Abordagem de Aprendizado Incremental para Sistemas de Detecção de Intrusão: Combatendo o Esquecimento Catastrófico

Rafael S. Lemos[1], Francisco J. Gomes[1], Carlielson D. Souza[1], César L. C. Mattos[1], José D. C. Neto[2], Jarelio G. da S. Filho[2], Nicksson C. A. de Freitas[2], Rodrigo C. S. Costa[3], Emanuel B. Rodrigues[1]

[1]Universidade Federal do Ceará (UFC)
Av. da Universidade, 2853 – CEP 60020-181 – Fortaleza – CE – Brasil

{rafael.lemos, janielgomes, carlielsondamasceno}@alu.ufc.br
{cesarlincoln, emanuel}@dc.ufc.br

[2]SiDi
Av. República do Líbano, 251 – CEP 51110-160 – Recife – PE – Brasil

{j.carneiro, j.filho, nicksson.a}@sidi.org.br

[3]Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)
R. Dez, S/N – CEP 62680-000 – Paracuru – CE – Brasil

rodrigo.costa@ifce.edu.br

Abstract. *With the increase in the storage of corporate data in the digital environment, cybercriminals are attracted and can cause serious damage. In this context, Intrusion Detection Systems are tools used for detection of cyber attacks.Currently, these systems have employed traditional machine learning models. However, such models require periodic retraining with the entire database to detect new intrusion techniques. In this work, we propose the use of incremental learning to incorporate knowledge of new attack patterns without the need to retrain the model from scratch and avoid catastrophic forgetting. For this purpose, an incremental learning methodology was developed, covering data handling, model construction, and evaluation of the final models. Experimental results demonstrate that the constructed model learned new attacks without forgetting what it had already learned. The final metrics obtained for precision, recall, and F1-score were above 87%, and an accuracy of 98.89% was achieved.*

Resumo. *Com o aumento do armazenamento de dados empresariais no ambiente digital, cibercriminosos são atraídos e podem causar sérios prejuízos. Nesse contexto, Sistemas de Detecção de Intrusão são ferramentas utilizadas para detecção de ataques cibernéticos. Atualmente, esses sistemas têm empregado modelos tradicionais de aprendizado de máquina. No entanto, tais modelos exigem retreinamento periódico com toda base de dados, permitindo a detecção de novas técnicas de intrusão. Neste trabalho, propõe-se o uso de aprendizagem incremental para incluir o conhecimento de novos padrões de ataque sem a necessidade de retreinar o modelo do início e não sofrer com o esquecimento catastrófico. Para este fim, foi desenvolvida uma metodologia de aprendizagem incremental, do tratamento dos dados à construção e avaliação*

dos modelos finais. Os resultados experimentais demonstram que o modelo construído aprendeu novos ataques sem esquecer o que já havia aprendido. As métricas finais obtidas de precisão, revocação e F1-score foram acima de 87% e uma acurácia de 98,89% foi alcançada.

1. Introdução

Atualmente, os dados são ativos que possuem um grande valor para as empresas e, a maioria delas, se não todas, os armazenam digitalmente. Consequentemente, isso atraiu pessoas mal-intencionadas para o mundo cibernético. Os cibercriminosos estão sempre inovando a maneira de atacar, buscando alguma falha de segurança para acessar dados sensíveis, obtidos com invasões a sistemas privados, sendo necessário portanto, ferramentas automáticas que as dificultem (Toupas, 2019).

Existem inúmeras ferramentas que podem ser usadas na proteção cibernética dos sistemas das instituições. Uma dessas ferramentas é o Sistema de Detecção de Intrusão (*Intrusion Detection System* - IDS), que pode ser utilizado para detectar se houve alguma invasão ao sistema ou à rede. Caso aconteça, o IDS alertará o time de segurança sobre o acesso que foi considerado malicioso. Segundo Amalapuram et al. (2022), tal mecanismo pode ser realizado por meio da identificação de assinatura ou por detecção de anomalia, sendo que o primeiro se baseia num conjunto de regras e o segundo, de acordo com atividades que fogem ao padrão considerado normal.

Segundo Kenyon (2020), a área de pesquisa de IDS baseada em Aprendizado de Máquina (*Machine Learning* - ML) depende de conjuntos de dados que permitam avaliar comparativamente técnicas e abordagens para obter soluções mais eficientes. Sendo assim, é crucial ter um conjunto de dados organizados e bem tratados, permitindo uma análise mais precisa e confiável. Os ataques de intrusão costumam gerar bancos de dados desbalanceados, ou seja, as instâncias das classes não ataques costumam ser muito mais numerosas que as das classes de ataques, tornando o processo de construção de modelos de ML mais desafiador. Nesse contexto, o balanceamento do *dataset* é uma técnica relevante a ser considerada, evitando que o modelo seja tendencioso em relação a uma classe específica (Abd Elrahman and Abraham, 2013). Dessa forma, é possível melhorar a precisão da classificação e evitar falsos positivos ou negativos. Este presente trabalho concentrou-se em métodos de subamostragem da classe majoritária, removendo-se aleatoriamente algumas das instâncias da classe de tráfego normal.

Segundo Mitchell et al. (2007), o objetivo do ML é a construção de programas que melhorem seu desempenho por meio de amostras. Embora os resultados obtidos por essa abordagem possam ser promissores, os algoritmos tradicionais de ML usam modelos estáticos, incapazes de adaptar seu comportamento ao longo do tempo. Portanto, caso surjam novos dados, potencialmente pertencentes a novas classes não encontradas antes, torna-se necessário reiniciar o processo de treinamento do modelo. Essa prática rapidamente torna-se intratável para fluxos de dados constantes que os IDSs recebem. Uma alternativa é usar o Aprendizado Incremental (*Incremental Learning* - IL) para realizar o incremento de classes. Segundo Data (2021), IL é um modelo que nos permite realizar seu treinamento, conforme novos ataques vão surgindo na rede, sendo um dos problemas do IL o esquecimento catastrófico.

Neste trabalho, são apresentados aprimoramentos em um sistema de detecção de

intrusão de rede (*Network Intrusion Detection System* - NIDS) por meio do uso de um modelo de IL, o qual demonstra a capacidade de aprender novas classes sem sofrer com o esquecimento catastrófico. Além disso, o modelo proporciona uma classificação altamente eficaz dos ataques baseados em assinaturas. Com o objetivo de realizar essa tarefa, foi proposta uma metodologia de treinamento incremental de novas classes que vão desde a preparação dos dados até o treinamento do modelo, finalizando com a fase de análise dos resultados. Para a criação dos modelos, foi utilizada a biblioteca *River* (Montiel et al., 2021) em conjunto com modelos de classificação, sendo eles o Classificador de Árvore Adaptativa de Hoeffding (*Hoeffding Adaptive Tree Classifier* - HAT) e o Classificador Adaptável de Floresta Aleatória (*Adaptive Random Forest Classifier* - ARF). A biblioteca foi escolhida por sua capacidade de lidar com fluxos de dados contínuos e implementação eficiente de algoritmos de ML em larga escala.

Na sequência, os trabalhos relacionados são discutidos na Seção 2, seguidos pela descrição da metodologia adotada neste estudo, na Seção 3 e uma aplicação dessa metodologia, na Seção 4. Os resultados e discussões dos experimentos realizados são apresentados na Seção 5, enquanto na Seção 6, denominada de conclusões, evidencia-se a viabilidade e a eficácia do uso de modelos de Aprendizado Incremental em IDSs baseados em assinatura.

2. Trabalhos Relacionados

Diversos trabalhos recentes têm sido realizados a respeito de NIDS. A pesquisa de Shah (2018), por exemplo, faz uso de métodos de ML para otimizar as detecções obtidas pelo IDS *Snort*, com o objetivo de reduzir os falsos positivos presentes nas descobertas de ataque. Os algoritmos usados: Rede Bayesiana (Bayes Net - BN), Árvore de Decisão (*Decision Tree* - DT), Lógica Fuzzy (Fuzzy Logic - FL), Naive Bayes (NB), Máquina de Vetores de Suporte (Support Vector Machine - SVM). Experimentos com os *datasets NSA Snort IDS Alert Logs, DARPA IDS e NSL-KDD* conseguiram melhorias de até 9,0% na redução dos falsos positivos. Os ataques presentes nos *datasets* são respectivamente: falsificação do endereço MAC, envenenamento do DNS, mascaramento do IP, *SSH Attacks* (obter acesso remoto via SSH), *FTP Attacks* (obter acesso à transferência de arquivos), *Scanning Attacks* (escanear o *host* da vítima), *DoS* (um *host* enviando inúmeras requisições para derrubar um serviço), U2R (ganhar acesso *root*), R2L (acesso remoto local) e *Probing Attack* (contornar as medidas de segurança).

Já o trabalho de Louati (2020) faz uso da aplicação de DL em um sistema multiagente utilizando três diferentes algoritmos, sendo eles: *Autoenconder* (AE) , para redução de dimensionalidade do *dataset*; Perceptron Multicamadas *(Multilayer Perceptron - MLP)* e K-vizinhos mais próximos (*K-Nearest Neighbors* - KNN), para classificação dos ataques. A classificação multiclasse realizada alcançou uma taxa de 99,95% de acurácia, utilizando o *dataset KDD CUP99*. O *dataset* usado no trabalho contém os seguintes ataques: *DoS*, *Probe*, *R2L* e *U2L*. Embora seja atual, assim como o trabalho anterior, faz uso de *datasets* mais antigos e modelos de ML convencionais.

Para lidar com o constante aumento dos dados, o trabalho de Horchulhack et al. (2022) faz uso de Redes Adversárias Generativas (GANs) para gerar dados e aumentar um conjunto de dados existente, sendo construído a partir de uma parte do *dataset MawiFlow*, atualizando os modelos treinados por meio da transferência de aprendizado gerados

pela rede GAN. Foram aplicados métodos de ML como DT, RF e MLP, de modo a otimizar o tempo de classificação e diminuir os falsos positivos, que foram reduzidos em até 18,1%. Neste trabalho não foram utilizadas técnicas de balanceamento nem apresentadas as métricas finais obtidas pelos modelos.

Buscando aprimorar os IDSs, a pesquisa de Mahdavi and Mirzaei (2022) propôs o uso de Aprendizado de transferência incremental para sistemas de detecção de intrusão (*Incremental Transfer Learning for Intrusion Detection Systems* - ITL-IDS), que possui a capacidade de trabalhar sem a necessidade de dados com *labels* iniciais, realizando um agrupamento por meio das características disponíveis, de modo que esse aprendizado pode ser incrementado à medida que é aplicado ao modelo, em diferentes momentos, permitindo a inclusão de novos ataques sem conhecimento prévio do sistema, conseguindo unir tanto o IL, quanto o de transferência. Foram usados os *datasets* KDD CUP99 e CIC-IDS2017, onde este último contém os seguintes ataques: *Brute Force FTP*, *Brute Force SSH*, *DoS*, *Heartbleed* (explora a biblioteca de criptografía OpenSSL), *Web Attack* (explora vulnerabilidades web), *Infiltration* (abre um *backdoor*), *Botnet* (rede de *bot*) e *DDoS* (mais de um host enviando inúmeras requisições para derrubar um serviço).

Tarefas de IL precisam lidar com o efeito do esquecimento catastrófico, ou seja, o fenômeno de esquecer o que já aprendeu ao buscar incluir novos padrões. O trabalho de Data (2021) enfrenta essa dificuldade, propondo o algoritmo Redes neurais feedforward profundas de árvore (*Tree deep feedforward neural networks* - T-DFNN) e aplicando-o ao *dataset* CIC-IDS2017. Foi obtido um *F1-score* superior a 85% para cada lote de dados incrementado, reduzindo-se o efeito do esquecimento. Como limitação, o algoritmo T-DFNN precisa de mais etapas computacionais, aumentando o tempo para a classificação.

O objetivo do presente trabalho é obter um modelo capaz de classificar diferentes tipos de ataques, de forma que ele possa receber novos dados e adicioná-los aos já existentes, sem a necessidade de retreinamento. Além disso, é importante garantir que o modelo não sofra de esquecimento catastrófico. Para este fim, foram utilizados os algoritmos de árvores HAT (Adhikari and Morris, 2019) e ARF (Gomes and Bifet, 2017) em conjunto com a biblioteca *River*, visto que os algoritmos em árvore utilizados pela literatura obtém as maiores métricas comparados aos demais modelos de ML. Para tornar os algoritmos mais eficientes, aplicou-se a técnica de balanceamento baseada no método de subamostragem. Nesse trabalho usou-se o *dataset* CIC-IDS2017, construído por Sharafaldin (2018), para realização dos testes.

Tabela 1. Sumário dos trabalhos relacionados

Artigos	Algoritmos	Aprendizado Incremental	Método de Balanceamento
Shah (2018)	BN, DT, FL, NB, SVM	Não	Não
Louati (2020)	AE, K-NN, MLP	Não	Não
Horchulhack et. al (2022)	DT, RF	Não	Sim
Mahdavi et. al (2022)	ITL-IDS	Sim	Não
Data et. al (2021)	T-DFNN	Sim	Não
Este trabalho	ARF, HTA	Sim	Sim

A Tabela 1 faz um comparativo deste trabalho com os trabalhos relacionados, indicando os algoritmos testados, se usaram IL e se implementaram algum método de balanceamento.

3. Abordagem Incremental para Detecção de Intrusão

A abordagem incremental para IDS proposta neste trabalho é capaz de aprender novos ataques, à medida que novos dados são adicionados, mantendo uma taxa de acerto elevada e preservando o conhecimento adquirido sobre ataques anteriores. A seguir serão abordadas as etapas da metodologia proposta.

A Figura 1 apresenta uma proposta de metodologia de IL baseada em modelos de ML para aplicação em IDS. O primeiro passo é o processo de tratamento dos dados do *dataset*, no qual ocorre a remoção dos *outliers*, renomeação e remoção de colunas, união de ataques que possuem assinaturas semelhantes e um balanceamento das classes, esse passo é importante para que os dados cheguem padronizados no modelo. No passo seguinte, é realizada a divisão do conjunto de dados em treino e teste, usando diferentes sementes para gerar os conjuntos de forma aleatória, gerando subconjuntos para medição da performance dos modelos de classificação.

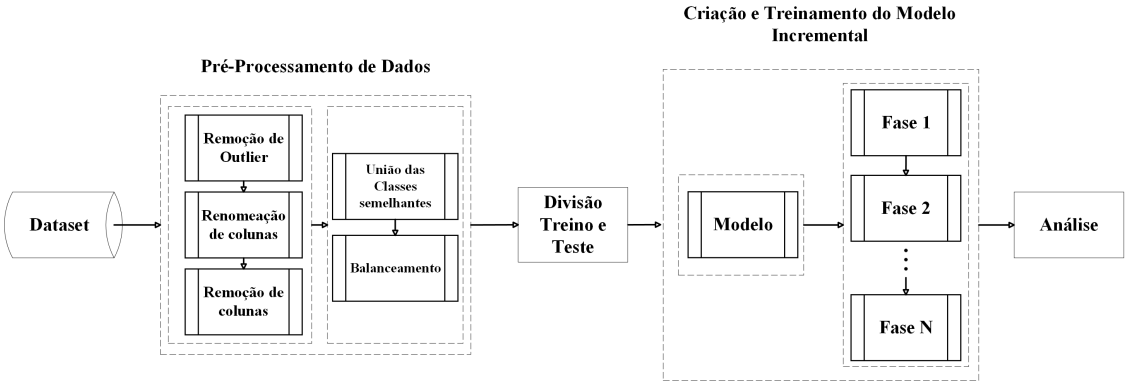

Figura 1. Metodologia de aprendizado incremental baseada em modelos de aprendizado de máquina para aplicação em IDS.

Após as fases de pré-processamento e de divisão do conjunto de dados, chega-se a etapa de criação do modelo que suporta o IL. O treinamento e a avaliação do modelo passa por diferentes fases de incremento de conhecimento: as classes são divididas de acordo com os dados disponíveis e depois são apresentadas ao modelo para treinamento, evitando a repetição do conjunto de treino entre uma fase e outra. Dessa forma, inicia-se o treinamento com a fase inicial e o teste com as suas respectivas classes. Na fase seguinte, o modelo é treinado com as novas classes e testado com as classes presentes na fase atual e nas fases anteriores, sendo executado esse procedimento até a última fase, evitando assim que as classes sejam repetidas durante os treinamentos. O resultado de cada incremento é coletado para verificar se o modelo está se comportando de forma adequada para o IL, ou seja, mantendo as taxas de classificação altas ou sofrendo com o esquecimento das classes aprendidas. Esse procedimento é realizado por cinco repetições, a partir da divisão do conjunto de dados para treino e teste, gerando diferentes conjuntos em cada repetição, até o treinamento e teste do modelo para coleta das métricas e, posterior análise. As métricas

coletadas são precisão, revocação e *F1-score* para cada uma das classes de ataque, além da acurácia geral (macro) do modelo. Essa métricas são importantes para compreender o desempenho do modelo ao fim de todo o processo.

4. Aplicação da Metodologia Proposta

Nesta seção é descrito o experimento realizado para validação da metodologia proposta. O presente trabalho utilizou o *dataset* CIC-IDS2017 para o treinamento e a avaliação dos modelos. O procedimento de IL foi dividido em três fases. Inicialmente, foram treinados os modelos com o *dataset* sem o balanceamento de dados e depois com os dados balanceados para que, posteriormente, pudesse ser avaliada a viabilidade da técnica.

O primeiro modelo de classificação utilizado foi o algoritmo GNB criado com a biblioteca *Scikit-learn*, a fim de analisar o processo em um modelo não adaptado ao aprendizado incremental. Em seguida, foram avaliados os modelos de IL, por meio da biblioteca *River*, para a classificação dos ataques com os algoritmos GNB, HAT e ARF.

4.1. *Dataset* CIC-IDS2017

O *dataset* CIC-IDS2017 criado por Sharafaldin (2018) é considerado um conjunto de dados grande, pelo fato de possuir 80 colunas e 2.830.743 linhas, cuja coluna *Label* identifica o tipo de tráfego. Ele inclui amostras de atividade benigna e ataques comuns ao longo de um período de cinco dias. Um sistema B-Profile descrito por Sharafaldin (2018) foi utilizado para gerar tráfego benigno realista. O tráfego benigno corresponde à interação humana de 25 usuários com base em protocolos de rede padrão, como Protocolo de Transferência de Hipertexto/Seguro (*Hypertext Transfer Protocol/Secure* - HTTPS), Protocolo de Transferência de Arquivos (*File Transfer Protocol* - FTP), *Shell* Seguro (*Secure Shell* - SSH), Protocolo de Acesso a Mensagens da Internet (*Internet Message Access Protocol* - IMAP) e Protocolo de Correio Versão 3 (*Post Office Protocol version 3* - POP3). Além disso, o *dataset* contém os ataques de *Brute Force* FTP e SSH, DoS, *Heartbleed*, *Web Attack*, *Infiltration, Botnet* e DDoS.

4.2. Tratamento e Divisão dos Dados

No tratamento dos dados foram executados os seguintes passos. Primeiro, realizou-se a verificação de valores inválidos, sendo encontrados um total 1.358 dados faltantes, que apesar de serem instâncias rotuladas, as mesmas foram removidas. Em seguida, as colunas foram renomeadas para eliminar espaços em branco no início dos nomes. Determinadas colunas, tais como "*Timestamp*", "*Source IP*", "*Source Port*", "*Destination IP*"e "*Flow ID*"foram removidas, pois essas informações variam de acordo com cada rede e poderiam enviesar os modelos treinados.

Os diferentes tipos de ataque DoS foram agrupados em uma única classe, devido à escassez de amostras para cada tipo específico, aumentando assim o número total de amostras e equilibrando as classes. Além disso, foram excluídas as classes com dados minoritários, "*Infiltration*", "*Web Attack - Sql Injection*"e "*Heartbleed*", que possuíam menos de 500 amostras. Para abordar o desequilíbrio dos dados em relação à classe "*Benign*"(tráfego normal), uma seleção aleatória de apenas uma parte dos dados dessa classe foi feita, evitando assim qualquer viés na seleção. Essas ações foram essenciais para garantir a qualidade e a confiabilidade das análises realizadas posteriormente. Na Tabela

2, é possível observar as classes consideradas e a quantidade de amostras disponíveis. A divisão das classes em cada fase ocorreu de forma aleatória, de modo que pudesse ser aplicada a qualquer *dataset*. Note que são consideradas 3 fases de IL, onde o modelo é apresentado a dados de 3 diferentes classes de tráfego (benigno ou malicioso) em cada fase.

Tabela 2. Distribuição de classes do conjunto de dados CIC-IDS2017 utilizado nesta pesquisa.

Fases incrementais	Categoria	Qtde. amostras
Fase 1	FTP	5.931 (0,93%)
	DDoS	128.014 (20,15%)
	PortScan	90.694 (14,28%)
Fase 2	Benign	209.522 (32,99%)
	DoS	193.745 (30,50%)
	SSH	3.219 (0,51%)
Fase 3	Botnet	1.948 (0,31%)
	Web Attack - Brute Force	1.470 (0,23%)
	Web Attack - XSS	652 (0,1%)

4.3. Criação dos Modelos de Aprendizado de Máquina

O processo de criação dos modelos utilizou os hiperparâmetros padrões das bibliotecas, de modo que os resultados pudessem ser alinhados e comparados. Os próximos tópicos tratam de cada passo realizado.

4.3.1. Treinamento do Modelo Convencional via *Scikit-learn*

A biblioteca em *Python*, chamada *Scikit-learn* e desenvolvida por Pedregosa et al. (2011), é uma biblioteca popular de código aberto para ML. Trata-se de uma ferramenta simples para análise preditiva de dados, construída usando *Numpy*, *SciPy* e *Matplotlib*. A biblioteca possui uma função chamada partial_fit capaz de realizar um ajuste incremental em um lote de amostras, permitindo que os dados sejam divididos e aplicados no treinamento do modelo, porém não está completamente habilitada para realizar o treinamento incremental nos mais variados modelos de ML, como por exemplo, *Decision Tree* e *Random Forest*, bastante utilizados para criação de modelos para IDSs.

Na presente pesquisa, o *Scikit-learn* foi usado para treinar o modelo GNB. Ele é um algoritmo de aprendizado supervisionado baseado na aplicação do teorema de Bayes, com a suposição "ingênua" de independência condicional entre cada par de recursos, dado o valor da variável de classe. O uso da função partial_fit permitiu que as classes fossem sendo adicionadas para o treinamento, de acordo com cada respectiva fase, conforme a Tabela 2. Sendo assim, a primeira fase tratou do incremento das classes FTP, DDoS e *PortScan*, passando pelo treinamento e teste contendo as respectivas classes. Na próxima fase, ao adicionar e treinar as novas categorias *Benign*, DoS e SSH, o conjunto para teste utilizado contém as novas e as antigas classes pertencente à fase 1. A terceira fase segue

o mesmo padrão, no qual o conjunto de testes contém todas as classes desde a primeira fase. Esse processo foi repetido 5 vezes e coletadas as métricas em cada fase para análise posterior, conforme definido na abordagem incremental.

4.3.2. Treinamento do Modelo Incremental via *River*

A biblioteca em *Python*, denominada *River*, é uma ferramenta desenvolvida com o objetivo de ser uma biblioteca amigável para utilizar ML em fluxos de dados (Montiel et al., 2021). De acordo com o mesmo autor, em termos gerais, um fluxo de dados consiste em uma sequência de elementos individuais. No contexto do ML, cada elemento representa uma amostra ou observação. Desse modo, cada amostra é composta por um conjunto de recursos, os quais podem possuir uma estrutura fixa, mantendo os mesmos atributos em todas as amostras, ou podem variar ao longo do tempo, dependendo do caso de uso específico. O *River* é o resultado da fusão das bibliotecas *Creme* (Halford et al., 2019) e *scikit-multiflow* (Montiel et al., 2018). Ela fornece uma variedade de modelos que são capazes de trabalhar em tempo real, lidando com a chegada contínua de novos dados e se adaptando a mudanças nos padrões dos mesmos.

A biblioteca *River* foi usada para treinar o modelo convencional GNB e dois modelos incrementais, a saber: HAT e ARF. HAT é um algoritmo de indução de árvore de decisão incremental projetado para lidar com fluxos contínuos de dados e capaz de aprender e atualizar modelos de árvores ao receber novos dados. Já o modelo ARF tem por objetivo adaptar o modelo *Random Forest* para permitir que ele se adapte a fluxos de dados incrementais. Ao invés de treinar cada árvore com o conjunto completo de dados de treinamento, utiliza um conjunto em constante atualização de amostras recentes para treinar as novas árvores adicionadas à floresta. Os modelos foram treinados e testados seguindo as fases dispostas na Tabela 2.

5. Resultados e Discussão

Realizados os treinamentos e testes de todas as fases, as métricas foram coletadas e apresentadas graficamente para melhor compreensão dos resultados. O experimento realizado consistiu dos seguintes passos: na fase 1, os modelos foram treinados e testados com dados dos ataques DDoS, FTP e PortScan; na fase 2, o tráfego benigno (classe *Benign*) e os ataques DoS e SSH foram apresentados ao modelo e na fase 3, os dados dos ataques Botnet, Web (*Web Attack - Brute Force*) e XSS (*Web Attack - XSS*) foram apresentados ao modelo (ver Tabela 2).

A Figura 2 apresenta as métricas de precisão, revocação (*recall*) e F1-score para todas as classes (tráfego benigno e ataques), considerando o modelo GNB treinado com a biblioteca *Scikit-learn*. Observa-se que os gráficos de cima se referem às classes presentes na fase 1, os gráficos do meio são da fase 2 e os gráficos de baixo são da fase 3. Esses serão os resultados de benchmark, visto que o GNB treinado com a biblioteca *Scikit-learn* é um modelo convencional que não está completamente adaptado à técnica de IL.

É interessante observar, na Figura 2, o fenômeno do esquecimento catastrófico à medida que novas classes de tráfego foram apresentadas ao modelo. Vê-se que a detecção dos ataques DDoS, FTP e PortScan foram prejudicadas com a entrada das fases 2 e 3. Com exceção da revocação do DDoS, todas as outras métricas degradaram, em especial

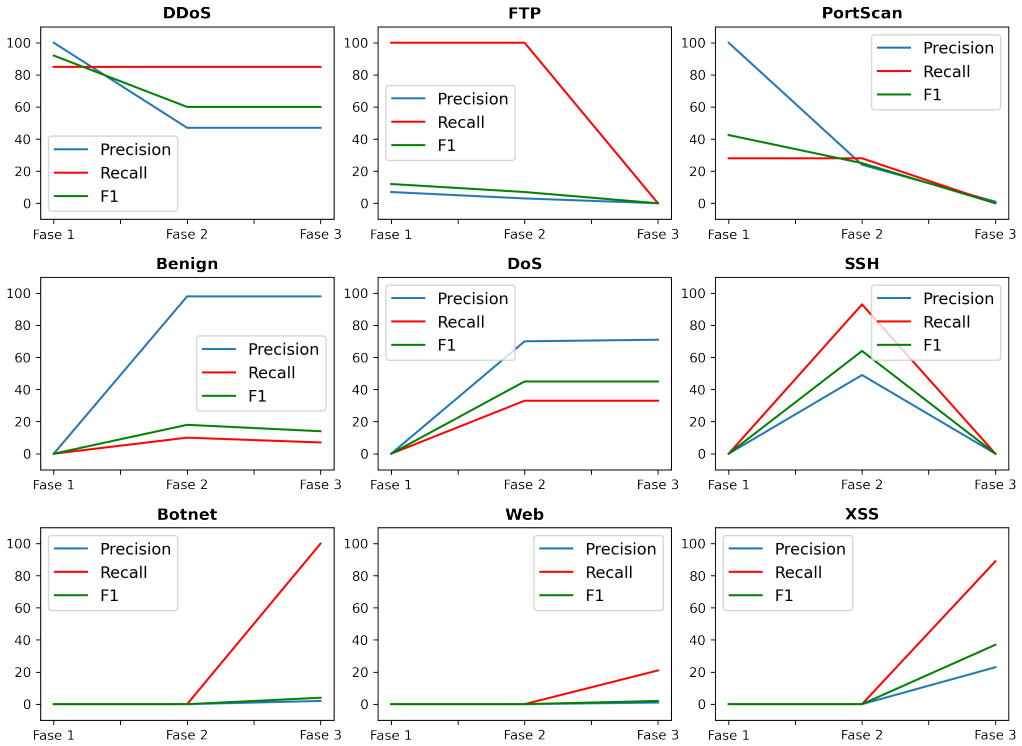

Figura 2. Desempenho das métricas no decorrer das fases com o modelo GNB usando a biblioteca *Scikit-learn*: Fase 1 (DDoS, FTP, PortScan), Fase 2 (Benign, DoS, SSH), Fase 3 (Botnet, Web, XSS).

as métricas relativas aos ataques FTP e PortScan, que chegaram a zero na fase 3, evidenciando o fato de que o modelo não conseguiu lidar adequadamente com o esquecimento das informações aprendidas na fase anterior, na medida que os novos dados foram introduzidos. Por outro lado, a detecção do tráfego benigno e do ataque DoS não sofreu muito com a entrada da fase 3, ao contrário da detecção do ataque SSH, que foi severamente prejudicada. Por fim, considerando as classes Botnet, Web e XSS, nota-se que somente a métrica de revocação aumentou de forma mais significativa quando essas classes foram apresentadas ao modelo na fase 3.

Quando o mesmo modelo GNB é treinado com a biblioteca *River*, a qual utiliza a abordagem de IL, o resultado é diferente, conforme mostrado na Figura 3. Observa-se que as detecções do tráfego benigno, bem como dos ataques FTP, PortScan e DoS melhoraram e as métricas de desempenho se mantiveram altas mesmo após as fases 2 e 3. Isso mostra que o treinamento incremental do *River* surtiu efeito. Em relação ao ataque DDoS, a métrica de revocação piorou, mas por outro lado, a precisão melhorou. No entanto, o algoritmo GNB não foi capaz de lidar bem com a detecção dos ataques SSH, Web e XSS, provavelmente por causa de limitações do modelo ou pouca quantidade de amostras no treinamento.

A fim de melhorar os resultados de classificação alcançados pelo GNB, foram utilizados modelos mais adequados à abordagem de IL. As figuras 4 e 5 mostram o desempenho dos classificadores HAT e ARF treinados com a biblioteca *River*, respectivamente. Nota-se que os modelos HAT e ARF lidam muito bem com o IL, enfrentando o esquecimento catastrófico e apresentando métricas de desempenho muito melhores que

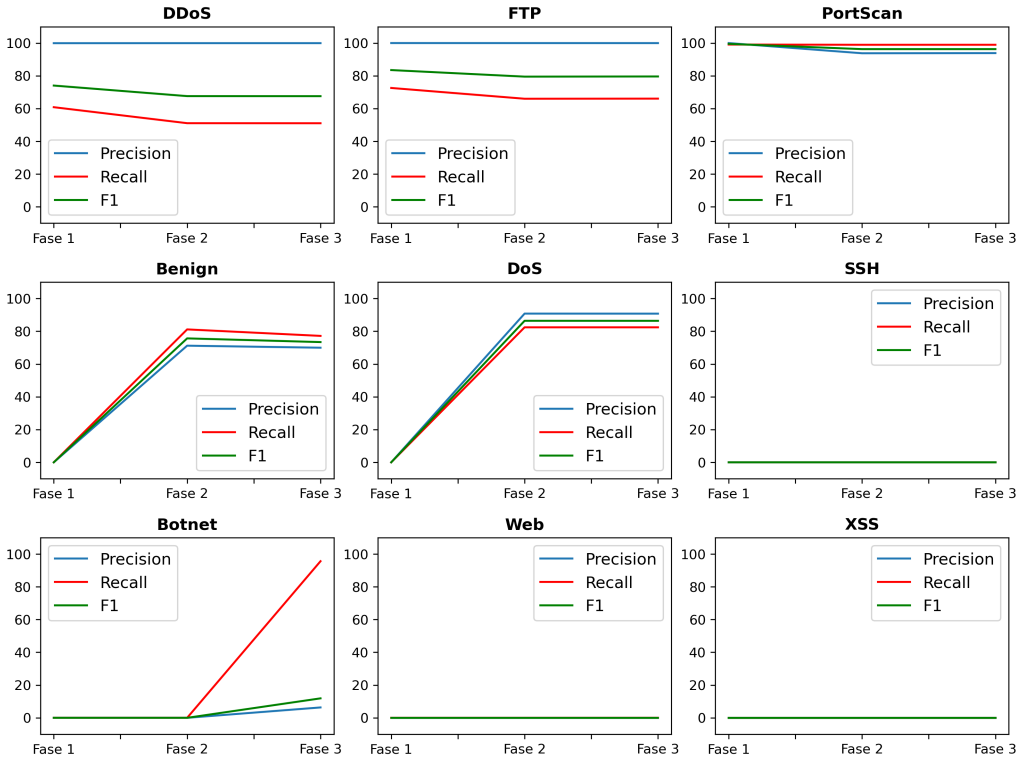

Figura 3. Desempenho das métricas no decorrer das fases com o modelo GNB usando a biblioteca *River*: Fase 1 (DDoS, FTP, PortScan), Fase 2 (Benign, DoS, SSH), Fase 3 (Botnet, Web, XSS).

o GNB. Essa capacidade de reter conhecimento prévio ao lidar com novas informações é uma característica valiosa desses modelos. O modelo ARF incremental se sobressaiu, principalmente, na detecção dos ataques FTP e SSH.

A Tabela 3 apresenta todas as métricas de desempenho de todos os algoritmos, para todas as classes, ao final da fase 3. Neste experimento, não foi usada nenhuma técnica de balanceamento de dados. Nota-se que os melhores valores foram destacados em negrito para cada um dos casos. É possível observar que o modelo GNB, treinado com o *Scikit-learn*, apresentou o pior desempenho, demonstrando que ele não foi capaz de lidar com o caráter incremental do experimento. Ao mudar o treinamento para a biblioteca *River*, os resultados melhoraram muito, sendo maximizados para os algoritmos HAT e ARF, que são projetados para trabalhar com fluxos de dados incrementais. Dos dois modelos incrementais, ARF foi o que apresentou as melhores taxas de detecção, atingindo uma acurácia geral de 99,61%. Dessa forma, o modelo ARF é um ótimo candidato para ser selecionado como classificador dos diferentes tipos de ataque, com auxílio do IL para otimizar o IDS.

Os resultados obtidos com a técnica de balanceamento de dados são apresentados na Tabela 4. Nota-se que, no geral, o algoritmo ARF treinado com *River* novamente obteve os melhores resultados. Embora tenha ocorrido um pequeno decréscimo na acurácia geral do modelo, as médias macro da precisão, revocação e *F1-score* aumentaram. Para o problema de detecção de intrusão, as métricas de precisão e revocação são especialmente importantes, pois elas indicam se o sistema IDS é capaz de minimizar os eventos de falsos positivos e falsos negativos. A métrica F1-score é uma média harmônica da precisão e do

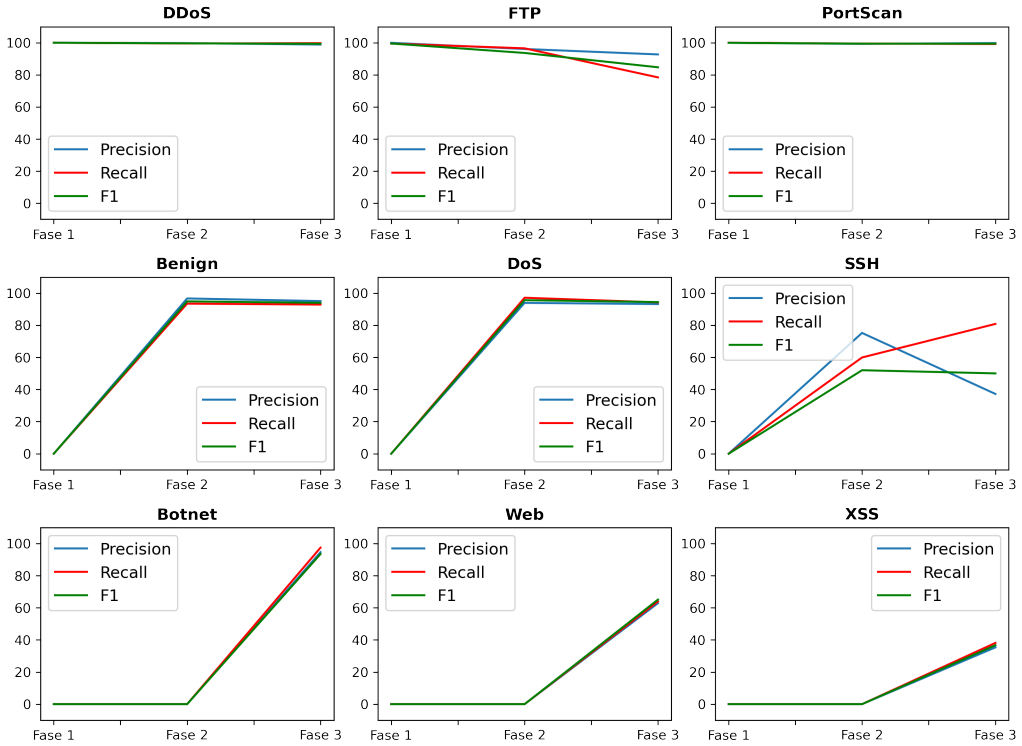

Figura 4. Desempenho das métricas no decorrer das fases com o modelo HAT usando a biblioteca *River*: Fase 1 (DDoS, FTP, PortScan), Fase 2 (Benign, DoS, SSH), Fase 3 (Botnet, Web, XSS).

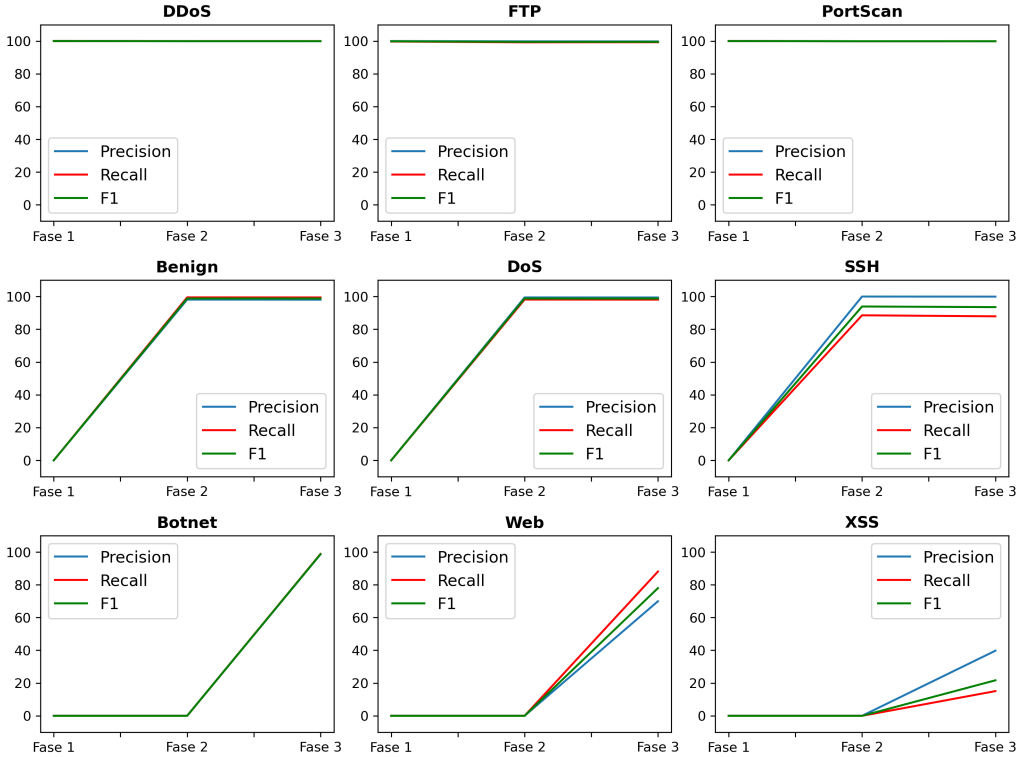

Figura 5. Desempenho das métricas no decorrer das fases com o modelo ARF usando a biblioteca *River*: Fase 1 (DDoS, FTP, PortScan), Fase 2 (Benign, DoS, SSH), Fase 3 (Botnet, Web, XSS).

revocação.

Com as métricas *F1-score* coletadas, utilizou-se o *Autorank*, um pacote Python com a tarefa de simplificar a comparação entre múltiplas populações (Herbold, 2020). Dessa forma, realizado o teste estatístico Bayesiano dos postos sinalizados, após o pacote assumir que duas populações (GNB River e GNB Sciki-learn) das métricas utilizadas não são normais (menor valor p observado de 0,006), foram calculados os valores médios das populações e relatada a mediana (MD) e o desvio absoluto mediano (MAD) para cada população. Os resultados obtidos, para cada caso foram: ARF River (MD=0.867+-0.007, MAD=0.002), HAT River (MD=0.698+-0.042, MAD=0.027), GNB River (MD=0.550+-0.005, MAD=0.000) e GNB Scikit-learn (MD=0.110+-0.005, MAD=0.000). Constatou-se que o modelo ARF possui a mediana da população superior aos demais modelos. Além disso, realizando o teste estatístico Bayesiano entre os modelos ARF balanceado e não balanceado, o pacote assumiu as métricas como normais (menor valor p observado de 0,036) e foram calculados os valores médios (M) e desvios padrões (SD) das métricas para cada modelo. Os resultados obtidos foram: ARF Balanceado (M=0.876+-0.012, SD=0.006) e ARF River (M=0.866+-0.010, SD=0.005). O modelo aplicado aos dados balanceados alcançou resultados estatisticamente superiores com relação ao valor médio das populações.

Tabela 3. Métricas finais para cada algoritmo com os dados originais

	GNB Scikit-learn			**GNB *River***			**HAT *River***			**ARF *River***		
	Precisão	**Revocação**	**F1-score**	**Precisão**	**Revocação**	**F1-score**	**Precisão**	**Revocação**	**F1-score**	**Precisão**	**Revocação**	**F1-score**
Benign	**100,00%**	7,00%	14,00%	95,77%	73,77%	83,19%	99,13%	99,80%	99,46%	99,69%	**99,94%**	**99,82%**
Botnet	0,00%	**100,00%**	1,00%	86,00%	97,51%	1,71%	52,84%	73,87%	60,45%	**95,03%**	97,73%	**96,34%**
DDoS	31,00%	85,00%	46,00%	**99,93%**	50,89%	67,45%	98,96%	95,49%	97,19%	99,87%	**99,91%**	**99,89%**
DoS	20,00%	34,00%	25,00%	89,42%	82,50%	85,79%	97,88%	91,29%	94,47%	**99,44%**	**96,84%**	**98,14%**
FTP	0,00%	5,00%	8,00%	**100,00%**	67,12%	79,23%	57,42%	61,39%	54,37%	99,60%	**98,43%**	**99,01%**
PortScan	0,00%	0,00%	0,00%	58,43%	99,06%	73,48%	97,57%	96,81%	97,18%	**99,90%**	**99,79%**	**99,85%**
SSH	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	96,75%	66,98%	78,86%	**99,77%**	**88,82%**	**93,92%**
Web Attack - Brute Force	0,00%	48,00%	1,00%	0,00%	0,00%	0,00%	14,65%	20,00%	15,54%	**68,80%**	**91,16%**	**78,17%**
Web Attack - XSS	4,00%	58,00%	8,00%	0,00%	0,00%	0,00%	34,39%	**56,53%**	**40,85%**	**41,61%**	8,11%	13,33%
Média Macro	17,00%	37,00%	11,00%	49,37%	52,14%	43,54%	72,13%	73,57%	70,93%	**88,93%**	**86,81%**	**86,73%**
Acurácia Geral		13,00%			74,52%			98,51%			**99,61%**	

Tabela 4. Métricas finais para cada algoritmo com os dados balanceados

	GNB Scikit-learn			**GNB *River***			**HAT *River***			**ARF *River***		
	Precisão	**Revocação**	**F1-score**	**Precisão**	**Revocação**	**F1-score**	**Precisão**	**Revocação**	**F1-score**	**Precisão**	**Revocação**	**F1-score**
Benign	98,00%	7,00%	14,00%	69,98%	77,20%	73,41%	95,08%	92,89%	93,98%	**98,08%**	**99,45%**	**98,76%**
Botnet	2,00%	**100,00%**	4,00%	6,34%	95,63%	11,89%	94,69%	97,51%	93,52%	**98,61%**	98,87%	**98,74%**
DDoS	47,00%	85,00%	60,00%	**99,94%**	51,04%	67,57%	98,86%	99,71%	99,32%	99,90%	**99,96%**	**99,93%**
DoS	71,00%	33,00%	45,00%	90,78%	82,44%	86,41%	93,21%	94,27%	94,45%	**99,44%**	**98,11%**	**98,77%**
FTP	0,00%	0,00%	0,00%	**99,97%**	66,07%	79,55%	92,71%	78,41%	84,70%	99,78%	**99,30%**	**99,54%**
PortScan	1,00%	0,00%	0,00%	93,86%	98,96%	96,34%	99,78%	99,18%	99,42%	**99,89%**	**99,92%**	**99,90%**
SSH	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	37,23%	80,85%	50,03%	**99,95%**	**87,93%**	**93,55%**
Web Attack - Brute Force	1,00%	21,00%	2,00%	0,00%	0,60%	0,00%	62,90%	63,95%	65,12%	**69,96%**	**88,11%**	**77,98%**
Web Attack - XSS	23,00%	89,00%	37,00%	0,00%	0,00%	0,00%	35,37%	**38,16%**	**36,71%**	**39,76%**	15,09%	21,68%
Média Macro	27,00%	37,00%	18,00%	51,21%	52,37%	46,13%	77,74%	81,89%	76,88%	**89,49%**	**87,42%**	**87,65%**
Acurácia Geral		31,00%			75,68%			95,04%			**98,89%**	

6. Conclusão

Tendo em vista o cenário atual, no qual as redes de computadores são essenciais para o funcionamento dos diferentes ambientes de trabalho, nota-se a necessidade de ferramentas que possam se adaptar à grande massa de dados gerada pelas instituições e que consigam

evitar que essas informações sejam perdidas ou roubadas. O IDS é uma das ferramentas que podem auxiliar nesse processo. Para tanto, sua associação a ferramentas de ML torna-se fundamental, tendo em vista o requisito de analisar e classificar padrões em larga escala.

Dessa maneira, os *pipelines* de ML convencionais não contam com a capacidade de adaptação necessária para os novos tipos de ataques que surjam com o tempo. Tal comportamento foi constatado empiricamente no resultado reportado na Figura 2, utilizando o modelo GNB convencional. Essa constatação enfatiza a importância de modelos que consigam aprender, à medida que recebam novos dados de novas classes.

A tarefa de IL pode enfrentar um fenômeno conhecido como esquecimento catastrófico, em que dados anteriores são esquecidos à medida que novos dados de novas classes são apresentados. Com o objetivo de combater esse problema, os classificadores GNB modificado, HAT e ARF foram avaliados com a biblioteca *River*. Os dois últimos modelos mostraram-se capazes de aprender novos padrões sem a necessidade de serem retreinados do início, mantendo um bom nível das métricas nas diversas classes, ao longo das múltiplas fases de treinamento. Seus desempenhos foram ilustrados nas Figuras 4 e 5. O melhor modelo foi o ARF com uma técnica de balanceamento de dados, que alcançou resultados de precisão, revocação e *F1-score* superiores a 87,42% e acurácia geral de 98,89%.

Após todos os experimentos e resultados obtidos, conclui-se que o IDS, aliado ao aprendizado incremental, pode ser aplicado aos problemas de classificação encontrados nos ambientes institucionais atuais. Tal metodologia permite que os modelos possam ser atualizados mais rapidamente, sem a necessidade de retreino, com uma grande massa de dados. Além disso, novas classes de ataque podem ser aprendidas sem o esquecimento de padrões anteriores. Como limitação do trabalho, aponta-se a aplicação dos modelos em apenas um conjunto de dados. Para futuros trabalhos, planeja-se incorporar o aprendizado profundo ao IL, visando verificar a possibilidade de se obter uma otimização sobre os resultados alcançados. Além disso, pretende-se criar um conjunto de dados com classes balanceadas, o que pode levar a métricas ainda mais consistentes.

7. Agradecimentos

Parte dos resultados apresentados neste trabalho foram obtidos através do projeto "RESIDÊNCIA EM SEGURANÇA DA INFORMAÇÃO", executado pela UFC, em parceria com o SiDi e financiado pela Samsung Eletrônica da Amazônia Ltda., no âmbito da Lei de Informática no. 8.248/91.

Referências

Shaza M Abd Elrahman and Ajith Abraham. A review of class imbalance problem. *Journal of Network and Innovative Computing*, 1(2013):332–340, 2013.

Uttam Adhikari and et. al Morris. Adaptive trees for real-time cyber-power event and intrusion classification. Disponível em: `https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.trees.HoeffdingAdaptiveTreeClassifier.html`. Acesso em: 09 mar. 2023., 2019.

Suresh Kumar Amalapuram, Akash Tadwai, Reethu Vinta, Sumohana S Channappayya, and Bheemarjuna Reddy Tamma. Continual learning for anomaly based network intrusion detection. pages 497–505, 2022.

Masayoshi Data, Mahendra ; Aritsugi. T-dfnn: An incremental learning algorithm for intrusion detection systems. *IEEE Access*, 9:154156–154171, 2021.

Heitor M Gomes and et. al Bifet. Adaptive random forests for evolving data stream classification. Disponível em: `https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.meta.AdaptiveRandomForestClassifier.html`. Acesso em: 09 mar. 2023., 2017.

Max Halford, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, and Adil Zouitine. creme, a Python library for online machine learning, 2019. URL `https://github.com/MaxHalford/creme`.

Steffen Herbold. Autorank: A python package for automated ranking of classifiers. *Journal of Open Source Software*, 5(48):2173, 2020. doi: 10.21105/joss.02173. URL `https://doi.org/10.21105/joss.02173`.

Pedro Horchulhack, Eduardo K Viegas, Altair O Santin, and Jhonatan Geremias. Atualizaçao de modelo baseado em aumento de dados e transferência de aprendizagem para detecçao de intrusao em redes. In *Anais do XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 223–235. SBC, 2022.

Lipika ; Elizondo David Kenyon, Anthony ; Deka. Are public intrusion datasets fit for purpose characterising the state of the art in intrusion event datasets. *Computers & Security*, 99:102022, 2020.

Farah Barika Louati, Faten ; Ktata. A deep learning-based multi-agent system for intrusion detection. *SN Applied Sciences*, 2(4):675, 2020.

Ali Mahdavi, Ehsan; Fanian and Zahra Mirzaei, Abdolreza ; Taghiyarrenani. Itl-ids: Incremental transfer learning for intrusion detection systems. *Knowledge-Based Systems*, 253:109542, 2022.

Tom Michael Mitchell et al. *Machine learning*, volume 1. McGraw-hill New York, 2007.

Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*, 19(72): 1–5, 2018. URL `http://jmlr.org/papers/v19/18-251.html`.

Jacob Montiel, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, Adil Zouitine, Heitor Murilo Gomes, Jesse Read, Talel Abdessalem, et al. River: machine learning for streaming data in python. 2021.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Biju Shah, Syed Ali Raza ; Issac. Performance comparison of intrusion detection systems and application of machine learning to snort system. *Future Generation Computer Systems*, 80:157–170, 2018.

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

Dimitra ; Giannoutakis Konstantinos M ; Drosou Anastasios ; Tzovaras Dimitrios Toupas, Petros ; Chamou. An intrusion detection system for multi-class classification based on deep neural networks. pages 1253–1258, 2019.