q_\theta(z|x) \parallel p_\theta(z))$$

Deste modo, o ELBO representa o equilíbrio entre a qualidade de reconstrução dos dados e a regularização do espaço latente.

No entanto, a retropropagação de *gradients* surge como uma limitação, visto que as variáveis aleatórias tendem a serem instáveis devido à elevada variância. Na tentativa de contornar este problema, o VAE utiliza o chamado truque de reparametrização, no qual o vetor latente z é reescrito como uma transformação determinística de uma variável auxiliar $\in \sim \mathrm{N}(0, \mathrm{I})$:

$$z = \mu + \sigma \odot \epsilon$$

DOI: https://doi.org/10.29352/mill0220e.41569

Onde μ e σ são as saídas do codificador e correspondem, respetivamente, à média e ao desvio padrão da distribuição aproximada $q_\theta(z|x)$. Este processo auxilia o método a tornar-se derivável, ao permitir a propagação durante o período de treino. Ao assumir que $q_\theta(z|x) \parallel N(\mu, \sigma^2 I)$ e que $p_\theta(z) = N(0, I)$, a divergência KL entre essas duas distribuições gaussiana multivariada poder computada de forma analítica:

$$D_{kL}(q_\theta(x|z) \parallel p_\theta(z)) = \frac{1}{2}\sum_{j=1}^{k}(\sigma_j^2 + \mu_j^2 - 1 - \log \sigma_j^2)$$

Este resultado é fundamental para a formulação da função de perda final do algoritmo, que combina a reconstrução dos dados com a regularização do espaço latente. Esta estrutura possibilita que o VAE possua a capacidade de aprender representações latentes consistentes, estruturada e apropriada para gerar dados realistas (Zemouri et al., 2022).

1.4.3 DeepLog

O DeepLog surge como uma abordagem assente em *deep learning*, desenvolvido para analisar e monitorizar eventos de *logs*, tirando partido das redes neuronais recorrentes. Construído de acordo com a arquitetura LSTM, este modelo foi concebido para operar em tempo real e, portanto, possibilita o processamento contínuo de evento à medida que são produzidos. Por ser inspirado em técnicas PLN, o DeepLog interpreta os registos como sequências estruturadas, semelhantes a frases linguísticas, permitindo a aprendizagem de padrões típicos de execução com base no histórico de eventos. Deste modo, é capaz de identificar desvios que possam indicar uma potencial anomalia, mesmo em ambientes amplamente dinâmicos e com exigências de resposta imediata.

A arquitetura do DeepLog é composta por três componentes principais, nomeadamente, o modelo de deteção de anomalias baseado nas chaves de *log*, o modelo de deteção de anomalias em valores contínuos e o modelo de fluxo de trabalho, também conhecido como *workflow.*

O modelo de deteção de anomalias nas chaves de *logs*, é utilizada uma rede LSTM para estimar a probabilidade condicional do próximo evento m_{t+1} através numa janela de eventos antecedentes $w = \{m_t - h, \dots, m_t\}$.Assim sendo, a previsão é dada pela seguinte expressão:

$$Pr[m_{t+1} = k_i | \, w] = softmax\big(f_{LSTM}(w)\big)$$

Eventos cujas chaves não se encontrem entre os d valores mais prováveis são considerados anómalos.

O segundo componente trata da deteção de anomalias em valores contínuos, como tempos de resposta ou métricas de desempenho. Estes são tratados como séries temporais multivariadas, ao permitir que o modelo antecipe os valores futuros dos parâmetros. A discrepância entre os valores previstos e os valores observados é calculada através do erro quadrático médio (MSE), definido pôr:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(v_i^{\text{real}} - v_i^{\text{previsto}}\right)^2$$

No decorrer do treino, o modelo ajusta uma distribuição *Gaussiana* sobre os erros de previsão, o que possibilita definir limiares estatísticos para identificar desvios anómalos.

Por fim, através do modelo de fluxo de trabalho, o algoritmo reconstrói os percursos normais de execução de diferentes tarefas do sistema. Este mecanismo permite identificar ramificações, concorrência e ciclos nos processos, oferecendo uma visão mais aprofundada para diagnosticar comportamento atípicos.

Entre os diversos benefícios do DeepLog, destaca-se a sua elevada taxa de precisão, com valores *F-measure* superiores a 96% mesmo com conjuntos de dados de treino reduzidos. Outra característica relevante é a sua capacidade de atualização incremental, ou seja, o modelo adapta-se recorrendo ao *feedback* do utilizador, sem a necessidade de um reprocessamento completo. Além disso, o algoritmo não requer dados categorizados, exigindo apenas um conjunto representativo do comportamento normal. A inferência dos *workflows* adiciona uma camada adicional de interoperabilidade, que facilita na identificação da origem e contexto das anomalias.

Apesar das suas vantagens, o DeepLog apresentas algumas limitações. A sua sensibilidade perante a ordem dos eventos pode comprometer o desempenho em sistemas com múltiplas tarefas em execução concorrente, sendo indispensável a utilização de identificadores para segmentar corretamente os fluxos. Adicionalmente, a fiabilidade do modelo depende da precisão na extração de chaves e parâmetros dos *logs*, o que requer ferramentas de *parsing* fiáveis. Por último, a natureza computacionalmente intensiva das redes LSTM pode representar um entrave em ambientes com restrições de recursos (Du et al., 2017b).

2. RESULTADOS

Com o objetivo de definir uma abordagem metodológica eficaz para a análise de *logs* no contexto institucional, foi realizada esta pesquisa abrangente dos diversos modelos de IA existentes. Esta investigação visa identificar, dentro de cada categoria, os

DOI: https://doi.org/10.29352/mill0220e.41569

modelos mais adequados à deteção de intrusões e comportamentos anómalos, tendo em consideração as especificidades dos dados normalmente presentes em ficheiros de *log*.
Para uma análise comparativa mais estruturada e intuitiva dos modelos de Inteligência Artificial examinados, as suas características e desempenho forma classificados qualitativamente. As categorias principais utilizadas são 'Muito Alta', 'Alta', 'Moderada' e 'Baixa', embora alguns critérios possam usar termos relacionados para refletir nuances específicas. Estas classificações não representam valores quantitativos abso
lutos, mas sim uma síntese interpretativa do consenso predominante e das tendências observadas na literatura científica revisada para cada critério avaliado.
A interpretação das categorias referidas anteriormente é a seguinte:

- 'Muito Alta' – Indica um desempenho ou característica superior, destacando-se significativamente na maioria dos cenários e estudos avaliados. Representa um ponto forte excecional do modelo para o critério específico;
- 'Alta' – Sugere um desempenho ou característica fiável, ao demonstrar uma eficácia elevada na maioria das aplicações. É uma indicação de um ponto forte consistente;
- 'Moderada', 'Média' – Reflete um desempenho ou característica intermédia ou equilibrada, que pode ser eficaz em certas condições ou apresentar um balanço entre vantagens e desvantagens;
- 'Baixa' – Sinaliza um desempenho ou característica limitada, com eficácia reduzida ou dependência de condições muito específicas para ser viável. Representa uma fraqueza significativa do modelo para o critério em questão.

Adicionalmente, alguns critérios utilizam classificações específicas:

- Parametrização ('Simples', 'Complexa') – Refere-se à dificuldade e ao número de hiperparâmetros que necessitam de ajuste rigoroso para otimizar o desempenho do modelo. 'Simples' indica poucos parâmetros ou ajuste intuitivo e 'Complexa' indica um número elevado de parâmetros e/ou dificuldade significativa no seu ajuste;
- Suporte a Múltiplas Classes ('Sim', 'Limitado') – 'Sim' significa que o modelo lida nativamente com classificações multiclasse. 'Limitado' implica que o modelo foi concebido primariamente para problema binários e requer estratégia adicionais para lidar com múltiplas classes, o que pode aumentar a complexidade ou custo computacional;
- Tipos de Dados ('Temporal', 'Textual') – Descreve a natureza dos dados para os quais o modelo é mais eficazmente concebido ou aplicado, baseando-se na sua arquitetura fundamental e nos domínios de aplicação típicos.

Para finalizar, critérios como 'Custo Computacional' ou 'Desempenho Computacional' são avaliados de forma inversa em relação à preferência, ou seja, um 'Custo Computacional Alto' ou 'Desempenho Computacional Baixo' seria considerado menos favorável, ao passo que 'Baixo' para custo ou 'Alto' para desempenho são desejáveis.

2.1 Análise Comparativa dos modelos Machine Learning Supervisionado

Tabela 1 - ML Supervisionado

	Random Forest	XGBoost	SVM
Sensibilidade a Padrões Anómalos	ALTA	ALTA	ALTA
Resistência a Ruído	ALTA	ALTA	MODERADA
Custo Computacional	MODERADO	ALTO	ALTO
Velocidade de Treino	MODERADA	BAIXA	BAIXA
Parametrização	SIMPLES	COMPLEXA	MODERADA
Suporte a Múltiplas Classes	SIM	SIM	LIMITADO
Interoperabilidade	MÉDIA	BAIXA	BAIXA
Adaptação a Dados Desequilibrados	MODERADO	ALTA	BAIXA

De acordo com a análise apresentada na Tabela 1, verifica-se que o modelo *Random Forest* destaca-se como uma das abordagens mais promissoras para a deteção de ataques e intrusões. A sua resistência a possíveis ruídos, aliada à sua eficiência na generalização e à facilidade de aplicação em ambientes com elevada dimensionalidade, tornam-no particularmente indicado no tratamento de *logs* com estruturas e padrões diversos. O XGBoost, apesar de mostrar um desempenho competitivo e um forte poder preditivo, exige um processo de parametrização mais complexo e uma maior capacidade computacional, o que eventualmente poderá limitar a sua aplicação em sistemas em que os recursos sejam mais restritos. Por sua vez, o SVM, apesar da sua eficácia teórica em contextos binários e de elevada dimensionalidade, revela-se menos adequado para conjunto de dados volumosos e com mais de duas categorias, levando assim a necessidade de adicionar estratégia na classificação de várias classes e um maior custo em termos de processamento. Deste modo, os resultados analíticos sustentam a escolha do *Random Forest* como o modelo de referência a ser aplicado dentro do contexto do projeto.

DOI: https://doi.org/10.29352/mill0220e.41569

2.2 Análise Comparativa dos modelos de Machine Learning Não Supervisionado

Tabela 2 - ML Não Supervisionado

	DBSCAN	K-Means	Isolation Forest
Deteção de Intrusões	BOA	FRACA	MUITO ALTA
Sensibilidade a Comportamentos Ocultos	MODERADA	BAIXA	ELEVADA
Robustez a Ruido	ALTA	BAIXA	ALTA
Necessidade de Conhecimento Prévio	MÉDIA	ALTA	BAIXA
Interpretação de Eventos Suspeitos	MÉDIA	ALTA	MÉDIA
Eficácia em Ambientes de Tráfego Variável	ALTA	FRACA	ALTA
Capacidade de lidar com Grandes Volumes de Dados	MÉDIA	ALTA	MUITO ALTA

Conforme a comparação realizada, constata-se que o algoritmo *Isolation Forest* destaca-se como o mais adequado para a classificação de *logs*. A sua elevada sensibilidade a padrões anómalos, em conjunto com a sua resistência face a eventos atípicos e à baixa necessidade de parametrização, tornam-no particularmente ideal em ambientes em que a alta densidade de dados como é recorrente na diversidade de registos. O DBSCAN, apesar de evidenciar uma excelente capacidade na identificação de comportamentos suspeitos, apresenta uma maior sensibilidade na densidade dos dados e uma menor escalabilidade, podendo comprometer a sua aplicação em contextos operacionais de grande escala. Em contrapartida, o K-Means, demonstra limitações substanciais no que diz respeito à deteção de eventos anómalos isolados, revelando-se mais adequado em tarefas de segmentação de padrões conhecidos do que para identificação de ataques desconhecidos ou pouco recorrentes.

2.3 Análise Comparativa dos modelos Deep Learning Supervisionado

Tabela 1 - DL Supervisionado

	LSTM	GRU	BERT
Tipos de Dados Mais Indicados	TEMPORAL	TEMPORAL	TEXTUAL
Retenção de Contexto	ALTA	MODERADA	MUITO ALTA
Desempenho Computacional	BAIXO	ALTO	MODERADO
Velocidade de Treino	LENTA	RÁPIDA	MÉDIA
Adequação à Deteção de Intrusões	ALTA	MODERADA	MUITO ALTA
Necessidade de Dados Categorizados	ALTA	ALTA	ALTA
Interpretação do Modelo	BAIXA	BAIXA	MÉDIA

Segundo a análise comparativa efetuada, o modelo BERT destaca-se como o mais eficiente no teor do estudo, em virtude da sua capacidade de compreender o contexto textual dos registos e de identificar padrões linguísticos relacionados a possíveis comportamentos suspeitos. A sua arquitetura bidirecional torna este algoritmo especialmente apropriado para a análise de *logs* com informação detalhada e contextualizada. O LSTM apesar de se mostrar eficiente na deteção de padrões temporais mais extensos, como tentativas de acessos distribuídas ao longo do tempo, apresenta uma maior complexidade e tempo de treino, o que pode limitar a sua aplicação na prática. Por outro lado, o GRU, é um método mais leve e eficaz, destacando-se em cenários que exigem uma deteção rápida, embora seja sacrificada alguma profundidade na compreensão sequencial dos dados. Assim sendo, a escolha do modelo ideal depende dos requisitos específicos da tarefa em concreto. Quando se trata de registos ricos em linguagem natural, como as mensagens de erro ou descrições de eventos, o Bert torna-se o mais apropriado. O LSTM é vantajoso em cenário onde os ataques manifestam-se de modo sequencial no decorrer do tempo e, por fim, o GRU surge como uma alternativa eficiente na deteção rápida de padrões suspeitos, particularmente útil em ambientes menos exigentes em termos de processamento.

2.4 Análise Comparativa dos modelos Deep Learning Não Supervisionado

Tabela 2 - DL Não Supervisionado

	LogBert	VAE	DeepLog
Sensibilidade a Anomalias	MUITO ALTA	ALTA	ALTA
Interoparebilidade de Padrões Complexos	ELEVADA	MODERADA	ELEVADA
Eficiência à Variação nos Logs	ALTA	MODERADA	BAIXA
Custo Computacional	ELEVADO	MODERADO	ELEVADO
Interpretação dos Resultados	MODERADA	BAIXA	ALTA
Adaptação a Dados Ruidosos	ALTA	ALTA	MODERADO
Aplicabilidade Geral à Deteção de Intrusões	MUITO ALTA	MODERADA	ALTA

DOI: https://doi.org/10.29352/mill0220e.41569

Com base na comparação realizada, verifica-se que o modelo LogBert apresenta o melhor desempenho na deteção de intrusões, devido à sua elevada capacidade de compreender relações semânticas complexas e ao facto de não exigir dados categorizados. O DeepLog mostra-se eficaz na deteção de alterações em sequências de eventos, sendo útil em sistemas com fluxos de execução bem definidos, embora dependa fortemente da qualidade da segmentação de *logs*. Por sua vez, o VAE, apesar de ser flexível na reconstrução de padrões normais, revela uma menor precisão na identificação de intrusões complexas, devido à baixa interoperabilidade e à sensibilidade ao treino. Entre os modelos analisados, o LogBert surge como a solução mais eficiente e versátil para este tipo de tarefa.

CONCLUSÃO

O presente artigo teve como objetivo examinar e comparar diversos modelos de IA aplicados à deteção de anomalias em eventos de *logs*, com o propósito de identificar os mais adequados para futuras implementações práticas em cibersegurança.

A avaliação de diferentes abordagens de *Machine Learning* e *Deep Learning*, permitiu delinear uma visão abrangente das suas capacidades e limitações face ao problema em estudo. Tendo em conta os critérios definidos, foram selecionados os modelos que, teoricamente, mostram-se mais promissores.

No entanto, é importante salientar que a escolha dos modelos mais apropriados se baseou numa análise teórica, uma vez que os dados específicos da rede institucional ainda não se encontram disponíveis. Deste modo, permanece em aberto a possibilidade de outros modelos analisados revelarem-se mais eficazes, caso adequem-se melhor às características concretas dos dados a utilizar.

Neste sentido, sublinha-se que este trabalho contribui para uma base sólida para orientar a seleção inicial de modelos, porém não representa o ponto final da investigação. Assim sendo, a etapa seguinte passará pela implementação prática dos modelos previamente selecionados, o que permitirá confirmar ou ajustar as escolhas teóricas efetuadas até ao momento.

CONTRIBUIÇÃO DOS AUTORES

Conceptualização, P.C., F.S e P.L.; tratamento de dados, P.C., F.S. e P.L.; análise formal, P.C., F.S. e P.L.; aquisição de financiamento, P.C., F.S. e P.L.; investigação, P.C., F.S. e P.L.; metodologia, P.C., F.S. e P.L.; administração do projeto, P.C., F.S. e P.L.; recursos, P.C., F.S. e P.L.; programas, P.C., F.S. e P.L.; supervisão, P.C., F.S. e P.L.; validação, P.C., F.S. e P.L.; visualização, P.C., F.S. e P.L.; redação – preparação do rascunho original, P.C., F.S. e P.L.; redação – revisão e edição, P.C., F.S. e P.L.

CONFLITO DE INTERESSES

Os autores declaram não existir conflito de interesses.

REFERÊNCIAS BIBLIOGRÁFICAS

Abdiyeva-Aliyeva, G., Aliyev, J., & Sadigov, U. (2022). Application of classification algorithms of machine learning in cybersecurity. *Procedia Computer Science*, *215*, 909–919. https://doi.org/10.1016/J.PROCS.2022.12.093

Abellán, J., Mantas, C. J., & Castellano, J. G. (2017). A Random Forest approach using imprecise probabilities. *Knowledge-Based Systems*, *134*, 72–84. https://doi.org/10.1016/J.KNOSYS.2017.07.019

Ahmed, M., Seraj, R., & Islam, S. M. S. (2020). The k-means Algorithm: A Comprehensive Survey and Performance Evaluation. *Electronics*, *9*(8), 1295. https://doi.org/10.3390/electronics9081295

Al Farizi, W. S., Hidayah, I., & Rizal, M. N. (2021). Isolation forest based anomaly detection: A systematic literature review. *2021 8th International Conference on Information Technology, Computer and Electrical Engineering, ICITACEE 2021*, 118–122. https://doi.org/10.1109/ICITACEE53184.2021.9617498

Amaratunga, D., Cabrera, J., & Lee, Y. S. (2008). Enriched random forests. *Bioinformatics*, *24*(18), 2010–2014. https://doi.org/10.1093/BIOINFORMATICS/BTN356

Asperti, A., Evangelista, D., & Loli Piccolomini, E. (2021). A Survey on Variational Autoencoders from a Green AI Perspective. *SN Computer Science*, *2*(4), 1–23. https://doi.org/10.1007/S42979-021-00702-9/FIGURES/20

Astekin, M., Zengin, H., & Sözer, H. (2018). Evaluation of distributed machine learning algorithms for anomaly detection from large-scale system logs: A case study. In *Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018* (pp.2071–2077). IEEE. https://doi.org/10.1109/BIGDATA.2018.8621967

Aung, Y. Y., & Min, M. M. (2017). An analysis of random forest algorithm based network intrusion detection system. *Proceedings - 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2017*, (pp.127–132). IEEE. https://doi.org/10.1109/SNPD.2017.8022711

DOI: https://doi.org/10.29352/mill0220e.41569

Belcic, I., & Stryker, C. (n.d.). *What Is supervised learning?* IBM. Retrieved April 4, 2025, from https://www.ibm.com/think/topics/supervised-learning

Bergmann, D., & Stryker, C. (2024). *What is a Variational Autoencoder?* IBM. https://www.ibm.com/think/topics/variational-autoencoder

Chabchoub, Y., Togbe, M. U., Boly, A., & Chiky, R. (2022). *An In-Depth Study and Improvement of Isolation Forest*. IEEE Access, 10, 10219–10237. https://doi.org/10.1109/ACCESS.2022.3144425

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, (pp. 1724–1734). https://doi.org/10.3115/v1/d14-1179

Chung, H., & Shin, K. S. (2018). Genetic algorithm-optimized long short-term memory network for Stock Market Prediction. *Sustainability, 10*(10)*,* 3765. https://doi.org/10.3390/SU10103765

de Amorim, R. C. (2016). A survey on feature weighting based k-means algorithms. *Journal of Classification*, *33*(2), 210–242. https://doi.org/10.1007/S00357-016-9208-4/METRICS

Du, M., Li, F., Zheng, G., & Srikumar, V. (2017). DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the ACM Conference on Computer and Communications Security*, (pp. 1285–1298). https://doi.org/10.1145/3133956.3134015

El Mrabet, M. A., El Makkaoui, K., & Faize, A. (2021). Supervised machine learning: A survey. IN *2021 4th International Conference on Advanced Communication Technologies and Networking (CommNet)*, (pp. 1–10). IEEE. https://doi.org/10.1109/CommNet52204.2021.9641998

Feng, W., Ma, C., Zhao, G., & Zhang, R. (2020). FSRF:An improved random forest for classification. In *Proceedings of 2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications, AEECA 2020*, (pp. 173–178). https://doi.org/10.1109/AEECA49918.2020.9213456

Giradin, L., & Brodbeck, D. (2002). *A visual approach for monitoring logs*. USENIX Association. https://abrir.link/NglCN

Graves, A. (2012). Long short-term memory. *In Supervised sequence labelling with recurrent neural networks* (pp. 37-45). Springer. https://doi.org/10.1007/978-3-642-24797-2_4

Guo, H., Yuan, S., & Wu, X. (2021). LogBERT: Log anomaly detection via BERT. In *Proceedings of the International Joint Conference on Neural Networks*, (pp. 1-8). IEEE. https://doi.org/10.1109/IJCNN52387.2021.9534113

Hashemi-Pour, C., & Lutkevich, B. (n.d.). *What is the BERT language model?* TechTarget. https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model

Gohiya, H. M., Lohiya H., & Patidar, K. (2018). A survey of XGBoost system. *International Journal of Advanced Technology & Engineering Research (IJATER)*, *8*(3).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/NECO.1997.9.8.1735

Holdsworth, J., & Scapicchio, M. (n.d.). *What Is Deep Learning?* IBM. https://www.ibm.com/think/topics/deep-learning

Kulkarni, O., & Burhanpurwala, A. (2024). A survey of advancements in DBSCAN clustering algorithms for big data. In *2024 3rd International Conference on Power Electronics and IoT Applications in Renewable Energy and Its Control, PARC 2024* (pp. 106–111). IEEE. https://doi.org/10.1109/PARC59193.2024.10486339

Landauer, M., Onder, S., Skopik, F., & Wurzenberger, M. (2023). Deep learning for anomaly detection in log data: A survey. *Machine Learning with Applications*, *12*, 100470. https://doi.org/10.1016/J.MLWA.2023.100470

Li, Y., & Wu, H. (2012). A clustering method based on K-means algorithm. *Physics Procedia*, *25*, 1104–1109. https://doi.org/10.1016/J.PHPRO.2012.03.206

Liu, H., & Lang, B. (2019). Machine learning and deep learning methods for intrusion detection systems: A survey. *Applied Sciences 2019, 9*(20)*,* 4396. https://doi.org/10.3390/APP9204396

Marinho, T. L. (2021). *Otimização de hiperparâmetros do XGBoost utilizando meta*-aprendizagem [Dissertação de Mestrado, Universidade Federal de Alagoas]. Repositório Institucional da UFAL. http://www.repositorio.ufal.br/jspui/handle/123456789/9851

Mello, T. R. de. (2021). *Comparativo entre redes neurais recorrentes GRU e LSTM para a predição de instrumentos financeiros* [Trabalho de Conclusão de Curso].

Na, S., Xumin, L., & Yong, G. (2010). Research on k-means clustering algorithm: An improved k-means clustering algorithm. In *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*, (pp. 63–67). IEEE. https://doi.org/10.1109/IITSI.2010.74

Parilama, M., Lopez, D., & Senthilkumar, N. C. (2011). A survey on density based clustering algorithms for mining large spatial databases. *International Journal of Advanced Science and Technology, 31*. https://abrir.link/PVZdG

DOI: https://doi.org/10.29352/mill0220e.41569

Podlodowski, L., & Kozlowski, M. (2019). Application of XGBoost to the cyber-security problem of detecting suspicious network traffic events. In *Proceedings - 2019 IEEE International Conference on Big Data, Big Data 2019*, (pp. 5902–5907). https://doi.org/10.1109/BIGDATA47090.2019.9006586

Resende, P. A. A., & Drummond, A. C. (2018). A survey of random forest based methods for intrusion detection systems. *ACM Computing Surveys (CSUR)*, *51*(3). https://doi.org/10.1145/3178582

Schölkopf, Bernhard., & Smola, A. J. (2002). *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT Press.

Singh, H. V., Girdhar, A., & Dahiya, S. (2022). A Literature survey based on DBSCAN algorithms. In *Proceedings - 2022 6th International Conference on Intelligent Computing and Control Systems, ICICCS 2022*, (pp. 751–758). IEEE. https://doi.org/10.1109/ICICCS53718.2022.9788440

Singh, P., & Meshram, P. A. (2018). Survey of density based clustering algorithms and its variants. In *Proceedings of the International Conference on Inventive Computing and Informatics, ICICI 2017*, (pp. 920–926). IEEE. https://doi.org/10.1109/ICICI.2017.8365272

Somvanshi, M., Chavan, P., Tambade, S., & Shinde, S. V. (2017). A review of machine learning techniques using decision tree and support vector machine. In *Proceedings - 2nd International Conference on Computing, Communication, Control and Automation, ICCUBEA 2016*. IEEE. https://doi.org/10.1109/ICCUBEA.2016.7860040

Masolo, C. (2017). Supervised, unsupervised and deep learning. *TDS Archive, Medium.* https://medium.com/data-science/supervised-unsupervised-and-deep-learning-aa61a0e5471c

Van Houdt, G., Mosquera, C., & Nápoles, G. (2020). A review on the long short-term memory model. *Artificial Intelligence Review*, *53*(8), 5929–5955. https://doi.org/10.1007/S10462-020-09838-1/TABLES/1

Variational AutoEncoders. (n.d). GeeksforGeeks. https://www.geeksforgeeks.org/variational-autoencoders/

Wang, H., Li, J., & Li, Z. (2024). AI-generated text detection and classification based on BERT deep learning algorithm. *arXiv.* https://arxiv.org/abs/2405.16422v1

Pradhan, A. (2012). Support Vector Machine -A Survey. *International Journal of Emerging Technology and Advanced Engineering, 2*(8).

Xu, D., Wang, Y., Meng, Y., & Zhang, Z. (2018). An improved data anomaly detection method based on isolation forest. In *Proceedings - 2017 10th International Symposium on Computational Intelligence and Design, ISCID 2017* (pp. 287–291). IEEE. https://doi.org/10.1109/ISCID.2017.202

Xu, H., Pang, G., Wang, Y., & Wang, Y. (2023). Deep Isolation Forest for anomaly detection. *IEEE Transactions on Knowledge and Data Engineering*, *35*(12), 12591–12604. https://doi.org/10.1109/TKDE.2023.3270293

Yen, S., & Moh, M. (2019). Intelligent log analysis using machine and deep learning. In M.A. Ferrag & A.Ahmim (Eds.), *Machine learning and cognitive science applications in cyber security* (pp. 154-189). IGI Global. https://doi.org/10.4018/978-1-5225-8100-0.CH007

Zemouri, R., Levesque, M., Boucher, E., Kirouac, M., Lafleur, F., Bernier, S., & Merkhouf, A. (2022). Recent research and applications in variational autoencoders for industrial prognosis and health management: A Survey. In *Proceedings - 2022 Prognostics and Health Management Conference, PHM-London 2022*, (pp. 193–203). IEEE. https://doi.org/10.1109/PHM2022-LONDON52454.2022.00042

Zhang, Y., Lin, J., Zhao, L., Zeng, X., & Liu, X. (2021). A novel antibacterial peptide recognition algorithm based on BERT. *Briefings in Bioinformatics*, *22*(6), 1–11. https://doi.org/10.1093/BIB/BBAB200

Zhao, W.-L., Deng, C.-H., & Ngo, C.-W. (2018). K-means: A revisit. *Neurocomputing*, *291*, 195–206. https://doi.org/10.1016/j.neucom.2018.02.072