DISSERTAÇÃO DE MESTRADO PROFISSIONAL

WEAPON: Uma Arquitetura de Aprendizado Não Supervisionado para Detecção de Anomalias de Comportamento de Usuários

André Luiz Bandeira Molina

Programa de Pós-Graduação Profissional em Engenharia Elétrica
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

WEAPON: AN UNSUPERVISED LEARNING ARCHITECTURE FOR USER BEHAVIOR ANOMALY DETECTION

WEAPON: UMA ARQUITETURA DE APRENDIZADO NÃO SUPERVISIONADO PARA DETECÇÃO DE ANOMALIAS DE COMPORTAMENTO DE USUÁRIOS

ANDRÉ LUIZ BANDEIRA MOLINA

ORIENTADOR: GERALDO PEREIRA ROCHA FILHO, Ph.D
COORIENTADOR: VINÍCIUS PEREIRA GONÇALVES, Ph.D

DISSERTAÇÃO DE MESTRADO PROFISSIONAL EM ENGENHARIA ELÉTRICA

PUBLICAÇÃO: PPEE.MP.020
BRASÍLIA/DF, SETEMBRO - 2022

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

DISSERTAÇÃO DE MESTRADO PROFISSIONAL

WEAPON: Uma Arquitetura de Aprendizado Não Supervisionado para Detecção de Anomalias de Comportamento de Usuários

André Luiz Bandeira Molina

Dissertação de Mestrado Profissional submetida ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica

Banca Examinadora

Prof. Geraldo Pereira Rocha Filho, Ph.D, IE/UnB _____
Orientador

Prof. Demétrio Antônio da Silva Filho, Ph.D, IF/UnB _____
Examinador Interno

Lauro César Araujo, Ph.D, Senado Federal _____
Examinador Externo

FICHA CATALOGRÁFICA

MOLINA, ANDRÉ LUIZ BANDEIRA
WEAPON: Uma Arquitetura de Aprendizado Não Supervisionadopara Detecção de Anomalias de Comportamento de Usuários [Distrito Federal] 2022.
xvi, 107 p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2022).
Dissertação de Mestrado Profissional - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Anomalias de Comportamento de Usuários
2. Detecção de Anomalias
3. Segurança Cibernética
4. Autoencoder
I. ENE/FT/UnB
II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

MOLINA, A. L. B. (2022). *WEAPON: Uma Arquitetura de Aprendizado Não Supervisionadopara Detecção de Anomalias de Comportamento de Usuários*. Dissertação de Mestrado Profissional, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 107 p.

André Luiz Bandeira Molina
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

DEDICATÓRIA

Dedico este trabalho às minhas amadas filhas, Natália e Yasmin.

Que esse triunfo as inspire em suas vidas, na certeza de que todo esforço em busca de algo bom para a humanidade e voltado para o bem nunca terá sido em vão.

AGRADECIMENTOS

Toda conquista merece uma profunda reflexão acerca daqueles que apoiaram seu alcance. Primeiramente, agradeço aos meus pais por terem me ensinado o quão valioso é o conhecimento. Agradeço ao meu orientador, professor Geraldo Pereira Rocha Filho, e ao meu coorientador, professor Vinícius Pereira Gonçalves, que sempre me apoiaram a manter o foco e a suscitar a capacidade de pesquisa. Agradeço também à minha família, que me apoiou a enfrentar esse desafio. Não poderia de deixar de agradecer ainda aos meus colegas de trabalho e ao Senado Federal, que certamente contribuíram para a minha dedicação a esta pesquisa. Por fim, agradeço a Deus por despertar em mim uma imensa curiosidade e inquietude na busca pelo conhecimento.

RESUMO

Nos últimos anos, a detecção de anomalias de comportamento do usuário tem conquistado a atenção em segurança cibernética. Um dos desafios cruciais que têm sido pesquisados na literatura é que modelos supervisionados que utilizam grandes quantidades de dados para treinamento não se aplicam a cenários reais de detecção de anomalias. Em contrapartida, modelos não supervisionados tendem a apresentar problemas de escalabilidade com relação ao número de usuários do *dataset*, uma vez que abordam os aspectos comportamentais de forma individual para cada usuário. Dentro deste contexto, a necessidade de obter *datasets* rotulados com anomalias de comportamento provou ser um fator limitante para avaliar diferentes modelos, e esta limitação é explorada nesta pesquisa. Este trabalho apresenta uma arquitetura para a detecção de anomalias de comportamento de usuários baseada *Wide and Deep Convolutional LSTM Autoencoders* – WEAPON. O WEAPON utiliza aprendizado não supervisionado e requer uma pequena quantidade de dados para construir perfis de comportamento considerando a individualidade de cada usuário. Além disso, o WEAPON implementa uma abordagem de supervisão fraca para rotulação de anomalias de comportamento a partir do Snorkel. Quando comparado com outras abordagens, o WEAPON provou ser mais eficiente superando o segundo melhor modelo em mais de 4% na curva ROC. Além disso, WEAPON supera os métodos baseados em regras ao encontrar anomalias que um especialista não anteciparia.

ABSTRACT

In recent years, user behavior anomaly detection has been gaining attention in cybersecurity. A crucial challenge that has been discussed in the literature is that supervised models that use vast amounts of data for training do not apply to real scenarios for anomaly detection. In contrast, unsupervised models tend to face scalability problems with respect to the number of users in the dataset, since they address behavioral aspects on an individual basis for each user. Within this context, the requirement to gather datasets with labeled behavior anomalies has proven to be a significant limiting factor for evaluating different models, and this limitation is explored in this research. This work presents an architecture for user behavior anomaly detection based on Wide and Deep Convolutional LSTM Autoencoders – WEAPON. WEAPON uses unsupervised learning and requires a small amount of data to build behavior profiles considering the individuality of each user. Furthermore, WEAPON implements weak supervision-based behavior anomaly labeling approach using Snorkel. When compared to other approaches, WEAPON proved to be more efficient, surpassing the ROC curve of the second best model by more than 4%. Furthermore, WEAPON outperforms rule-based methods by finding anomalies that an expert would not anticipate.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Siglas

1D	Unidimensional
2D	Bidimensional
DNN	*Deep Neural Network* (Rede Neural Profunda)
CERT	*Computer Emergency Response Team*
CM	*Confusion Matrix* (Matriz de Confusão)
CNN	*Convolutional Neural Networks* (Redes Neurais Convolucionais)
ELU	*Exponencial Linear Unit*
FN	*False Negative* (Falso Negativo)
FP	*False Positive* (Falso Positivo)
FPR	*False Positive Rate* (Taxa de Falso Positivo)
IA	Inteligência Artificial
LSTM	*Long Short-Term Memory*
GPU	*Graphics Processing Unit* (Unidade de Processamento Gráfico)
GRU	*Gated Recurrent Unit*
mae	*Mean Absolute Error* (Erro Médio Absoluto)
MLP	*Multilayer Perceptron* (Perceptron Multicamadas)
ns	Nanossegundos
OCSVM	*One-Class Support Vector Machine*
OCSVM-SGD	*One-Class Support Vector Machine using Stochastic Gradient Descent*
OLAP	*Online Analytical Processing*
RPCA	*Robust Principal Component Analysis*
RNA	Rede Neural Artificial
RNN	*Recurrent Neural Network* (Rede Neural Recorrente)
ROC	*Receiver Operating Characteristic* (Característica Operacional do Receptor)
ROC-AUC	*Area Under the Curve Receiver Operating Characteristic* (Área Sob a Curva Característica Operacional do Receptor)
SVM	*Support Vector Machine*
SAEBN	*Stacked Autoencoder with Batch Normalization*
TF-IDF	*Term Frequency and Inverse Document Frequency*
TLU	*Threshold Logic Unit* (Unidade de Limiar Lógico)
TN	*True Negative* (Verdadeiro Negativo)
TNR	*True Negative Rate* (Taxa de Verdadeiro Negativo)
TP	*True Positive* (Verdadeiro Positivo)
TPR	*True Positive Rate* (Taxa de Verdadeiro Positivo)
WDSAEBN	*Wide and Deep Stacked Autoencoder with Batch Normalization*
WEAPON	Wide and Deep Convolutional LSTM Autoencoder

Subscritos

max	Máximo
min	Mínimo
$scaled$	Escalonado

Sobrescritos

^{T}	Transposta

1 INTRODUÇÃO

O advento da pandemia do novo Coronavírus consolidou definitivamente o uso de sistemas distribuídos pela sociedade moderna [5, 6, 7]. Onde antes existia certa resistência na utilização de formas trabalho remoto, a nova realidade imposta obrigou não apenas as organizações, mas a sociedade como um todo, a disponibilizarem e inovarem o trabalho e as interações sociais com inúmeras ferramentas para colaboração [5, 8], boa parte sem um estudo preliminar de levantamento dos riscos [5, 9]. Como exemplo, muitos serviços em nuvem foram disponibilizados sem autenticação forte (multifator), trazendo como risco a maior exposição desses serviços a furtos de credenciais [10, 11].

Essa corrida das organizações para se adaptarem a esse novo cenário trouxe consigo um aumento expressivo dos ataques cibernéticos. O uso intenso de serviços de nuvem, acesso remoto com *Virtual Private Network* – VPN e utilização de dispositivos domésticos sem proteções adequadas, entre outros fatores, aumentou significativamente a superfície de ataque aos sistemas de informação utilizados. O resultado foi um aumento significativo de ataques em escala global [12].

Recentemente, o Brasil sofreu ataques de ransomware[1] em órgãos governamentais e empresas privadas, mega vazamento de dados de cerca de 220 milhões de brasileiros, eventos de indisponibilidades de serviços críticos à sociedade, entre outros impactos [5, 13]. Os prejuízos gerados por esses ataques são incalculáveis, pois afetam não só a infraestrutura de tecnologia da instituição, mas também podem implicar em danos ao patrimônio intangível da empresa, como a confiança dos clientes na marca [14].

Os ataques cibernéticos possuem diversas motivações, tais como a insatisfação de um funcionário, a obtenção de vantagens financeiras e até mesmo o terrorismo contra um Estado-Nação, indisponibilizando infraestruturas críticas nacionais, tais como sistemas de energia, telecomunicações e finanças [15]. Desse modo, a segurança cibernética tornou-se um dos principais assuntos da agenda das universidades, empresas e líderes de governo em todo o mundo, reservando recursos financeiros e humanos para o avanço nessa área, particularmente no que se refere a detectar anomalias que estejam correlacionadas com os ataques [3, 16].

Grande parte dos ataques de sucesso apresentam como características algum desvio de comportamento do usuário [17, 18], de modo que quando tem início uma ação maliciosa, o perfil de comportamento é alterado. Diante dos inúmeros vazamentos de dados pessoais, bem como ataques de *phishing* ou engenharia social, é previsível que mudanças de comportamento de usuário evidenciem uma ação maliciosa. Essa mudança de comportamento é considerada uma anomalia.

Anomalias são tipos de *outliers*, sendo definidas como instâncias de dado que se desviam significativamente das observações sobre a maioria dos dados, de modo a possibilitar a suspeita de que foram geradas por um mecanismo diferente daquele que gerou a maioria dos dados considerados normais [3, 19]. Porém, diferentemente dos *outliers*, as anomalias não podem ser caracterizadas como ruído. Assim, anomalias geralmente contêm informações importantes sobre as características anormais dos sistemas ou entidades

[1] Ransomware é um tipo de código malicioso que, geralmente usando criptografia, torna inacessíveis os dados armazenados em um equipamento e então exige pagamento de resgate (*ransom*) para restabelecer o acesso do usuário. Fonte: Cartilha de Segurança para Internet - https://cartilha.cert.br/.

que afetam o mecanismo de geração dos dados, sendo objeto de interesse para a análise.

Por sua vez, a detecção de anomalias é o processo de identificação de dados que se desviam significativamente dos dados considerados normais [3, 19, 20]. Dada sua abrangência e versatilidade, a detecção de anomalias tem sido pesquisada em diversas áreas e domínios de aplicação [3, 16], tais como análise de dados multidimensionais [20, 21], detecção de intrusão ou detecção de ataques [17, 22, 23, 24, 25, 26], séries temporais para identificar padrões inesperados [27, 28, 29, 30], e comportamento de usuários [17, 19, 22, 25, 31]. Grande parte desse sucesso decorre dos avanços em capacidade computacional com tecnologias em nuvem e computação distribuída, uma vez que permitiram a aplicação de diversas técnicas baseadas em aprendizado de máquinas, o que se verifica nos trabalhos [3, 19, 20, 32, 33, 34, 35, 36], que apresentam uma extensa visão geral acerca das técnicas de detecção de anomalias.

A detecção de anomalias de comportamento de usuário vem sendo objeto de atenção no campo da segurança cibernética [19, 37]. Nesse sentido, um usuário apresenta comportamento anômalo quando é possível estabelecer que ele está agindo de maneira incomum com base em um padrão de comportamento pré-estabelecido [17]. Essa abordagem pode permitir que contramedidas sejam iniciadas antes mesmo que um ataque seja bem-sucedido.

Contudo, ferramentas de segurança tradicionais são inadequadas para detectar anomalias de comportamento de usuário, pois são incapazes de endereçar contextos comportamentais. O *Security Information and Event Management* – SIEM, por exemplo, requer a configuração de regras previamente estabelecidas para a obtenção de um alerta, o que se mostra ineficaz diante do grande número de possibilidades de anomalias comportamentais. Ademais, o alto volume de *logs* de segurança gerados a partir de múltiplas origens requer mecanismos analíticos semelhantes às soluções de *Business Inteligence* [20, 21]. Isso tem levado a um aumento significativo de trabalhos endereçando detecção de anomalias de comportamento de usuários por meio de técnicas de *big data* e aprendizado de máquinas [17, 19, 20, 32, 21, 24, 25, 31, 37, 38, 39].

Apesar desses avanços, a detecção de anomalias de comportamento de usuário ainda não é uma tarefa trivial e ainda há diversos problemas que esta pesquisa investiga:

a) O primeiro problema é que parte dos trabalhos propõem modelos que utilizam treinamento supervisionado para detectar as anomalias. Isso não é apropriado, uma vez que detecção de anomalias é um típico problema de detecção de *outliers*, cuja natureza é principalmente não supervisionada. Nesse caso, os exemplos de anomalias geralmente não estão disponíveis para encontrar o melhor modelo [3]. Além disso, esses cenários de estudo utilizam a quase totalidade dos dados para o treinamento supervisionado, o que se traduz em um horizonte temporal muito longo. Isso não é consistente com ambientes reais, nos quais é necessário estabelecer um perfil de comportamento de usuário com brevidade para que a detecção de anomalias ocorra e possibilite a adoção de contramedidas rápidas;

b) Outro problema que se apresenta é a dificuldade de obtenção de *datasets* rotulados com anomalias de comportamento para a construção e validação de modelos. Os *datasets* públicos com anomalias rotuladas são, em geral, específicos a um tipo específico de problema, rotulando uma pequena quantidade de anomalias que não consideram situações de alterações comportamentais que um analista de segurança certamente solicitaria providências. Além disso, muitos *datasets* utilizados em pesquisas são privados e não podem ser compartilhados por questões de proteção de dados e privacidade, o que inviabiliza a

reprodução e validação de modelos da literatura.

c) Um último problema que se apresenta é referente à individualização dos usuários quanto ao estabelecimento do padrão comportamental e à detecção de anomalias. Modelos que individualizam os usuários tendem a ter problemas de escalabilidade quando o número de usuários cresce. Por outro lado, modelos que não individualizam os usuários caracterizam o perfil comportamental e detectam anomalias somente em função da totalidade de usuários, o que implica em não detectar alterações específicas de um usuário.

A partir desses problemas, observa-se que há espaço para melhorias. Mais especificamente, em relação a investigar uma arquitetura não supervisionada, utilizando para validação um *dataset* rotulado tal como se um especialista houvesse analisado as anomalias, que possibilite a construção de perfis comportamentais e detecção de anomalias de comportamento de usuários de forma abrangente e em horizonte temporal factível a um cenário real.

1.1 JUSTIFICATIVA

Com a escalada dos ataques cibernéticos, tornou-se imprescindível a adoção de inúmeros mecanismos segurança para minimizar as chances de um incidente. Por exemplo, os casos de ransomware aumentaram 90% de 2020 para 2021 [40], com o Brasil figurando como o quinto país no mundo com maior incidência desse tipo de ataque [5]. No setor de finanças, os ataques cibernéticos tiveram um aumento de 300% [41].

Isso implica necessariamente em vultosos investimentos em ferramentas de segurança e contratação de recursos humanos altamente capacitados para lidar com as ameaças complexas. Contudo, no que tange à Administração Pública Brasileira, os investimentos e contratação de pessoal não puderam avançar como esperado. Assistimos entre 2020 e 2022 a inúmeros incidentes de segurança em várias instituições públicas renomadas, tais como Superior Tribunal de Justiça - STJ, Tribunal Superior Eleitoral - TSE, Senado Federal, Ministério da Saúde e Ministério da Economia. Cada incidente com sua peculiaridade, mas todos expondo de forma contundente fragilidades em segurança cibernética no setor público.

Assim, é fundamental dotar a Administração Pública Brasileira de um arcabouço moderno e eficiente para detecção de situações anômalas. Nesse contexto, detectar anomalias de comportamento de usuários pode representar um grande avanço no conjunto de medidas possíveis para as organizações públicas, na medida em que isso pode reduzir a possibilidade de sucesso de ataques cibernéticos que possuem como caracaterística a mudança significativa de comportamento de usuários. Além disso, tal proposta vai ao encontro do objetivo central do Mestrado Profissional em Segurança Cibernética decorrente do Convênio ABIN/UnB – TED ABIN 08/2019.

1.2 OBJETIVO

Este trabalho tem como objetivo propor uma arquitetura para detectar anomalias de comportamento de usuário baseada em *Wide and Deep Convolutional LSTM Autoencoders* – WEAPON. O WEAPON é um modelo não supervisionado adequado para aplicação em cenários reais, pois requer menos dados para estabelecer padrões de comportamento e detectar anomalias quando comparado a outras abordagens da literatura. O WEAPON implementa uma abordagem de supervisão fraca para rotulação de anomalias de comportamento a partir do Snorkel [42], de modo a simular a análise de um especialista na rotulação de anomalias de comportamento. Além disso, o WEAPON é capaz de individualizar usuários e detectar anomalias de comportamento abrangentes, complementando mecanismos tradicionais baseados em regras.

1.3 RESULTADOS ALCANÇADOS

Como prova de conceito, foi utilizado o CERT *Dataset* R6.2, que é um *dataset* sintético aberto contendo registros de eventos de 4000 usuários com diferentes perfis de comportamento. Por se tratar de um *dataset* voltado para um pequeno subconjunto de anomalias específico, foi desenvolvida uma estratégia de rotulação de anomalias de comportamento de usuário com base em supervisão fraca utilizando o Snorkel. Quando comparado a outras abordagens, o WEAPON alcançou os seguintes resultados:

a) Alcançou um desempenho de Curva ROC da ordem de 4% superior ao segundo melhor modelo;

b) Identificou anomalias de comportamento de usuário complexas e não previsíveis por métodos analíticos baseados em regras;

c) Reduziu significativamente a quantidade de dados necessários para estabelecer padrões de comportamento em relação aos métodos supervisionados; e

d) Disponibilizou uma metodologia de rotulação de anomalias de comportamento de usuários que permite a reprodução e comparação de estudos futuros.

1.4 PUBLICAÇÕES RESULTANTES DESTA PESQUISA

Os principais resultados obtidos deste trabalho foram publicados nos Anais do XI Brazilian Workshop on Social Network Analysis and Mining – BraSNAM 2022 [43], Qualis B3:

- MOLINA, Andre L B et al. WEAPON: Uma Arquitetura para Detecção de Anomalias de Comportamento do Usuário. In: Anais do XI Brazilian Workshop on Social Network Analysis and Mining. SBC, 2022. p. 121-132.

1.5 ESTRUTURA DA DISSERTAÇÃO

O restante desta dissertação está organizado da seguinte forma. O Capítulo 2 apresenta a fundamentação teórica relacionada à detecção de anomalias de comportamento utilizando aprendizado de máquinas. O Capítulo 3 apresenta os trabalhos relacionados à detecção de anomalias de comportamento e suas principais limitações que são objeto de pesquisa nessa dissertação. O Capítulo 4 apresenta a visão geral da estratégia de detecção de anomalias de comportamento, o CERT *Dataset*, a rotulação de anomalias de comportamento de usuários com base em supervisão fraca, e o WEAPON. O Capítulo 5 apresenta a avaliação do WEAPON em relação a outras abordagens de aprendizado de máquinas. Por fim, o Capítulo 6 apresenta as conclusões da pesquisa e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Para uma melhor compreensão da arquitetura de detecção de anomalias de comportamento proposta neste trabalho, faz-se necessário apresentar uma contextualização do arcabouço teórico que serviu de fundamento para o alcance dos objetivos. Nesse sentido, esta pesquisa fundamenta-se na aplicação do aprendizado de máquinas – AM à detecção de anomalias.

Portanto, dentro desse escopo, este capítulo apresenta uma introdução dos conceitos relacionados ao AM, à detecção de anomalias, e à aplicação do AM na detecção de anomalias. Por fim, são apresentadas as principais métricas para avaliação de desempenho dos modelos voltados à detecção de anomalias.

2.1 APRENDIZADO DE MÁQUINAS

Nos últimos anos, o AM, o *deep learning* e a inteligência artificial – IA têm sido destaque em inúmeras publicações, permitindo a visualização de um futuro promissor em inúmeras frentes, tais como automóveis e taxis aéreos totalmente autônomos e diagnósticos médicos inequívocos [3, 16]. Os muitos avanços no AM, associados a contribuições de outras subáreas de IA, como o processamento de linguagem natural, visão computacional e robótica, possibilitaram que tarefas até então desafiadoras computacionalmente se tornassem simples e aplicáveis em diferentes setores [44].

A IA consiste nos esforços de automatizar tarefas intelectuais normalmente executadas por humanos [1]. Assim, a IA é um campo amplo que engloba o AM e o *deep learning*, mas que também inclui muitas outras abordagens que não envolvem nenhum aprendizado.

O AM consiste em programar computadores para otimizar o critério de desempenho utilizando dados ou experiência passados [4]. Assim, seu objetivo é desenvolver técnicas computacionais sobre o aprendizado bem como construir sistemas capazes de adquirir conhecimento de forma automática. As decisões são baseadas em experiências acumuladas através da solução bem-sucedida de problemas anteriores. Nesse sentido, os dados são referentes a eventos passados, constituindo-se a principal fonte de entrada para que os algoritmos de AM sejam eficientes e produzam resultados relevantes.

Assim, um modelo baseado em AM é treinado tendo como base uma série de exemplos relevantes para sua tarefa, e então ele busca uma representação estatística que eventualmente permita ao sistema responder de forma automática às tarefas. Um modelo baseado em AM requer [1]:

a) Conjunto de dados de entrada: são as informações que serão fornecidas para o sistema de AM;

b) Exemplos das saídas esperadas: são os resultados esperados da execução do sistema de AM; e

c) Uma métrica de êxito para medir se o algoritmo está realizando um bom trabalho, o que é necessário para determinar o quão distante está a resposta obtida do valor esperado.

Os algoritmos de AM podem ser divididos em dois tipos básicos de tarefas [44]: Preditivas e Des-

critivas. Em tarefas preditivas, a partir do treinamento dos algoritmos de AM sobre dados rotulados, é esperado que o modelo seja capaz de predizer valores alvo a partir de valores de seus atributos preditivos. Como exemplo, um modelo pode predizer o estado de saúde de um paciente a partir de seus sintomas. Os modelos preditivos são organizados em métodos baseados em distâncias, métodos probabilísticos, métodos simbólicos, métodos conexionistas e métodos baseados em maximização de margens.

Já nas tarefas descritivas, os algoritmos de AM extraem padrões ou representações dos valores preditivos de um conjunto de dados, normalmente não dependendo de nenhum rótulo nos dados de entrada. Como exemplo de tarefa descritiva temos o agrupamento de dados, que procura grupos de objetos similares entre si no conjunto de dados. Os modelos descritivos são organizados em sumarização, associação e agrupamento.

No contexto desta pesquisa, utilizaram-se as tarefas preditivas com os seguintes métodos [44]:

a) Métodos simbólicos: utilizam estruturas simbólicas para uma interpretação mais direta por seres humanos. Utilizados, por exemplo, no algoritmo de AM *Isolation Forest* – IF;

b) Maximização de margens: a partir de um procedimento de maximização de margens, buscam separar os exemplos pertencentes às diferentes classes. As margens são valores que determinam o grau de separação dos dados de diferentes classes. Como exemplo desse tipo de algoritmo, o *Support Vector Machine* – SVM aplica esse conceito; e

c) Métodos conexionistas: buscam simular o processo de aprendizado semelhante ao do cérebro humano. Como exemplo, temos as redes neurais artificiais – RNAs.

Para que um modelo de AM possa aprender a executar uma tarefa, seja preditiva, descritiva, ou outra que não se enquadre nesses dois tipos, tipicamente há quatro categorias de aprendizado [2]:

a) Aprendizado supervisionado;

b) Aprendizado não supervisionado;

c) Aprendizado semi-supervisionado; e

d) Aprendizado por reforço.

No aprendizado supervisionado, o objetivo é que o modelo de AM aprenda o mapeamento da entrada para uma saída desejada de acordo com um conjunto de dados disponibilizados por um supervisor. Nesse tipo de aprendizado dizemos que o conjunto de dados de treinamento é rotulado, ou seja, possui os valores (rótulos) de saída desejados. Exemplos de aplicações de modelos de AM que utilizam o aprendizado supervisionado são os sistemas de *Optical Character Recognition* – OCR, de classificação de imagens, de tradução de línguas e de reconhecimento da fala [1]. Alguns exemplos de algoritmos de AM baseados em aprendizado supervisionado são: *K-Nearest Neighbors*; Regressão Linear; Regressão Logística; SVM; *Decision Trees* e *Random Forests*.

Já o aprendizado não supervisionado consiste em fazer o modelo de AM encontrar transformações relevantes dos dados de entrada sem a ajuda de saídas desejadas. Nesse caso, não há rótulos nos dados de

treinamento. Exemplos de aplicações de modelos de AM que utilizam o aprendizado não supervisionado são detecção de anomalias, detecção de novidade, compressão de dados, redução de dimensionalidade, clusterização, remoção de ruído e visualização de dados. Os algoritmos de aprendizado não supervisionado têm sido largamente utilizados em *data analytics* para melhor compreensão dos dados antes de sua utilização. Alguns exemplos típicos de algoritmos de AM baseados em aprendizado não supervisionado são: K-Means; DBSCAN; *One-Class* SVM – OCSVM; IF; *Principal Component Analysis* – PCA; e Kernel PCA.

O aprendizado semi-supervisionado é semelhante ao supervisionado, mas a principal diferença é que uma pequena quantidade de dados possui a saída desejada (rótulos) em detrimento da grande maioria dos dados que não possuem rótulos. Como exemplo de aplicação desse método de aprendizado, o *Google Photos* [1] utiliza essa técnica para classificação dos rostos presentes nas fotos carregadas. A maioria dos algoritmos de aprendizado semi-supervisionado são combinações de algoritmos não supervisionados com supervisionados.

O aprendizado por reforço é voltado para modelos de AM cuja saída consiste em uma sequência de ações, de modo que uma única ação não é tão importante quanto o conjunto de saídas como um todo. Assim, o que importa é a sequência de ações corretas para alcançar o objetivo [4]. Um agente recebe informação sobre o ambiente e aprende a escolher as ações que irão maximizar alguma recompensa. Caso erre, o agente recebe uma penalidade. Do contrário, recebe a recompensa. O agente deve aprender por si só qual a melhor estratégia, denominada política. Normalmente os robôs utilizam o aprendizado por reforço para aprenderem a andar.

Após a etapa de aprendizado, o modelo de AM pode incorrer em dois tipos de problemas:

a) *Overfitting*, que é quando o modelo apresenta um bom desempenho para o conjunto de treinamento, mas um desempenho ruim para os casos fora desse conjunto. Nessa situação, também podemos dizer que o modelo não consegue generalizar a partir de seu aprendizado. Isso geralmente acontece quando o modelo é muito complexo em relação aos dados de treinamento; e

b) *Underfitting*, que ocorre quando o modelo não apresenta bom desempenho tanto para o conjunto de treinamento como para os casos fora desse conjunto. Ocorre geralmente quando o modelo é muito simples em relação à complexidade dos dados de treinamento.

2.1.1 *Shallow Learning* e *Deep Learning*

Deep Learning é um subconjunto do AM que se baseia em aprender a partir de camadas sucessivas de representações cada vez mais significativas dos dados. Modelos mais recentes de *deep learning* chegam a ter centenas de camadas representativas, que aprendem automaticamente a partir dos dados de treinamento. Por outro lado, outras abordagens de AM tendem a focar na aprendizagem de apenas uma ou duas camadas de representações dos dados, o que as faz serem conhecidas como *shallow learning*. A Figura 2.1 apresenta a relação entre IA, AM e *deep learning*.

Via de regra, *deep learning* é implementado a partir de modelos conhecidos como RNAs estruturados

[1] https://photos.google.com/

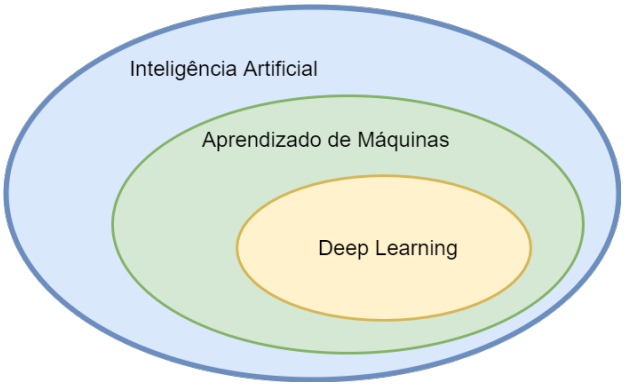


Figura 2.1: Relacionamento entre inteligência artificial, aprendizado de máquinas e *deep learning*. Adaptado de [1].

em camadas empilhadas. Cada camada atuará no treinamento para aprender uma representação significativa para os dados, de modo a executar a tarefa esperada de forma eficiente de acordo com a métrica de êxito. A Figura 2.2 apresenta as representações aprendidas pelas camadas de um modelo *deep learning* baseado em RNAs destinado a identificar dígitos entre 0 e 9 escritos à mão.

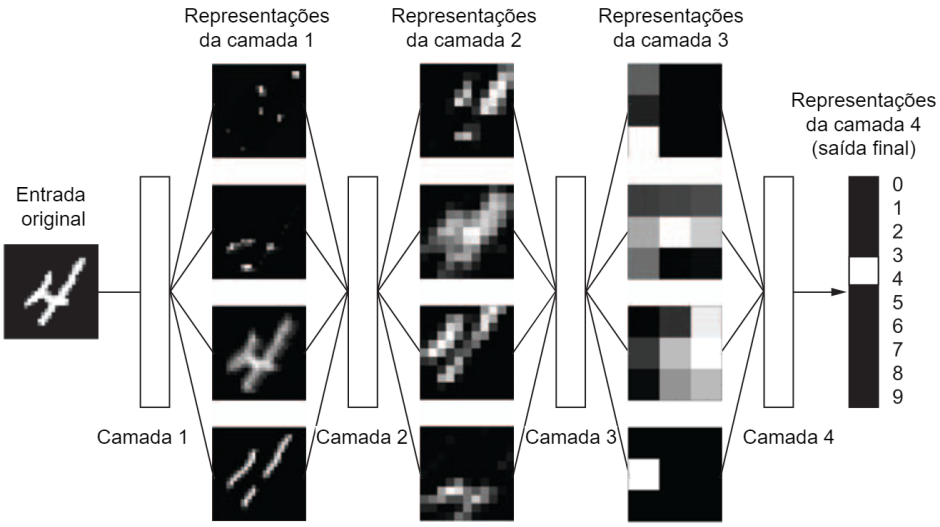


Figura 2.2: Representações aprendidas em cada camada por um modelo de classificação de dígitos escritos à mão. Adaptado de [1].

2.1.2 Redes Neurais Artificiais

Uma RNA é um modelo de AM inspirado nas redes de neurônios biológicos encontrados no cérebro humano, e podem ser considerados o coração do *deep learning* (2). Em sua forma mais geral, RNA é um processador paralelo maciçamente distribuído composto de unidades de processamento simples que tem uma aptidão natural para armazenar o conhecimento experiencial e torná-lo disponível para uso. Ela se assemelha ao cérebro em dois aspectos [45]:

a) O conhecimento é adquirido pela rede a partir de seu ambiente através de um processo de aprendizagem; e

b) Os pontos fortes da conexão interneuronal, conhecidos como pesos sinápticos, são usados para armazenar o conhecimento adquirido.

A partir do procedimento de treinamento, conforme as categorias de aprendizado apresentadas na Seção 2.1, os pesos sinápticos da rede são modificados de forma ordenada para atingir um objetivo de projeto desejado.

O perceptron, uma das formas mais simples de arquitetura de RNA, foi criada por Frank Rosemblatt em 1957 [46]. Ele foi baseado numa forma artificial ligeiramente diferente de um neurônio, e consiste na arquitetura apresentada na Figura 2.3. Nessa arquitetura, as entradas e saídas são números, e cada conexão de entrada está associada a um peso. O neurônio atua como uma unidade de limiar lógico (*Threshold Logic Unit* – TLU), que computa a soma ponderada das entradas ($z = w_1x_1 + w_2x_2 + ... + w_nx_n = \mathbf{X}^T\mathbf{W}$), e então aplica a função degrau $step$ à essa soma, e apresenta o resultado: $h_{\mathbf{W}}(\mathbf{X}) = step(z)$, onde $z = \mathbf{X}^T\mathbf{W}$. Treinar essa simples arquitetura para uma tarefa, significa encontrar os valores corretos de w_1, w_2 e w_3.

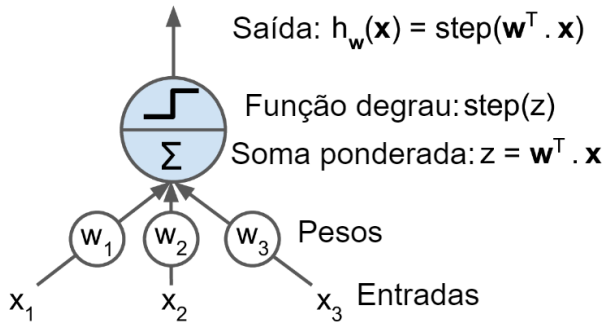


Figura 2.3: Arquitetura do perceptron de Rosenblatt. Adaptado de [2].

O perceptron é composto de uma única camada de TLUs, com cada TLU conectada a todas as entradas. As entradas do perceptron são alimentadas por neurônios especiais, denominados neurônios da camada de entrada, que apresentam como saída o mesmo sinal de entrada que recebem. Além disso, uma entrada de viés (*bias*) também é utilizada, na forma $x_0 = 1$, pois essa entrada sempre apresenta a saída fixa igual a 1.

O objetivo do perceptron é classificar corretamente o conjunto de estímulos recebidos externamente $x_1, x_2, ..., x_n$ em uma de duas classes possíveis ζ_1 ou ζ_2. Em sua forma mais simples, isso significa definir duas regiões de decisão separadas pelo hiperplano definido pela seguinte equação [45]:

$$\sum_{i=1}^{m} w_i x_i + b = 0 \tag{2.1}$$

Considerando a possibilidade de interconexão desses perceptrons, de forma semelhante aos neurônios biológicos, tem-se a formação do perceptron multicamadas (*Multilayer Perceptron* – MLP), ilustrado na Figura 2.4. Essa configuração é composta de uma camada de entrada, uma ou mais camadas intermediárias do tipo TLU, denominadas camadas escondidas, e uma camada final TLU chamada camada de saída. Todas as camadas, exceto a de saída, possuem o neurônio de viés e são totalmente conectadas na próxima camada. O empilhamento de várias camadas consiste em uma rede neural profunda (*Deep Neural Network* – DNN).

Os pesos sinápticos dos perceptrons podem ser ajustados a partir de um método passo a passo, utili-

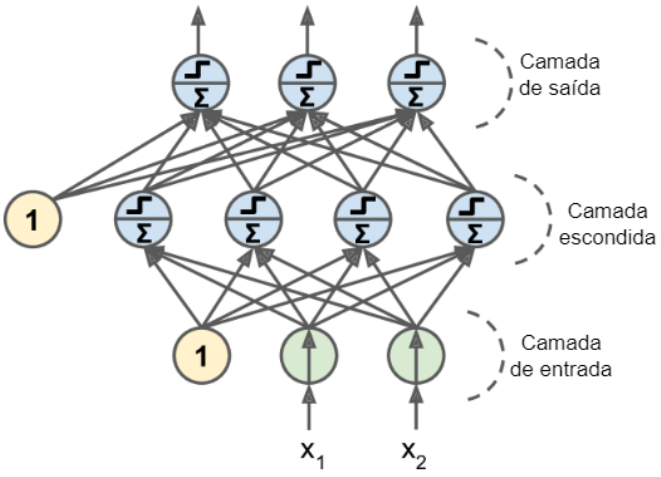


Figura 2.4: Perceptron multicamadas. Adaptado de [2].

zando uma regra de correção de erro conhecida como algoritmo de convergência do perceptron [45], que utiliza a seguinte equação para ajustar os pesos sinápticos:

$$w_{i,j}^{proximo\ passo} = w_{i,j} + \eta(y_j - \hat{y_j})x_i \tag{2.2}$$

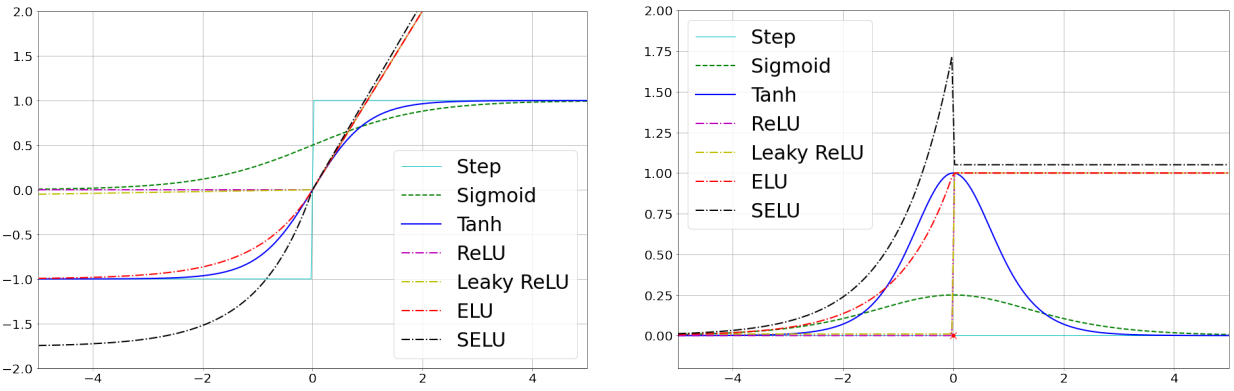
Nessa equação, $w_{i,j}$ é o peso da conexão entre o neurônio de entrada i e o neurônio j da saída, x_i é o valor da entrada i da instância de treinamento atual, $\hat{y_i}$ é a saída do neurônio j da instância de treinamento atual, e η é a taxa de aprendizado

O divisor de águas para o treinamento de MLPs veio a partir do algoritmo *backpropagation* [47], proposto por David Rumelhart, Geoffrey Hinton e Ronald Williams em 1986. O novo método utiliza uma maneira eficiente de computar automaticamente os gradientes em apenas dois passos através da MLP, e assim, consegue descobrir como cada peso de conexão e cada termo de viés deve ser ajustado a fim de reduzir o erro. Uma peça-chave para o funcionamento correto do algoritmo é a modificação da função degrau por uma função logística $sigmoid$ dada pela seguinte equação:

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \tag{2.3}$$

Essa modificação permite que o cálculo do gradiente descendente faça progressos ao longo de toda faixa de valores, pois a função logística tem derivadas diferentes de zero em toda sua curva. Essa modificação também se beneficia da utilização de outras funções de ativação, além da logística, conforme demonstrado na Figura 2.5. Pode ser observado que a função degrau (*step*) não possui derivada no ponto zero, enquanto todas as outras funções possuem derivada em todo o espectro de valores, o que as torna eficazes para uso no algoritmo *backpropagation* [47].

O uso do algoritmo *backpropagation* viabilizou um enorme progresso no uso das RNAs, e permitiu que diferentes tipos de arquiteturas fossem desenvolvidos. Uma arquitetura de RNA é a maneira pela qual os neurônios de uma rede neural são estruturados, interligados e até mesmo retroalimentados. As RNAs em que as ativações têm um fluxo em uma única direção, ou seja, da entrada para a saída, são conhecidas



(a) Funções de ativação.

(b) Derivadas das funções de ativação.

Figura 2.5: Funções de ativação comumente utilizadas e suas derivadas. Adaptado de [2].

como redes neurais *feedforward*. Contudo, nem toda RNA precisa ser sequencial ou totalmente conectada.

Um exemplo de uma RNA não sequencial é a arquitetura de RNA *Wide and Deep*, proposta por Heng Tze Cheng *et al.* em 2016 [48]. Nessa arquitetura, a totalidade ou uma parte da entrada é conectada à camada de saída, permitindo assim que a RNA aprenda tanto as representações profundas (a partir do caminho profundo da RNA), como representações simples (a partir do caminho mais curto). A Figura 2.6 ilustra um exemplo de arquitetura *Wide and Deep*. Observa-se que, nessa arquitetura, uma parte da entrada (Entrada A) é diretamente conectada a uma camada de concatenação próxima da saída (*Wide*), enquanto outra parte da entrada (Entrada B) é conectada na parte profunda da RNA (*Deep*).

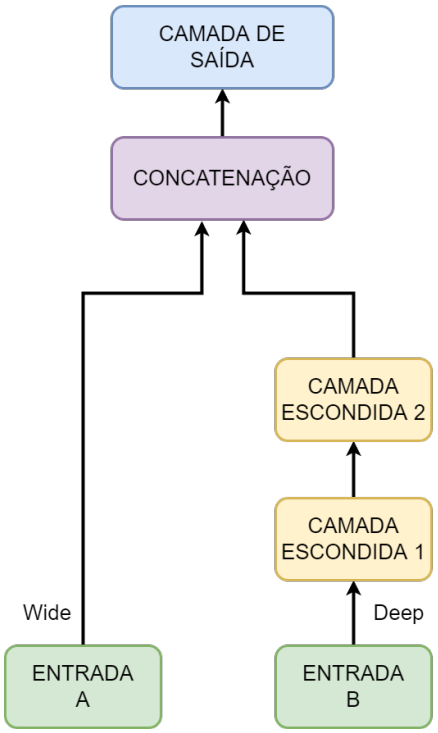


Figura 2.6: Exemplo de arquitetura *Wide and Deep* com duas entradas, sendo uma entrada *Wide* (Entrada A), e uma entrada *Deep* (Entrada B). Adaptado de [2].

Um outro tipo de arquitetura RNA que se consagrou é a Rede Neural Recorrente (*Recurrent Neural*

Network - RNN). A principal diferença de uma RNN para uma rede neural *feedforward* é que aquela possui pelo menos uma ligação de *loop* para o próprio neurônio ou neurônios em camadas anteriores. Assim, uma RNN tem a capacidade de processar sequencias a partir da iteração dos elementos da sequência e mantendo um estado com informação do evento passado. De certo modo, pode-se dizer que a RNN passa a ter uma capacidade de memória de curto prazo. Na sua forma mais simples, uma RNN é composta de um único neurônio que produz uma saída e se retroalimenta dessa saída no próximo instante. A Figura 2.7 ilustra uma RNN de uma camada. Pode-se observar que suas saídas são utilizadas como retroalimentação ao longo dos instantes de tempo.

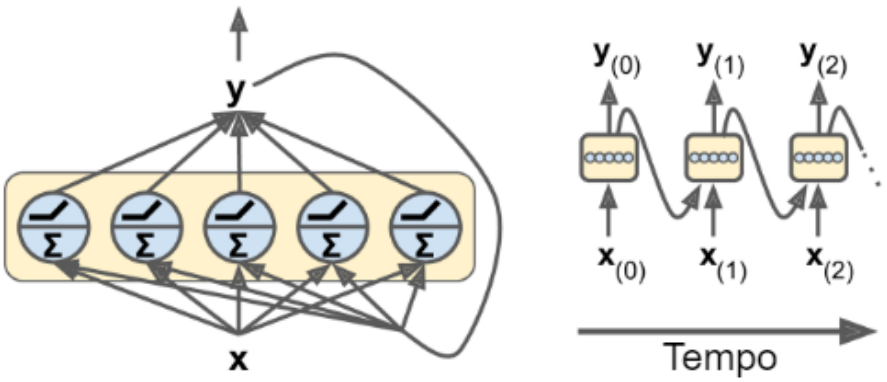


Figura 2.7: Exemplo de rede neural recorrente de uma camada. Adaptado de [2].

Esse modelo básico de neurônio recorrente permite o registro de um único estado de memória, ou seja, memória de curto prazo. Outros tipos de neurônios recorrentes foram desenvolvidos para abordar memórias de mais longo prazo. Alex Graves *et al.* propuseram em 1997 um novo tipo de neurônio recorrente, conhecido como *Long Short-Term Memory* – LSTM [49]. Suas modificações permitiram melhor desempenho, convergência mais rápida e a capacidade de detectar dependências de longo prazo nas sequências de dados [2].

Um neurônio recorrente LSTM é representado na Figura 2.8. Essa configuração permite que o neurônio possa armazenar estados de longo prazo, descartar alguns estados, e fazer novas leituras. Nessa representação, $h_{(t)}$ indica o estado de curto prazo, e $c_{(t)}$ o estado de longo prazo. A porta *forget gate* é responsável pelo descarte de estados de longo prazo, enquanto a porta *input gate* seleciona quais novas memórias devem ser adicionadas ao estado de longo prazo. Já a porta *output gate* atua na construção do estado de curto prazo. Há quatro camadas totalmente conectadas, representadas pelas caixas FC, com propósitos diferentes. A FC principal, cuja saída é $g_{(t)}$, atua analisando a entrada atual e o estado de curto prazo anterior $h_{(t-1)}$, e utiliza a função de ativação $tanh$ para gerar sua saída. As outras três FCs atuam como controladores de porta, utilizando a função logística $sigmoid$ para gerar suas saídas.

Outro tipo de arquitetura de destaque na literatura são as redes convolucionais (*Convolutional Neural Networks* – CNN), cuja inspiração veio a partir de estudos da estrutura do cortex cerebral. Um importante avanço no estudo das CNNs veio com o trabalho de Yann LeCun *et al.* em 1998 [50]. Em sua proposta inicial, uma CNN é uma MLP destinada a reconhecer formas bidimensionais com alto grau de invariância à translação, escalonamento, desvio e outras formas de distorção (45). Para alcançar esse objetivo, a estrutura da CNN inclui as seguintes características [45]:

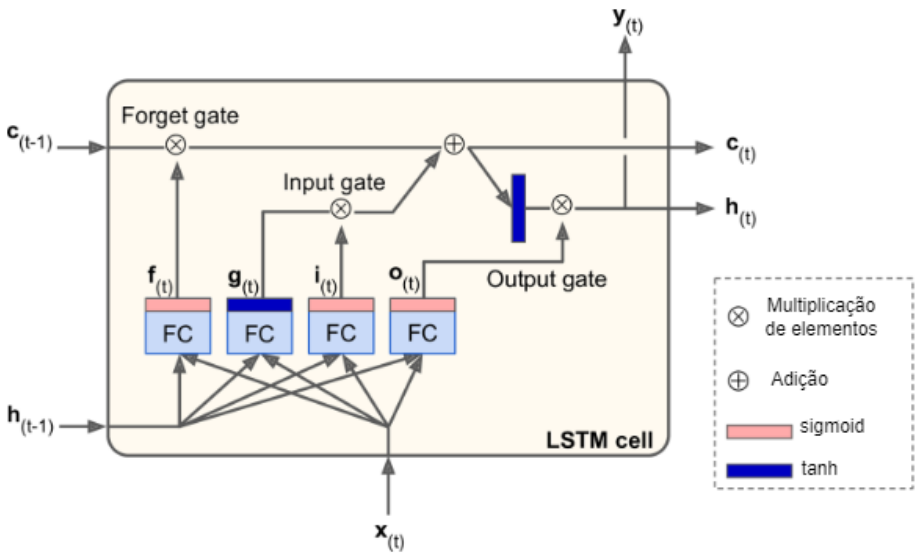


Figura 2.8: Neurônio recorrente *Long Short-Term Memory* – LSTM. Adaptado de [2].

a) Extração de *features* [2]: Cada neurônio extrai uma representação local de *features* a partir do seu campo receptivo sobre a camada anterior. Assim que representações são extraídas, sua localização exata torna-se dispensável, bem como sua posição relativa a outras características é preservada de forma aproximada;

b) Mapeamento de *features*: Cada camada da CNN é composta de múltiplos mapas de *features*, com cada mapa tendo a forma de um plano no qual todos os neurônios compartilham o mesmo conjunto de pesos sinápticos, o que garante invariância a deslocamento; e

c) Subamostragem: cada camada convolucional é seguida de uma camada que realiza média local e subamostragem, reduzindo a resolução do mapa de *features*. Com isto, a sensibilidade do mapa é reduzida quanto a efeitos de deslocamentos e outras distorções.

A principal diferença entre uma camada densa (i.e., totalmente conectada) e uma camada convolucional é que a camada densa aprende os padrões globais a partir do espaço de *features* de entrada, enquanto a camada convolucional aprende padrões locais [1]. A Figura 2.9 ilustra a captura de padrões locais por uma camada convolucional.

A peça-chave de uma CNN é a forma de conexão de suas camadas. Os neurônios na primeira camada convolucional não estão conectados a todos os pixels da imagem de entrada como ocorre em camadas densas, mas somente a pixels no campo de recepção. Por sua vez, os neurônios da segunda camada convolucional estão conectados somente aos neurônios da primeira camada localizados em seu campo receptivo. Essa arquitetura permite que a rede se concentre em aprender apenas os pequenos detalhes na primeira camada, que serão combinados com maiores características na segunda camada e assim por diante. Essa arquitetura permitiu que as CNNs tenham logrado tanto êxito no reconhecimento de imagens (espaço bidimensional - 2D).

[2] *Features* são as variáveis ou atributos independentes relacionados ao objeto que se está buscando analisar.

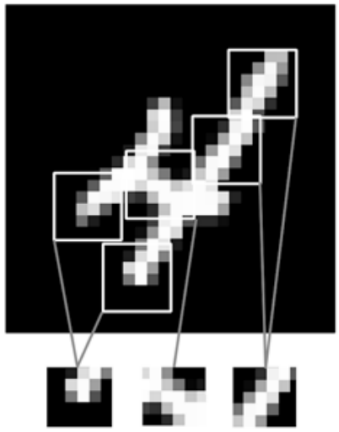


Figura 2.9: Exemplo de padrões locais aprendidos por uma camada convolucional. Adaptado de [1].

Esse êxito das CNNs também se aplica ao processamento de sequências unidimensionais, tais como geração de sinais de áudio e tradução. Quando uma CNN é aplicada sobre um espaço unidimensional – 1D, diz-se que é uma convolução temporal. De forma semelhante à convolução 2D, a convolução temporal extrai subsequências 1D das sequências, conforme ilustrado na Figura 2.10. A convolução unidimensional reconhece padrões locais nas sequências, e como a mesma transformação de entrada é aplicada a todas as subsequências, uma vez que um padrão é aprendido em uma posição, ele pode ser posteriormente reconhecido em uma posição diferente. Com isso, a convolução temporal também é invariante à translação.

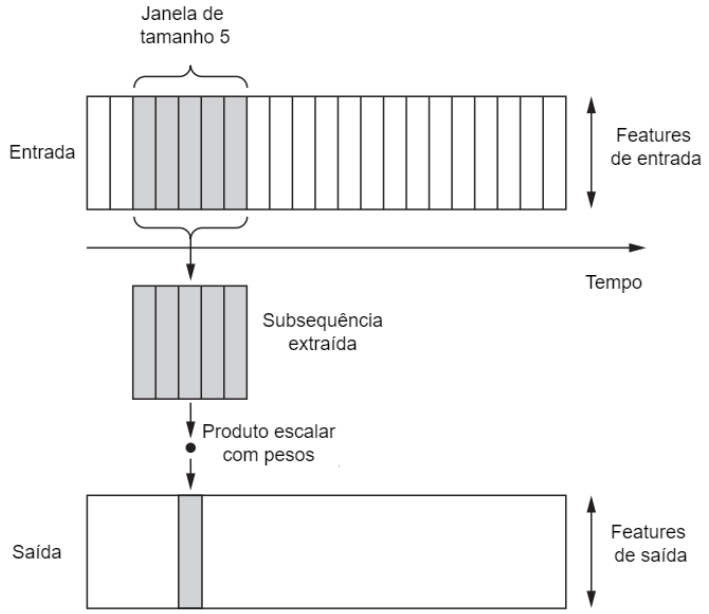


Figura 2.10: Funcionamento da convolução unidimensional. Adaptado de [1].

2.2 DETECÇÃO DE ANOMALIAS

Conceitualmente, uma anomalia é caracterizada quando uma determinada observação se desvia significativamente de outras observações a ponto de se suspeitar que foi gerada a partir de um mecanismo diferente daquele que produz os dados normais e não seja caracterizada como ruído [3].

Os inúmeros avanços das últimas décadas decorrentes de reconhecimento de padrões e mineração de dados, combinado com o aumento do poder computacional que tem facilitado endereçar problemas complexos que antes não eram possíveis, permitiu que a ciência de dados evoluísse exponencialmente, incluído o estudo das anomalias [16].

As anomalias existem em inúmeros domínios de estudo, como medicina, finanças, segurança cibernética, sociologia, entre vários outros. Algumas anomalias estão associadas a eventos indesejáveis, como fraudes em cartões de crédito, ataques cibernéticos, ou falhas em componentes de uma linha de produção fabril. Em outras situações, as anomalias estão associadas a resultados positivos, como situações de explosão de vendas no comércio.

As abordagens de detecção de anomalias são baseadas em modelos e previsões a partir de dados históricos [16]. Portanto, uma primeira consideração é que o comportamento esperado (ou normal) é estacionário, ou seja, os processos que levaram à geração dos dados não devem mudar significativamente.

Para alguns domínios de conhecimento, a detecção de anomalias tem papel relevante [3]:

a) Sistemas de detecção de intrusão: Diferentes tipos de dados coletados acerca de sistemas operacionais, atividades de usuários, tráfego de rede, entre outros, permitem identificar comportamentos incomuns característicos de atividade maliciosa;

b) Fraudes em cartão de crédito: Fraudes em cartões tornaram-se habituais diante dos inúmeros vazamentos de dados ocorridos recentemente. Em muitos casos, o uso não autorizado dos cartões demonstra padrões completamente diferentes, como locais e horários de compras, ou ainda valores exorbitantes;

c) Diagnósticos em medicina: Em diferentes aplicações médicas, os dados são coletados a partir de sensores ou dispositivos, e padrões anômalos podem indicar condições vinculadas a doenças.

Algumas anomalias podem ser encontradas em dados históricos, como por exemplo, ataques cibernéticos que ficaram registrados nos *logs* de eventos de segurança. Isso permite que assinaturas descritivas dos ataques sejam estabelecidas, de modo a permitir a identificação futura dos mesmos eventos que levaram ao incidente. Isso permite que regras sejam configuradas para o bloqueio das atividades relacionadas a esses eventos, tais como regras de bloqueio de *firewalls* ou de programas antivírus.

Contudo, essa abordagem de classificação atende apenas à detecção de problemas conhecidos em contextos específicos, enquanto os grandes desafios em segurança cibernética consistem em detectar os eventos desconhecidos que podem causar grandes danos. Além disso, há situações em que ações corretivas não podem ser realizadas com rapidez suficiente para evitar uma catástrofe, como, por exemplo, quando um funcionário fornece sua senha a alguém por telefone, e a segurança de um sistema sensível é comprometida.

Nesses casos, são necessários modelos de detecção de anomalias com capacidade de detecção abrangente, pois, ainda que os dados observáveis reflitam anomalias no processo ou comportamento subjacente,

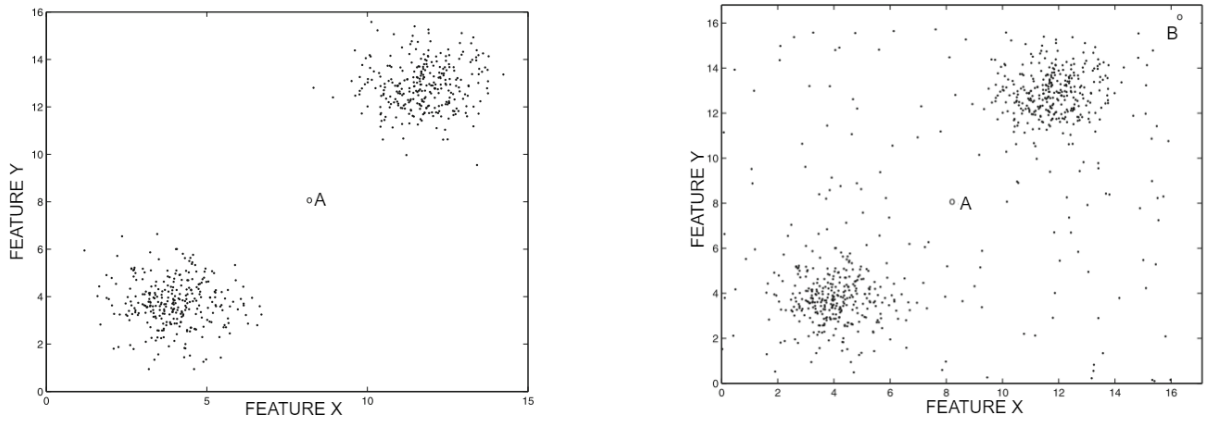
não se espera que as mudanças correspondentes nos dados sigam padrões previamente conhecidos. O modelo busca detectar que algo está errado, mesmo que não seja capaz de dizer exatamente como ou porquê isso ocorreu [16].

Um algoritmo de detecção de anomalia pode produzir como resultado:

a) Uma pontuação de anomalia: um valor quantificando o nível de anomalia atribuído ao dado analisado, que pode ser utilizado para criar uma ordenação dos dados em função da pontuação obtida; e

b) Rótulos binários: uma informação de classificação binária indicando se o dado é uma anomalia ou não. Em determinados casos, a partir da aplicação de um valor de limiar sobre as pontuações obtidas, podem ser obtidos como resultados os rótulos binários.

Um aspecto importante na detecção de anomalias é que ela está vinculada a um criério subjetivo de julgamento sobre a quantificação do desvio que indicará a anomalia. Além disso, muitas vezes junto da anomalia existirá ruído, o que pode tornar a tarefa de classificação ainda mais difícil, como ilustrado na Figura 2.11. Na Figura 2.11a, observa-se que o ponto marcado como "A" claramente parece ser diferente dos dados restantes, indicando ser uma anomalia. Já na situação apresentada na Figura 2.11b, a definição se torna muito mais subjetiva, pois há aparente ruído presente nos dados, dificultando a identificação de anomalias. Nesse último caso, baseando-se nas distribuições dos dados sabidamente normais, é mais provável que o ponto "B" seja considerado como anomalia e o ponto "A" como ruído.



(a) Anomalia em ambiente sem ruído.

(b) Anomalia em ambiente com ruído.

Figura 2.11: Diferença entre ruído e anomalia. Adaptado de [3].

A escolha do modelo é um aspecto fundamental para o sucesso na detecção de anomalias. Devido ao fato de ser um problema de natureza tipicamente não supervisionada, a escolha do modelo, na prática, se dá pela compreensão do analista acerca dos desvios significativos que são relevantes para a aplicação. Em outras palavras, a escolha dos melhores modelos é orientada pela especificidade dos dados no tocante ao domínio de estudo, baseado em suposições sobre a estrutura dos padrões normais em um determinado conjunto de dados. Claramente, a escolha do modelo "normal" depende muito da compreensão do analista sobre os padrões de dados naturais naquele domínio específico.

2.3 APRENDIZADO DE MÁQUINAS NA DETECÇÃO DE ANOMALIAS

A detecção de anomalias vem sendo explorada por uma comunidade ampla, compreendendo bancos de dados, mineração de dados, estatística e medicina. A evolução do poder de processamento aliado ao uso crescente do AM[3] têm permitido avanços importantes no campo da detecção de anomalias. Problemas complexos hoje são processados por meio de poderosos computadores com várias placas gráficas (*Graphics Processing Unit* – GPU), ou ainda, por meio de processamento paralelo distribuído, o que permite que questões antes impossíveis se tornem viáveis.

A detecção de anomalias utilizando o AM é um campo vasto, cuja literatura especializada [3, 19, 20, 32, 33, 34, 35, 36, 27, 16] nos permite ter uma compreensão aprofundada acerca do tema. Considerando que neste trabalho utilizamos um subconjunto do domínio de estudo de AM aplicado à detecção de anomalias, apresentaremos a seguir os principais algoritmos e arquiteturas de AM focados na detecção de anomalias que foram utilizados.

2.3.1 Isolation Forest

Isolation Forest – IF [51] é um algoritmo de AM não supervisionado baseado em um *ensemble* [3] de *Isolation Trees* – IT. Numa IT, os dados são recursivamente particionados em *features* e valores aleatoriamente definidos, a fim de isolar as instâncias em nós com cada vez menos instâncias até que os valores sejam isolados em nós contendo apenas uma instância. Nesses casos, os galhos de árvores contendo anomalias serão consideravelmente menos profundos, já que os valores estão localizados em regiões esparsas, ou seja, com menos dados [3]. Portanto, a distância de uma folha para a raiz é utilizada como uma pontuação de anomalia.

Durante o treinamento da IF, várias *isolation trees* são construídas. Cada árvore é binária, possuindo no máximo N nós folha para um conjunto de dados contendo N elementos. Para construir uma IT a partir de um conjunto de dados contendo N elementos, inicialmente é criado um nó raiz contendo todos os elementos, e este será o estado inicial da IT T. Uma lista candidata contendo os nós C para próximas divisões é inicializada, e então os passos seguintes são repetidos para criar a IT T até que a a lista canditada C esteja vazia:

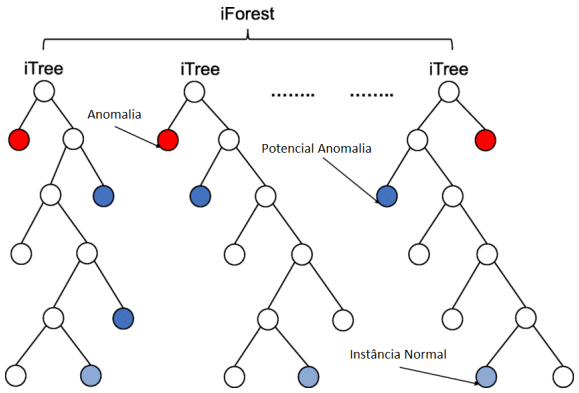
a) Selecione aleatoriamente um nó R de C e o remova.

b) Selecione uma *feature* i aleatoriamente e divida os dados no nó R em dois subconjuntos R_1 e R_2 a partir de um valor a aleatoriamente escolhido na *feature* i. Todos os elementos em R_1 satisfazem $x_i \leq a$, e todos os elementos em R_2 satisfazem $x_i > a$. O valor a é escolhido aleatoriamente entre os valores máximo e mínimo da *feature* i, dentre os elementos no nó R. Os nós R_1 e R_2 são fihos de R em T.

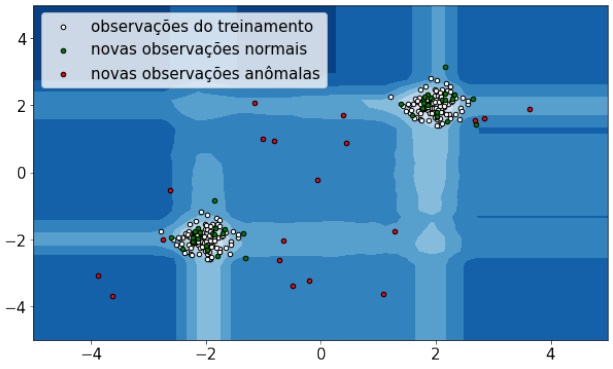
c) Executar essa etapa para cada $i \in \{1, 2\}$. Se R_i contém mais que um elemento, então acrescente ao conjunto C. Caso contrário, considere o nó como uma folha IT.

[3] *Ensemble* é uma técnica popularmente utilizada em AM para melhorar a acurácia de algoritmos a partir da combinação de saídas de diferentes algoritmos para a obtenção de uma única saída cujo resultado supera os resultados individuais [3].

Como resultado desse processo, uma árvore binária tipicamente não balanceada será criada e as anomalias deverão ser encontradas mais rapidamente que os dados normais. A Figura 2.12 apresenta um exemplo de cenário de detecção de anomalias utilizando o algoritmo IF. A Figura 2.12a mostra a estrutura do IF por meio da representação dos elementos do *dataset* nas árvores IT. A Figura 2.12b ilustra como a classificação dos dados pelo IF se apresenta no *dataset* : os elementos em branco correspondem aos dados de treinamento, enquanto os elementos azuis são valores identificados como normais na fase de teste, e os elementos vermelhos são anomalias detectadas no conjunto de teste.



(a) Representação das árvores *Isolation Tree* no algoritmo *Isolation Forest*. Adaptado de [52].



(b) Exemplo de detecção de anomalias com o algoritmo *Isolation Forest*. Adaptado de [53].

Figura 2.12: Detecção de anomalias com o algoritmo *Isolation Forest*.

2.3.2 One-Class Support Vector Machine com Gradiente Descendente Estocástico

O problema de detecção de anomalias pode ser interpretado como um problema de classificação no qual as classes "normal" e "anomalia" não estão disponíveis. Assim, pelo fato de que os dados normais são a grande maioria dos elementos, é possível abordar a totalidade do *dataset* como se fosse composto de dados normais, e portanto, criar um modelo de classificação dessa classe normal. Desse modo, eventos que se desviam dos dados normais são tratados como anomalias. Essa ótica permite que muitos modelos classificadores multi-classe sejam utilizados para detecção de anomalias por meio de treinamento não supervisionado, sendo chamados de *One-Class Analogs* [3].

Nesse sentido, o algoritmo *One-Class Support Vector Machine* – OCSVM é uma adaptação do algoritmo SVM. A formulação moderna do SVM foi desenvolvida por Vladimir Vapnik e Corinna Cortes [54], sendo conhecida como Métodos de Kernel [1] ou Máquinas de Kernel [4], e seu objetivo é encontrar fronteiras de decisão para a classificação. Para encontrar as fronteiras, o SVM procede da seguinte forma [1]:

a) Os dados são mapeados para uma nova representação multidimensional onde as fronteiras de decisão podem ser expressas como um hiperplano; e

b) Uma fronteira de decisão é calculada por meio da maximização da distância entre o hiperplano e os dados mais próximos que pertençam a cada classe, o que é chamado de maximização da margem. Isso permite que a fronteira possa generalizar para dados não presentes no *dataset* .

Para que essa abordagem do SVM seja computacionalmente viável é utilizado o que se chama de

Kernel trick, que é a utilização de funções que mapeiam quaisquer dois pontos de um espaço dimensional para a distância entre esses pontos no espaço de representação final. Com isso, o SVM funciona muito bem para *datasets* com muitas dimensões. Por outro lado, o SVM demonstra ser um algoritmo que não escala bem para *datasets* grandes [1, 4, 2].

O OCSVM é semelhante ao SVM, porém com a diferença que ele procura separar as instâncias de dados no espaço multidimensional em relação à origem. No espaço dimensional original, isso corresponde a encontrar uma pequena região que engloba todas as instâncias. Se uma nova instância não recai sobre essa região, então ela é considerada uma anomalia. A Figura 2.13 exemplifica a fronteira de decisão do OCSVM. Na imagem, o OCSVM atribui a fronteira mais suave a partir de um kernel linear (o círculo), sendo possível verificar três situações: Em (a) a instância é considerada normal; em (b) a instância recai sobre a fronteira de decisão (ou vetor de suporte); e em (c) a instância é considerada uma anomalia.

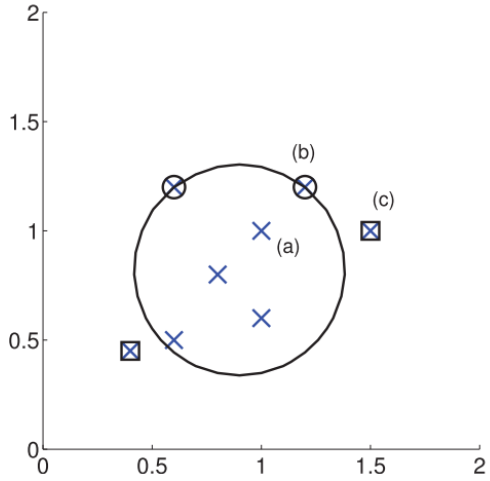


Figura 2.13: Exemplo do algoritmo *One-Class SVM*. Adaptado de [4].

Da mesma forma que o SVM, o OCSVM não escala para grandes *datasets*. A complexidade de modelos OCSVM com Kernel é no melhor caso quadrática de acordo com o número de amostras. Thong W Zhang [55] elaborou uma implementação do gradiente estocástico descendente (*Stochastic Gradient descent* – SGD) em modelos lineares de predição, tendo inspirado a aplicação do algoritmo SGD no modelo SVM, tornando-se SVM-SGD [56].

Assim, o OCSVM-SGD é a adaptação do SVM-SGD para classificação de única classe (*one class analog*). A técnica consiste em realizar aproximações de kernel utilizando o algoritmo SGD para alcançar uma solução de OCSVM com kernel, porém com complexidade linear de acordo com o número de amostras. Isso permite que o OCSVM-SGD seja adequado para *datasets* com grande número de amostras.

2.3.3 Autoencoders

Cottrell, Munro, e Zipser [57] propuseram em 1987 uma arquitetura de RNA denominada Autoencoder, capaz de aprender representações densas (ou reduzidas) dos dados de entrada sem supervisão, ou seja, sem dados de entrada rotulados. As representações aprendidas são classificadas como representações latentes ou codificações. Essas representações latentes possuem uma dimensionalidade menor que a dos

dados de entrada, o que permite que sejam utilizados para redução de dimensionalidade, pré-treinamento não supervisionado, ou mesmo para gerar novos dados semelhantes aos dados de treinameto (modelos generativos) [2].

A Figura 2.14 apresenta a arquitetura de um autoencoder com três camadas ocultas. Observa-se que o número de saídas é igual ao número de entradas, e cada valor de entrada x_i é reconstruído para a mesma dimensão para x'_i. Também pode ser observado que as camadas ocultas fazem a redução de dimensionalidade a partir da subsequente diminuição do número de células de cada camada, priorizando assim o aprendizado da representação das características latentes dos dados de entrada.

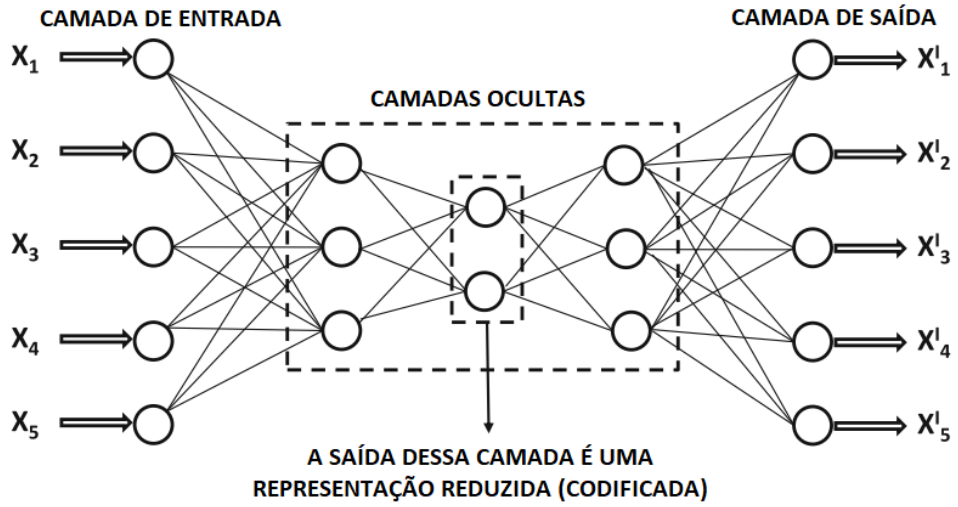


Figura 2.14: Exemplo da arquitetura de autoencoder. Adaptado de [3].

Durante o treinamento, os autoencoders buscam obter como saída os próprios dados de entrada, o que permite que também sejam classificados como modelos de aprendizado auto-supervisionado. O erro de reconstrução acumulado sobre todas as d dimensões em todas as instâncias dos dados é calculado conforme a Equação 2.4. Durante o treinamento da RNA esse erro é minimizado a partir dos melhores ajustes dos pesos, para que as saídas sejam o mais próximas possível com as entradas.

$$\delta = \sum_{i=1}^{d} (x_i - x'_i)^2 \tag{2.4}$$

Autoencoders são uma escolha natural para detecção de anomalias [3]. O valor de erro de reconstrução de cada instância do *dataset* proporciona uma pontuação de detecção de anomalia para aquela instância. A arquitetura de RNA multicamadas do autoencoder proporciona uma redução de dimensionalidade mais geral que técnicas como PCA, de modo que qualquer tipo de redução não linear se torna possível com essa arquitetura.

É comum abordar a estrutura dos autoencoders como composta de duas partes, o *encoder* e o *decoder*, conforme ilustrado na Figura 2.15. A primeira parte dessa RNA, o *encoder*, aprende a função de codificação ϕ, e a segunda parte, o *decoder*, aprende a função de decodificação ψ. Portanto, $\phi(D)$ consiste na representação reduzida (ou latente) do *dataset* D. Aplicando a função de decodificação ψ em $\phi(D)$,

obtém-se a reconstrução dos dados $D' = (\psi o \phi)(D)$, que pode não ser exatamente igual à D. Anomalias são resistentes à compressão, e portanto, irão apresentar grande variação entre D e D'. Assim, os valores absolutos da matriz residual $(D - D')$ proporcionam as pontuações de anomalias.

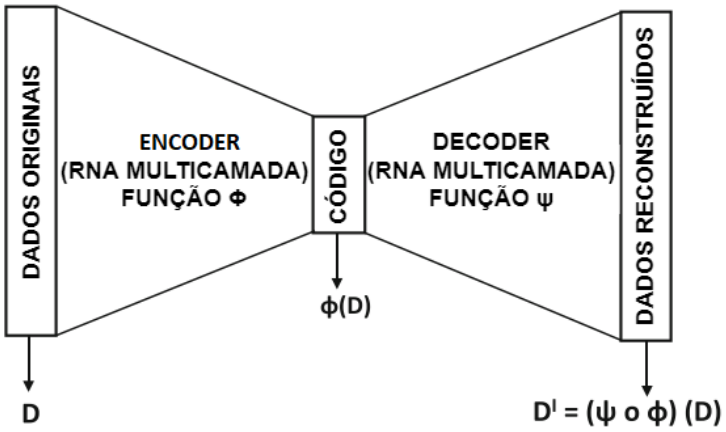


Figura 2.15: Estrutura do autoencoder evidenciando a redução de dimensionalidade ou codificação (*encoder*) e reconstrução dos dados ou decodificação (*decoder*). Adaptado de [3].

2.4 MÉTRICAS DE DESEMPENHO UTILIZADAS NA DETECÇÃO DE ANOMALIAS DE COMPORTAMENTO

Aggarwal *et al.* [3] descreve que a efetividade da detecção de algoritmos de detecção de *anomalias* é uma tarefa desafiadora em virtude de sua natureza rara. Isso se torna ainda mais difícil quando não se dispõe dos rótulos reais dos dados como anomalias ou normais. Considerando que estejam disponíveis tais rótulos, é importante considerar que o *dataset* possivelmente estará desbalanceado, com um alto número de classes normais e uma pequena quantidade de eventos anômalos. Assim, algumas métricas são mais indicadas à validação da eficiência de modelos de detecção de anomalias.

2.4.1 Matriz de Confusão

A matriz de confusão, ou *Confusion Matrix* – CM é uma forma de avaliar a performance de um classificador, indicando a quantidade de classificações corretas e equivocadas. Considerando que a classificação de anomalias consiste em determinar apenas duas classes, anomalia ou normal, a CM consiste numa matriz na qual as linhas correspondem às classes reais, e as colunas correspondem às predições do modelo, e os valores presentes correspondem à quantidade de acertos da classe positiva (Verdadeiro Positivo, ou *True Positive* – TP), a quantidade de acertos da classe negativa (Verdadeiro Negativo, ou *True Negative* – TN), a quantidade de predições positivas que são na verdade negativas (Falso Positivo, ou *False Positive* – FP), e a quantidade de predições negativas que são na verdade positivas (Falso Negativo, ou *False Negative* – FN). A Figura 2.16 ilustra uma matriz de confusão de um classificador de anomalias.

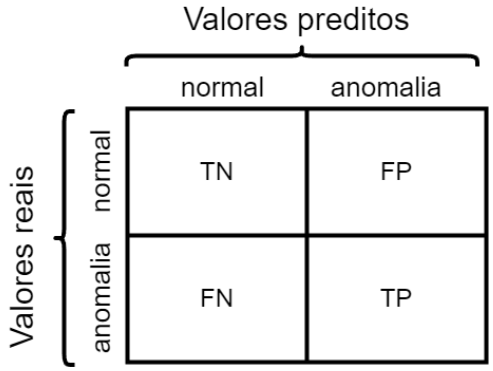


Figura 2.16: Formato da Matrix de Confusão.

2.4.2 *Precision* e *Recall*

A grande maioria dos modelos de detecção de anomalias produz como saída uma pontuação de anomalia. Aplicando-se um limiar sobre essa pontuação, é possível determinar os rótulos de anomalias. Se a escolha do limiar é muito restritiva para minimizar o número de anomalias, o modelo não detectará anomalias reais, pois as considerará como dados normais, situação na qual se encontram os falso negativos. No outro extremo, caso a esolha do limiar seja muito ampla, o modelo irá declarar dados normais como anomalias, situação na qual se encontram os falso positivos. Esse balanço por ser mensurado a partir das métricas de precisão (*Precision*) e revocação (*Recall*).

A métrica *Precision* é definida como a taxa de anomalias detectadas que realmente pertencem à classe de anomalias, sendo definida pela seguinte equação:

$$Precision = (\frac{TP}{TP + FP}) \tag{2.5}$$

A métrica *Recall* consiste na taxa de anomalias reportadas pelo modelo dentre o conjunto total de anomalias reais existentes. O *Recall* também é conhecido pelo termo sensitividade (*sensitivity*), ou taxa de verdadeiro positivos (*True Positive Rate* – TPR), e é definido pela seguinte equação:

$$Recall = (\frac{TP}{TP + FN}) \tag{2.6}$$

A depender do objetivo de um modelo de classificação, a escolha do limiar que definirá a linha de corte entre considerar que um dado é uma anomalia ou um evento normal implicará na preferência de priorização entre *Precision* e *Recall*. No caso da detecção de anomalias de comportamento de usuário, onde a anomalia é considerada um evento suspeito que deve ser investigado, é importante que a maior quantidade possível de anomalias sejam detectadas, ainda que eventos normais possam ser classificados como anômalos (falso positivos). Em outras palavras, a métrica *Recall* possui maior importância para a detecção de anomalias de comportamento quando comparada com a métrica *Precision*.

2.4.3 Curva *Receiver Operating Characteristic* – ROC

O impacto da escolha de diferentes valores de limiar nas métricas *Precision* e *Recall* pode ser avaliado por meio da curva de características operacionais do receptor (*Receiver Operating Characteristic* – ROC). Essa curva apresenta a relação entre a taxa de verdadeiro positivos (*Recall*) e a taxa de falso positivos (*False Positive Rate* – FPR), mostrando a variação da resposta do modelo para diferentes valores de limiares. A taxa de falso positivos é definida como:

$$FPR = (\frac{FP}{FP + TN}) \tag{2.7}$$

A FPR é igual a 1 - especificidade (*specificity*). A espeficidade é definida como a taxa de verdadeiro negativos (*True Negative Rate* – TNR), e é definida pela seguinte equação:

$$TNR = (\frac{TN}{FP + TN}) \tag{2.8}$$

A Figura 2.17 ilustra a curva ROC. Quanto melhor o classificador, mais sua curva se aproximará da extremidade superior esquerda da curva, ou seja, quanto mais alto o verdadeiro positivo e quanto mais baixo o falso positivo. A linha tracejada representa a curva de resposta de um classificador puramente aleatório.

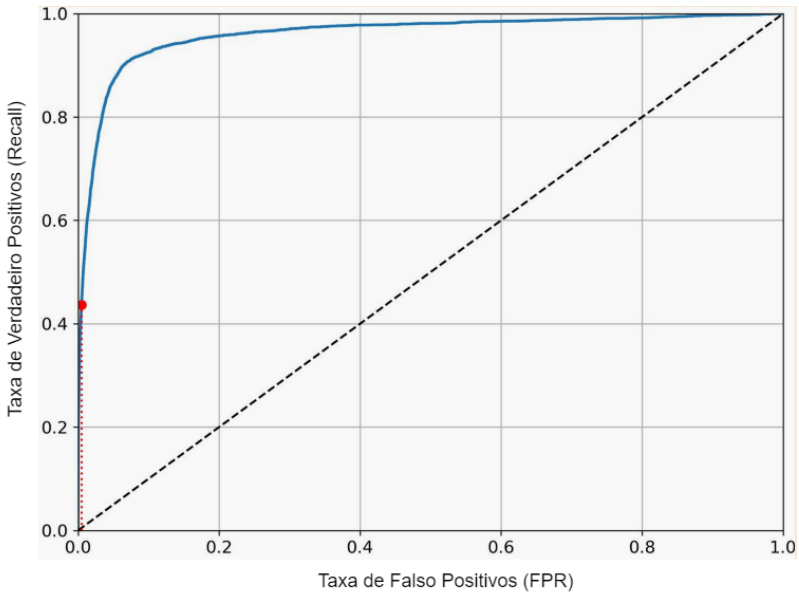


Figura 2.17: Curva Receiver Operating Characteristic - ROC. Adaptado de [2].

A curva ROC permite uma análise visual das respostas dos classificadores. A partir do cálculo da área sob a curva ROC (*Area Under the ROC Curve* – ROC-AUC) pode ser avaliado qual classificador tem a melhor resposta em função do maior valor de ROC-AUC. Quanto mais próximo de 1 for o valor de ROC-AUC, melhor o desempenho do classificador.

3 TRABALHOS CORRELATOS

Nos últimos anos, o uso de técnicas de AM e *big data* tem sido recorrente no campo da detecção de anomalias de comportamento. Esse avanço pode ser justificado por duas razões principais. A primeira decorre dos avanços em capacidade computacional com tecnologias em nuvem e computação distribuída, que têm permitido o uso de algoritmos mais complexos com maior volume de dados. A segunda é relacionada à escalada dos ataques cibernéticos, o que vem demandando a detecção precoce desses incidentes para acelerar a mitigação e facilitar a recuperação.

Conforme apresentado no Capítulo 2, uma anomalia é uma espécie de *outlier* compreendendo uma instância de dado que se desvia significativamente do comportamento esperado da maioria dos dados (*inliers*), e que não pode ser considerada ruído. Isto implica que a detecção de anomalias é um problema de natureza não supervisionada, no qual não há exemplos anteriores de anomalias disponíveis. Por essa razão, a obtenção de *datasets* rotulados para treinamento de modelos representa um grande obstáculo para a construção de modelos. Apesar disso, alguns trabalhos têm usado aprendizagem supervisionada para a detecção de anomalias de comportamento utilizando *datasets* rotulados públicos, como o CERT *Dataset*.

Diante desse cenário, neste capítulo serão apresentadas as diferentes abordagens que vêm sendo propostas para a detecção de anomalias, as quais podem ser separadas em dois grandes grupos: (i) Abordagens de Aprendizado Supervisionado; e (ii) Abordagens de Aprendizado Não Supervisionado.

3.1 ABORDAGENS DE APRENDIZADO SUPERVISIONADO PARA DETECÇÃO DE ANOMALIAS DE COMPORTAMENTO

Abordagens de aprendizado supervisionado pressupõem a existência de um *dataset* rotulado com os dados normais e as anomalias de comportamento que devem ser detectadas. Como forma de incentivar a evolução de pesquisas, há organizações que compartilham *datasets* sintéticos rotulados com anomalias específicas a um domínio de problema, como é o caso do Cert *Insider Threat Tools Dataset*, disponibilizado pela Carnegie Mellon's Software Engineering Institute, Pittsburgh, PA, USA [58], cuja proposta é a detecção de *insiders* dentre os usuários de uma organização.

Zhang *et al.* [37] apresentam um método para encontrar *insiders* no CERT *Dataset*, versões R4.2 e R6.2. Inicialmente, a etapa de extração de *features* utiliza a técnica *Term Frequency and Inverse Document Frequency* – TF-IDF para obter as características dos comportamentos dos usuários a partir dos *logs* de eventos, a fim de quantificar a importância contextual de entidades presentes nesses registros, como por exemplo, uma determinada URL (*Uniform Resource Locator*) presente nos *logs* de navegação na internet. A partir do conjunto de *features* obtido, os dados são submetidos a um conjunto de múltiplos detectores de anomalias composto de camadas de células LSTM. Como utilizam treinamento supervisionado sobre um *dataset* altamente desbalanceado, como forma de minimizar *overfitting*, os autores utilizam no treinamento uma estratégia baseada no método de *bagging*, também conhecido com o *bootstrap aggregating*, que consiste em subamostrar o conjunto de anomalias rotuladas para combinar com o conjunto de dados normais

no treinamento, criando um equilíbrio de classes. O modelo é treinado com 90% dos dados do *dataset*, e os 10% restantes são utilizados para teste.

O problema central dessa abordagem [37] é que devido ao uso de treinamento supervisionado, há uma grande chance de que o modelo resultante não seja capaz de generalizar, e assim pode não ser capaz de encontrar anomalias diferentes daquelas existentes no *dataset*, o que pode ainda ser intensificado dado o pequeno número de anomalias presentes devido ao grande desbalanceamento. Um segundo problema é que a utilização de 90% dos dados para treinamento corresponde a aproximadamente utilizar um ano e quatro meses de dados para estabelecer os padrões de comportamentos dos usuários. Isso implica em não dispor da detecção de anomalias de comportamento por um período muito distante de um cenário real, em que se faz necessário estabelecer padrões comportamentais em período mais curto para então iniciar a detecção de anomalias.

Kim *et al.* [24] também utilizam o CERT *Dataset* R6.2 para a detecção de *insiders*. Inicialmente, a etapa de extração de *features* consiste na elaboração de resumos diários das totalizações de ações presentes nos *logs* de eventos, como por exemplo, a quantidade de conexões USB por dia. Essa etapa gera um total de 60 *features* comportamentais. A detecção de anomalias é realizada a partir da combinação das técnicas de estimativa de densidade Gaussiana, estimativa de densidade de janelas Parzen, PCA, e clusterização K-means para classe única (*one class*). Aqui também os autores utilizaram 90% dos dados para treinamento, os 10% restantes para testes, mantendo as anomalias apenas no conjunto de testes.

Dentre as limitações do trabalho [24], a primeira é que a escolha das *features* foi realizada a partir da verificação de sua importância na detecção das anomalias do conjunto de teste, ou seja, nas melhores respostas, o que enfraquece ainda mais a capacidade de detectar anomalias diferentes daquelas poucas presentes no conjunto de dados. Uma segunda limitação é novamente a utilização de 90% do *dataset* para treinamento, o que foge a uma possível aplicação em um cenário real. Além disso, a avaliação deixa de verificar se a grande quantidade de *features* inseridas não trouxe consigo o problema da dimensionalidade (*curse of dimensionality*) [20].

Mohammed *et al.* [39] apresentam um *framework* multicamadas inteligente para a detecção de *insiders* no CERT *Dataset* R4.2. O modelo utiliza uma primeira camada do *framework* para escolher o melhor modelo de classificação de detecção de *insiders* entre nove modelos com base na integração de métodos de entropia aplicando o algoritmo de tomada de decisão multicritério (*multi-criteria decision making*). A segunda camada é construída a partir dos dois melhores modelos escolhidos na camada anterior, sendo composta de um modelo híbrido de detecção de *insiders*, utilizando o algoritmo *Random Forest*, e em seguida, aplicando o algoritmo *K-Nearest Neighbors*. Os autores criaram quatro conjuntos de treinamento e teste, sendo que em dois deles havia apenas atividades normais e maliciosas dos usuários sabidamente *insiders*, e em outros dois, usando 90% e 80% dos dados para treinamento, e 10% e 20% para testes.

Verifica-se que principal limitação desse trabalho [39] consiste em obter um modelo que é especializado em encontrar as anomalias presentes no *dataset*, mais precisamente, aquelas dos usuários sabidamente *insiders*, o que possivelmente irá influenciar na capacidade de encontrar anomalias diferentes daquelas existentes. Além disso, o modelo também utiliza uma quantidade de dados para treinamento que foge de um cenário real, pois utiliza no mínimo 80% do *dataset* para treinamento e formação dos padrões comportamentais.

3.2 ABORDAGENS DE APRENDIZADO NÃO SUPERVISIONADO PARA DETECÇÃO DE ANOMALIAS DE COMPORTAMENTO

Outros trabalhos são baseados em abordagens de aprendizado não supervisionado, e portanto, são mais aplicáveis a cenários de ambiente real. Contudo, a maior parte utiliza *datasets* privados, o que dificulta a reprodução e validação dos modelos.

Jin *et al.* [25] propõem o uso da técnica *Robust Principal Component Analysis* – RPCA para identificar vazamento de informações confidenciais a partir da caracterização de padrões normais de consultas a dados por parte de usuários. Na etapa de pré-processamento, os dados de consultas são consolidados como séries temporais. A partir da decomposição das informações estatísticas em matrizes *low rank* e *sparse*, bem como agrupamento K-means, foi demonstrado ser possível identificar anomalias de comportamento quando a matriz *sparse* tem uma quantidade significativa de elementos diferentes de zero. Foram utilizados dados simulados e reais, porém nenhum *dataset* público que permita a reprodução do estudo. Um problema dessa abordagem decorre da necessidade de se estabelecer um padrão comportamental individual para cada usuário por meio da decomposição RPCA combinada à clusterização K-means, o que pode ser impraticável em casos de grande número de usuários.

Pokhrel *et al.* [17] propõem uma solução híbrida baseada em OCSVM e Naïve Bayes – NB, cuja finalidade é identificar o mau uso de credenciais. Foram utilizados dois *datasets* proprietários, compostos de *logs* de auditoria do Windows 2008 com atividades de até 60 usuários. Inicialmente é realizada a etapa de construção de perfil de usuário, na qual a extração de *features* é realizada a partir de agregações e correlações dos dados, obtendo como resultado variáveis comportamentais, como tempo de duração de sessão e número de bloqueios de conta de cada usuário. A etapa de detecção é realizada a partir da comparação de similaridade do comportamento atual do usuário com o padrão comportamental existente. As anomalias de comportamento são detectadas quando essa comparação excede um determinado valor de limiar preestabelecido. A principal desvantagem aqui também é a necessidade de se estabelecer um perfil de comportamento individual para cada usuário, o que pode se tornar inviável em *datasets* com grande número de usuários. Além disso, como se trata de um *dataset* privado, não é possível reproduzir o estudo.

Gao *et al.* [31] apresentam o uso de clusterização para identificar usuários maliciosos usando *logs* de servidores web da *Beijing University of Posts and Telecommunications*. A partir das transformações aplicadas sobre os *logs*, são criadas sequências descritivas dos comportamentos de cada usuário e consolidadas em matrizes contendo os *eigenvalues* das sequências. Calculando-se a entropia das matrizes dos usuários, é possível estabelecer um padrão comportamental. A detecção e anomalias ocorre com a utilização da clusterização K-means, que considera a soma das distâncias para análise dos *clusters*. Essa abordagem apresenta como desvantagem a necessidade de um novo treinamento para cada nova entrada de *log*, uma vez que a detecção se baseia em algoritmo de clusterização.

Prarthana *et al.* [21] abordam o aspecto multidimensional dos *logs* de segurança com base no *Online Analytical Processing* – OLAP. Para tanto, inicialmente ocorre a etapa de fontes de dados, onde são coletados os *logs* de diferentes fontes, dando sequência à preparação dos dados, composta de agregações, transformações, filtros e limpeza. Por fim, ocorre a etapa de análise de dados, que utiliza algoritmos de clusterização para análise multidimensional das *features* por meio dos cubos OLAP, aplicando testes

por meio dos algoritmos DBSCAN, KS-test e Peacock 2D e 3D. Os autores evidenciam que os melhores resultados são obtidos trabalhando-se com maior número de dimensões. Entretanto, o trabalho possui como limitação um alto custo computacional por utilizar testes estatísticos que dependem da modelagem OLAP. Os próprios autores ressaltam que o acréscimo de dimensões e execuções aumenta o tempo exponencialmente. Assim, a limitação da quantidade de dimensões analisadas dificulta avaliação de padrões comportamentais que envolvam um número alto de dimensões.

Qu *et al.* [38] endereçam o uso de autoencoder com RNN usando células do tipo *Gated Recurrent Unit* – GRU. O objetivo é propor um modelo que possa garantir um treinamento eficiente para identificar anomalias relacionadas a comportamento de usuários. A saída do *encoder* é modelada com o logaritmo da média e da variância de *m* distribuições gaussianas. Esse dado então é submetido ao *decoder* do modelo. Essa abordagem apresenta a desvantagem de não identificar unicamente os usuários que apresentam comportamento anômalo, detectando apenas dados que se distanciam significativamente da totalidade. Além disso, ainda que a abordagem tenha obtido resultados satisfatórios, a métrica utilizada para análise do desempenho foi a acurácia, que é inadequada para indicar efetividade de modelos utilizados para a detecção de anomalias, pois desconsidera que os dados são desbalanceados. Nesses casos, as métricas Recall e ROC-AUC podem melhor indicar a qualidade do modelo obtido.

3.3 DISCUSSÃO SOBRE OS TRABALHOS

A Tabela 3.1 apresenta um comparativo entre os trabalhos citados no que se refere aos principais fatores diferenciais presentes no WEAPON, listado na última linha.

Tabela 3.1: Comparativo entre os trabalhos citados.

Autores	**Aprendizado**		**Técnica**		**Detecção Abrangente**	**Individualiza Usuários**	**Escalável**
	supervisionado	**não supervisionado**	**shallow learning**	**deep learning**			
Zhang *et al.* [37]	Sim	Não	Não	Sim	Não	Sim	Sim
Kim *et al.* [24]	Sim	Não	Sim	Não	Não	Sim	Sim
Mohammed *et al.* [39]	Sim	Não	Sim	Não	Não	Sim	Sim
Jin *et al.* [25]	Não	Sim	Sim	Não	Sim	Sim	Não
Pokhrel *et al.* [17]	Não	Sim	Sim	Não	Sim	Sim	Não
Gao *et al.* [31]	Não	Sim	Sim	Não	Sim	Sim	Não
Prarthana *et al.* [21]	Não	Sim	Sim	Não	Sim	Sim	Não
Qu *et al.* [38]	Não	Sim	Não	Sim	Sim	Não	Sim
WEAPON	Não	**Sim**	Não	**Sim**	**Sim**	**Sim**	**Sim**

A partir dos trabalhos citados, nota-se que a utilização de aprendizado supervisionado acaba por dificultar a utilização desses modelos em cenários reais. Isso ocorre pelas seguintes razões:

a) Necessidade da quase totalidade dos dados para realizar o treinamento do modelo, algo impeditivo no cenário real em que o perfil comportamental considerado normal deve ser estabelecido em um horizonte temporal factível para se promover a detecção de anomalias o mais rápido possível;

b) Devido ao grande desbalanceamento dos *datasets* quando se trata de detecção de anomalias, treinar os modelos com os poucos eventos anômalos implica em grande possibilidade de que o modelo não

seja capaz de detectar anomalias de comportamento muito diferentes daquelas existentes durante o treinamento, ou seja, não possibilita uma detecção de anomalias abrangente; e

c) Como os *datasets* públicos disponibilizados endereçam um foco de estudo, as anomalias são restritas àquele problema, como é o caso do CERT *Dataset*. Com isso, anomalias de comportamento que são de interesse para um analista de segurança e deveriam ser detectadas, são rotuladas como dados normais, o que também resulta em modelos cuja detecção de anomalias de comportamento não é abrangente, ou seja, menos capazes de generalizar.

Por outro lado, quando se trata das abordagens de aprendizado não supervisionado, observa-se que os trabalhos são mais suscetíveis de implantação em cenários reais. Nesse sentido, verificamos que em geral eles abordam problemas de ambientes reais, aplicando preparação de dados e extração de *features* para criar variáveis comportamentais, e em seguida, utilizam técnicas de AM para estabelecer padrões comportamentais e possibilitar a detecção de anomalias de comportamento. Como regra geral, a detecção de anomalias ocorre a partir da verificação de similaridade a partir de algum tipo de cálculo de distância, aplicando-se um limiar que separa as classes de eventos normais e de anomalias.

Os modelos que utilizam técnicas de *shallow learning*, como RPCA, OCSVM e K-means, requerem a criação de perfis individuais para cada usuário, o que pode ser computacionalmente inviável em ambientes com milhares de usuários. Por outro lado, modelos que usam técnicas de redes neurais (*deep learning*), permitem um rápido aprendizado, porém desconsideram a individualidade de cada usuário, gerando modelos genéricos, que detectam anomalias de comportamento em função da totalidade dos dados, mas pouco relacionados com a individualidade de cada usuário.

Outro aspecto relevante que se verifica a partir dos trabalhos é a dificuldade de obtenção de *logs* de segurança reais contendo as atividades dos usuários [24, 42], haja vista os obstáculos quanto à proteção, privacidade e dificuldade de rotular os dados. Além disso, a obtenção de dados rotulados para o treinamento de modelos tem sido um obstáculo para a construção de modelos capazes de generalizar, particularmente na detecção de anomalias de comportamento dada a sua natureza não supervisionada. Esses aspectos impossibilitam a avaliação de diferentes modelos sobre um *dataset* comum de referência.

A partir dessas vantagens e limitações, a arquitetura WEAPON foi elaborada utilizando a abordagem de aprendizado não supervisionado, garantindo assim sua capacidade de generalizar. Além disso, o WEAPON utiliza um horizonte temporal curto, e portanto, factível de aplicação em cenários reais. A modelagem do WEAPON ainda possibilita a individualização dos usuários, tanto na criação do padrão comportamental considerado normal como na fase de detecção, mesmo para *datasets* com milhares de usuários como é o caso do CERT *Dataset*, pois sua arquitetura foi construída de forma a otimizar o processo de treinamento e de detecção.

4 WEAPON: UMA ARQUITETURA PARA DETECÇÃO DE ANOMALIAS DE COMPORTAMENTO DO USUÁRIO

Este capítulo apresenta o WEAPON, uma arquitetura para detecção de anomalias de comportamento de usuários baseada em *Wide and Deep Convolutional LSTM Autoencoder*. A modelagem desenvolvida para validação do WEAPON implementa uma estratégia de rotulação baseada em supervisão fraca utilizando o Snorkel [1], que caracteriza anomalias de comportamento de usuários tal como se um analista de segurança avaliasse os eventos para classificá-los. Essa combinação permitiu otimizar a eficiência da WEAPON na detecção de anomalias de comportamento do usuário.

Para facilitar o entendimento do WEAPON, este capítulo apresenta a visão geral da estratégia de detecção de anomalias e validação do modelo, o *dataset* utilizado, a estratégia de rotulação com o Snorkel, e por fim, a arquitetura do WEAPON.

4.1 VISÃO GERAL DA ESTRATÉGIA PARA A DETECÇÃO DE ANOMALIAS DE COMPORTAMENTO DE USUÁRIO COM O WEAPON

A estratégia adotada para modelagem e validação do WEAPON no tocante à detecção de anomalias de comportamento é composta de sete estágios, conforme pode ser visualizado na Figura 4.1. A primeira etapa (Ponto A) é responsável pelo pré-processamento dos *logs*, que corresponde à consolidação dos eventos dos usuários e à extração de *features*, que são as variáveis comportamentais e variáveis categóricas. Essas variáveis habilitam o processo de criação de perfis de comportamento dos usuários.

A partir do novo conjunto de dados composto de variáveis comportamentais e categóricas, a segunda etapa (Ponto B) consiste em dividi-los em treinamento e teste. O terceiro estágio (Ponto C) é responsável pelo treinamento do modelo para o aprendizado dos padrões comportamentais dos usuários, e pode ser considerada a primeira interface com o WEAPON. Em que pese o WEAPON utilizar a abordagem não supervisionada, o treinamento consiste na formação dos padrões de comportamento dos usuários que devem ser considerados normais pelo modelo, e portanto, não requer que os dados sejam rotulados.

O quarto estágio (Ponto D) é responsável pela predição de comportamento dos dados de teste, onde se busca reproduzir esses dados em função dos padrões de comportamento aprendidos no estágio anterior. Com os dados obtidos da predição, inicia-se o quinto estágio (Ponto E), que é a detecção de anomalias propriamente dita, no qual o erro de reconstrução, que é a diferença entre os dados de teste e os dados reproduzidos, é comparado com um valor de limiar, tornando possível julgar se o dado é normal ou anomalia. A arquitetura do WEAPON compreende a parte tracejada da Figura 4.1.

O sexto estágio (Ponto F) é a etapa de rotulação dos dados de teste com base em supervisão fraca, simulando as inferências de especialistas. Nessa etapa, a supervisão fraca permite que grande parte das

[1] https://www.snorkel.org

anomalias de comportamento sejam rotuladas, ou seja, aquelas anomalias de comportamento previsíveis e que seguramente seriam regras analíticas de dispositivos de segurança.

O sétimo estágio (Ponto G) ocorre a partir da comparação do conjunto de dados rotulados por supervisão fraca com os resultados de detecção de anomalias de comportamento obtidos pelo WEAPON, permitindo assim sua avaliação de desempenho.

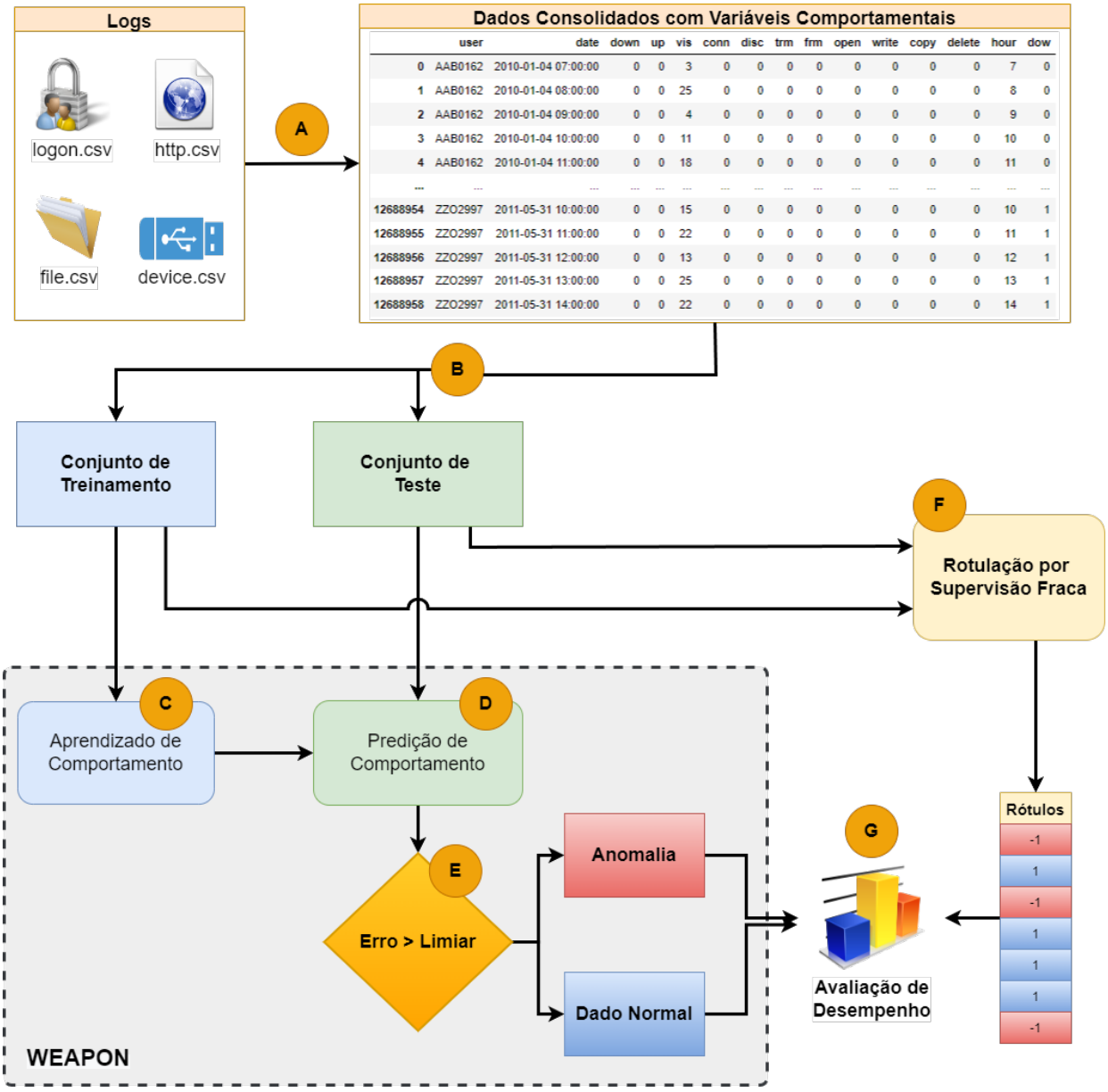


Figura 4.1: Visão Geral da Estratégia de Detecção de Anomalias com o WEAPON.

4.2 UTILIZAÇÃO DO CERT *DATASET* PARA DETECÇÃO DE ANOMALIAS DE COMPORTAMENTO DE USUÁRIO

O processo de detecção de anomalias de comportamento de usuário depende necessariamente da criação de padrões de comportamento do usuário a partir de registros de *logs* eventos. Dados os obstáculos inerentes à proteção de dados, privacidade e dificuldade em rotular dados, a obtenção desses *logs* é uma tarefa desafiadora [24, 42].

Por essa razão, nesta pesquisa foi utilizado o CERT *Dataset* [58, 59], que é uma coleção de *datasets* sintéticos voltados para ameaças internas (*insiders*). Essa coleção se insere no contexto de uma organização cujas atividades diárias de usuários são registradas em diferentes *logs* de eventos de acordo com a ação:

a) Atividades de *logon* e *logoff*, registradas no arquivo logon.csv;

b) Uso de dispositivos removíveis, registrado nos arquivos device.csv e file.csv;

c) Navegação WEB, registrada no arquivo http.csv;

d) Uso do e-mail, registrado no arquivo email.csv; e

e) Informações de perfis, registradas no arquivo ldap.csv.

Neste trabalho utilizamos o CERT *Dataset* versão R6.2, que consiste em um conjunto de *datasets* composto de *logs* de eventos do período compreendido entre 02/01/2010 a 31/05/2011 contendo atividades de quatro mil usuários, dos quais cinco são funcionários que desempenham atividades maliciosas (*insiders*). A Tabela 4.1 apresenta um subconjunto das atividades de navegação web dos usuários contido no arquivo http.csv.

	id	date	user	pc	url	activity	content
0	{V1D3-W8BL16YA-2594OWGB}	2010-01-02 06:21:31	ANC1950	PC-4921	http://icio.us/John_Edward_Brownlee_as_Attorne...	WWW Visit	Further consultation with post-production team...
1	{C7A0-F6CK17IX-5508HGRY}	2010-01-02 06:21:42	ANC1950	PC-4921	http://babycenter.com/Manchester_SmallScale_Ex...	WWW Visit	These two populations have been observed in ma...
2	{C0X3-I5RZ16GM-1535IAMY}	2010-01-02 06:21:58	ANC1950	PC-4921	http://babycenter.com/Manchester_SmallScale_Ex...	WWW Visit	These two populations have been observed in ma...
3	{G8V6-M0CF17SO-9968THHW}	2010-01-02 06:28:04	ANC1950	PC-4921	http://timeanddate.com/Accurate_News_and_Infor...	WWW Visit	Cape sold the US rights to the recently formed...
4	{I9A3-Z4JU92SK-7362XXPN}	2010-01-02 06:29:26	SAB1954	PC-5091	http://timeanddate.com/Accurate_News_and_Infor...	WWW Visit	Cape sold the US rights to the recently formed...
5	{W0M2-I3IQ18JF-2903CXZO}	2010-01-02 06:37:40	JEV3132	PC-7700	http://discovery.com/Temple_of_Eshmun/awali/20...	WWW Visit	With 10:39 remaining in the third quarter, Aub...
6	{U2O3-U9GT45TQ-4462SMEY}	2010-01-02 06:37:41	JEV3132	PC-7700	http://soundcloud.com/Don_Valley_Parkway/eglin...	WWW Visit	Williams also had the most rushing touchdowns ...
7	{A8Z2-I6PF54TO-1778EYTR}	2010-01-02 06:37:46	JEV3132	PC-7700	http://soundcloud.com/Don_Valley_Parkway/eglin...	WWW Visit	Williams also had the most rushing touchdowns ...
8	{A8O4-M5AJ09ZM-3721QTUC}	2010-01-02 06:38:15	LSN1672	PC-9021	http://foodnetwork.com/History_of_the_National...	WWW Visit	Throughout the latter part of Speer's imprison...
9	{G6T6-X3MH45SJ-4376INNO}	2010-01-02 06:38:16	JEV3132	PC-7700	http://wsj.com/Blackburn_Olympic_FC/etonian/20...	WWW Visit	Due to the controversy surrounding Auburn's fa...

Tabela 4.1: *Logs* de navegação web contidos no arquivo http.csv

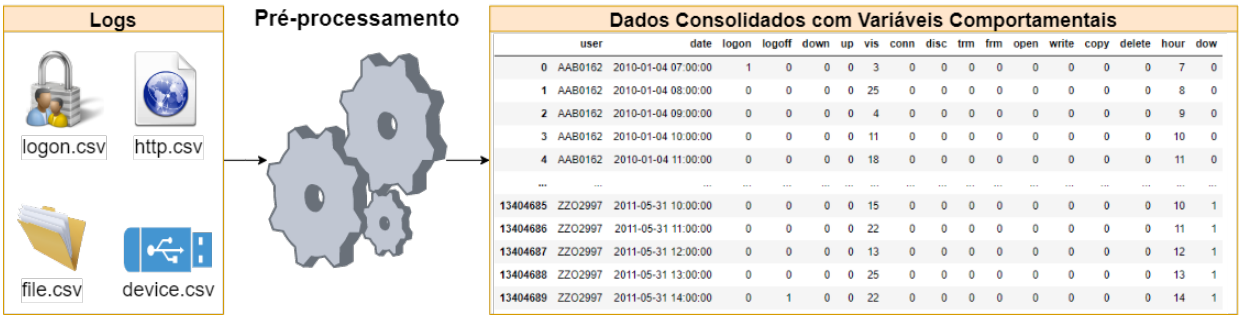
O objetivo central do CERT *Dataset* é fomentar estudos sobre atores maliciosos dentro das organizações. Portanto, o conjunto de anomalias rotuladas do CERT *Dataset* é voltado apenas para os eventos sabidamente maliciosos. Por outro lado, situações de anomalias de comportamento mais abrangentes são

rotuladas como normais no CERT *Dataset*, a exemplo de um funcionário que não tem hábitos de trabalhar fora do expediente e nos fins de semana, mas que subitamente passa a desenvolver atividades nesses horários anômalos conforme seu padrão de comportamento anterior.

É certo que diante dos recentes episódios de ataques cibernéticos, cada vez mais elaborados e complexos, tais modificações comportamentais precisam ser avaliadas, e despertam o interesse de análise dos analistas de segurança. Além disso, o objetivo central deste trabalho é proporcionar um modelo abrangente para a detecção de anomalias de comportamento de usuários. Por essas razões, estendemos o uso do CERT *Dataset* para além do seu propósito original de encontrar *insiders*. No cenário deste estudo, o CERT *Dataset* foi utilizado para identificar padrões de comportamento dos usuários, bem como desvios significativos no comportamento que indicam comportamento anômalo. Isso simplificou o processo de identificação de fontes de comportamento de usuários, e permite que futuros estudos possam ser padronizados, reproduzidos e comparados a partir de um *dataset* em comum, independentemente de uma rotulação prévia.

4.2.1 Pré-processamento dos dados e extração de *features*

Uma estapa crucial em todo projeto de AM refere-se ao pré-processamento e extração de *features* dos dados originais, para então serem submetidos ao modelo de AM desenvolvido. Nesse sentido, considerando o problema de detecção de anomalias de comportamento de usuários e os trabalhos relacionados, observamos que as metodologias de pré-processamento e extração de *featrures* dos trabalhos [17, 21, 24, 25] são convenientes para o CERT *Dataset*. Nesses trabalhos, os dados pré-processados representam consolidações das atividades dos usuários como resumos em intervalos de tempo fixos, caracterizando variáveis comportamentais. Assim, a etapa de pré-processamento dos dados e extração de *features* para a construção do *dataset* final com as variáveis comportamentais é ilustrada na Figura 4.2.



	user	date	logon	logoff	down	up	vis	conn	disc	trm	frm	open	write	copy	delete	hour	dow
0	AAB0162	2010-01-04 07:00:00	1	0	0	0	3	0	0	0	0	0	0	0	0	7	0
1	AAB0162	2010-01-04 08:00:00	0	0	0	0	25	0	0	0	0	0	0	0	0	8	0
2	AAB0162	2010-01-04 09:00:00	0	0	0	0	4	0	0	0	0	0	0	0	0	9	0
3	AAB0162	2010-01-04 10:00:00	0	0	0	0	11	0	0	0	0	0	0	0	0	10	0
4	AAB0162	2010-01-04 11:00:00	0	0	0	0	18	0	0	0	0	0	0	0	0	11	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
13404685	ZZO2997	2011-05-31 10:00:00	0	0	0	0	15	0	0	0	0	0	0	0	0	10	1
13404686	ZZO2997	2011-05-31 11:00:00	0	0	0	0	22	0	0	0	0	0	0	0	0	11	1
13404687	ZZO2997	2011-05-31 12:00:00	0	0	0	0	13	0	0	0	0	0	0	0	0	12	1
13404688	ZZO2997	2011-05-31 13:00:00	0	0	0	0	25	0	0	0	0	0	0	0	0	13	1
13404689	ZZO2997	2011-05-31 14:00:00	0	1	0	0	22	0	0	0	0	0	0	0	0	14	1

Figura 4.2: Etapa de Pré-processamento dos dados e extração de *features*

Os *logs* contidos nos arquivos logon.csv, device.csv, file.csv e http.csv foram utilizados para modelar o perfil dos usuários, uma vez que buscamos alterações de comportamento relacionadas às atividades de *logon*, navegação *web* e uso de dispositivo removível. O pré-processamento e extração de *features* consiste em uma agregação temporal dos *logs* supracitados, considerando o intervalo de uma hora. Essa agregação constrói um novo *dataset*, composto de treze variáveis comportamentais que descrevem resumos das atividades dos usuários a cada hora. São incluídos também no novo *dataset* as informações de código de usuário, data, hora e dia da semana, que são dados categóricos, relevantes para associar as variáveis

comportamentais aos hábitos de utilização de cada usuário, como atividades fora do expediente ou fins de semana.

A Tabela 4.2 apresenta as variáveis comportamentais e categóricas, que foram consolidadas em resumos das atividades de cada usuário a cada hora. Esse conjunto possibilita mensurar a quantidade de ações do usuário nos intervalos de uma hora, e assim, traçar um perfil vinculado à hora e dia da semana.

Tabela 4.2: *Features* extraídas do pré-processamento.

Feature	**Descrição**
user	Código do usuário
date	Data e hora da consolidação
logon	Quantidade de logons no computador durante a hora
logoff	Quantidade de logoffs no computador durante a hora
down	Quantidade de downloads durante a hora
up	Quantidade de uploads durante a hora
vis	Quantidade de visitas web durante a hora
conn	Quantidade de conexões de dispositivo removível durante a hora
disc	Quantidade de desconexões de dispositivo removível durante a hora
trm	Quantidade de ações executadas com destino no dispositivo removível durante a hora
frm	Quantidade de ações executadas com origem no dispositivo removível durante a hora
open	Quantidade de abertura de arquivos no dispositivo removível durante a hora
write	Quantidade de escritas de arquivo no dispositivo removível durante a hora
copy	Quantidade de cópias de arquivos no dispositivo removível durante a hora
delete	Quantidade de deleções de arquivos no dispositivo removível durante a hora
hour	Hora, variando de 0 a 23
dow	Dia da Semana, variando de 0 (segunda) a 6 (domingo)

Antes das etapas de treinamento para aprendizado dos comportamentos e teste para detecção de anomalias de comportamento, as variáveis comportamentais (*logon, logoff, down, up, vis, conn, disc, trm, frm, open, write, copy, delete*) são escalonadas para que seus valores residam entre zero e um, em função dos valores mínimo e máximo de cada variável no conjunto de treinamento. Isso é realizado a partir da seguinte equação, onde X_{scaled} é o valor escalonado resultante, X é o valor a ser escalonado, $X_{train-min}$ e $X_{train-max}$ são os valores mínimo e máximo da variável no conjunto de treinamento, respectivamente.

$$X_{scaled} = \frac{X - X_{train-min}}{X_{train-max} - X_{train-min}} \tag{4.1}$$

Por sua vez, as variáveis categóricas *user, dow, hour* são convertidas para matrizes binárias, procedimento conhecido como *One Hot Encoding*[2].

O algoritmo desenvolvido para a etapa de pré-processamento está disponível no ANEXO I.

[2]Codifica dados categóricos no formato de matrizes binárias.

4.2.2 Seleção dos conjuntos de treinamento e teste

O processo de detecção de anomalias de comportamento de usuários depende da criação de padrões de comportamento de usuário a partir dos registros históricos de suas atividades nos *logs* de eventos. Naturalmente, a criação de um padrão de comportamento requer que se estabeleça um período de tempo de análise para que o conjunto de atividades dos usuários possa caracterizar um padrão. A definição desse período de tempo é uma tarefa subjetiva a critério do analista, mas cuja escolha influencia diretamente na detecção de anomalias. Um período muito curto pode representar um conjunto insuficiente de ações do usuário para formar um perfil, gerando assim muitos falso positivos. Já um período muito longo pode incorrer na incorporação de padrões anômalos juntos aos dados de treinamento, e assim gerar falso negativos, além de possivelmente não atender à necessidade da organização de se estabelecer uma resposta rápida às ameaças cibernéticas.

Uma vez construído o padrão de comportamento, a detecção de anomalias pode ocorrer a partir da comparação dos eventos novos com o padrão construído. Contudo, esse padrão de comportamento construído deve ter um prazo de validade, uma vez que usuários podem mudar seus hábitos, seja em virtude de novos cargos, ou mesmo por incorporação de novas rotinas de trabalho. Desse modo, a escolha do prazo de validade de um padrão de comportamento para a detecção de anomalias também é uma tarefa subjetiva a cargo do analista.

Diante desse contexto, a partir da análise do CERT *Dataset* e considerando horizontes temporais factíveis a um cenário real, foram analisados diferentes períodos para construção do perfil comportamental e para validade desse perfil da detecção de anomalias. Dentre esses períodos, escolheu-se utilizar quatro meses de dados na etapa de pré-processamento para o treinamento do modelo de modo que os padrões comportamentais dos usuários pudessem ser aprendidos. Isso representa o período entre 02/01/2010 a 02/05/2010, aproximadamente $1/4$ dos dados do CERT *Dataset*. O WEAPON utiliza esses dados como aprendizado auto-supervisionado durante o treinamento, pois seu autoencoder converge na seleção dos melhores pesos capazes de reconstruir os dados de entrada. Quanto ao período de validade dos padrões comportamentais, escolheu-se também o período de quatro meses de dados, subsequentes ao conjunto de treinamento, para formar o conjunto de teste, compreendendo o período entre 03/05/2010 a 02/09/2010 para a validação do WEAPON na detecção de anomalias de comportamento de usuário. A Figura 4.3 ilustra a divisão dos dados entre os conjuntos de treinamento (formação do padrão de comportamento) e teste (detecção de anomalias).

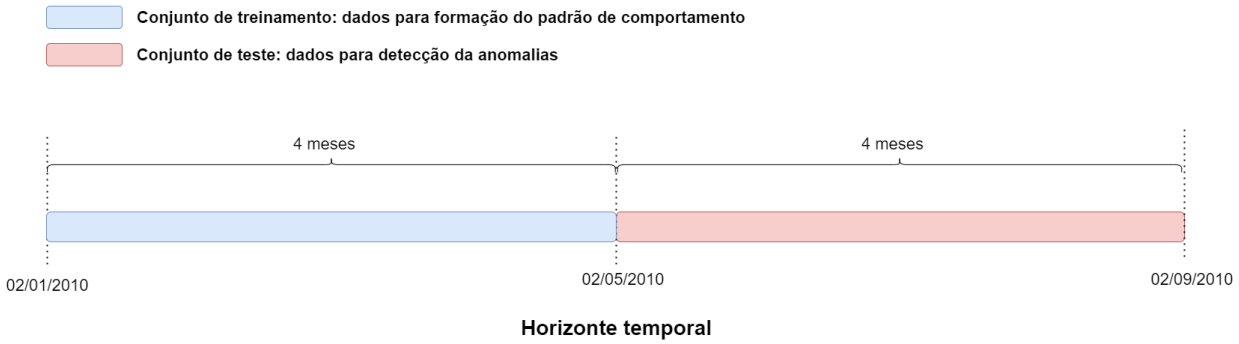


Figura 4.3: Conjuntos de treinamento e teste.

4.3 ROTULAÇÃO DOS DADOS PARA VALIDAÇÃO DO WEAPON

Conforme apresentado na Seção 4.2, o CERT *Dataset* teve seu uso estendido para englobar anomalias de comportamento de usuários de forma mais abrangente em comparação à proposta inicial de detecção de *insiders*. Para tanto, é necessário inicialmente estabelecer quais parâmetros dos padrões comportamentais influenciam o processo de detecção de anomalias. Sobre esse aspecto, enfatizamos que em um cenário real, avaliar a efetividade de um modelo de detecção de anomalias é uma tarefa desafiadora uma vez que as anomalias são eventos de natureza rara, geralmente os dados não estão rotulados, e o critério de julgamento acerca de uma anomalia é algo subjetivo a critério do analista [37].

4.3.1 Padrões de anomalias de comportamento de usuário

Conforme mencionado na Seção 4.2.2, a formação do padrão comportamental dos usuários que servirá de referência para detecção de anomalias de comportamento requer a definição de um horizonte temporal, que no caso são os quatro primeiros meses do *dataset*. A partir desse padrão comportamental, torna-se possível a detecção de anomalias de comportamento no conjunto de teste, que compreende os quatro meses seguintes, tendo em vista que os padrões comportamentais possuem uma validade.

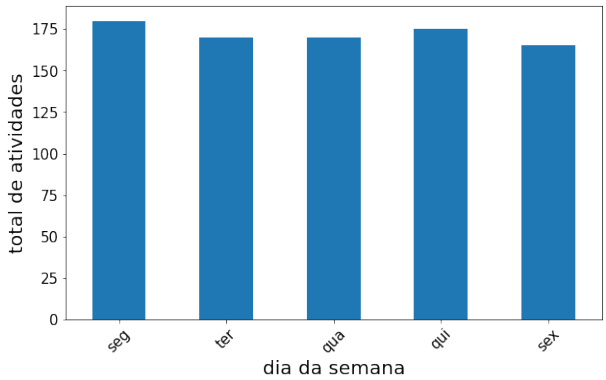
No contexto dessa pesquisa, cujo foco é a detecção de anomalias de comportamento de forma abrangente, mudanças de comportamento significativas encontradas no conjunto de teste em comparação com o padrão comportamental definido no conjunto de treinamento são indicativos de anomalias a serem detectadas. Nesse sentido, a Figure 4.4 apresenta um exemplo dessa mudança significativa de comportamento.

O padrão de comportamento extraído do conjunto de treinamento demonstra um usuário que tem sua rotina de trabalho apenas em dias úteis (Figura 4.4a), entre 7h e 17h (Figura 4.4c), e cuja navegação na web teve no máximo 9 visitas no intervalo de uma hora (valor máximo do boxplot na Figura 4.4e). Já no conjunto de teste, é possível observar as mudanças significativas de comportamento, pois o usuário passou a trabalhar aos sábados (Figura 4.4b), em horáros fora do expediente (Figura 4.4d), e com até 29 visitas web no intervalo de uma hora (valor máximo do boxplot na Figura 4.4f). Estas situações de mudança significativa de comportamento são consideradas anomalias de comportamento do usuário e, consequentemente, são objeto de detecção pelo WEAPON, ainda que no propósito original do CERT *Dataset* sejam consideradas situações rotuladas como normais.

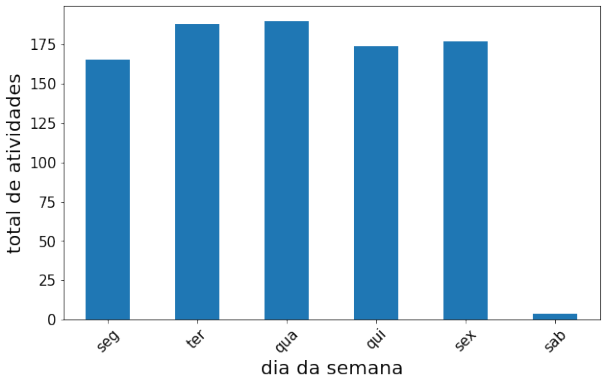
4.3.2 Rotulação de anomalias de comportamento de usuário utilizando supervisão fraca com o Snorkel

Para avaliar o desempenho do WEAPON quanto à detecção de anomalias de comportamento de usuários, é necessário que os dados do conjunto de teste sejam rotulados como normais ou anômalos de acordo com a análise dos eventos em função dos padrões de comportamento estabelecidos no conjunto de treinamento.

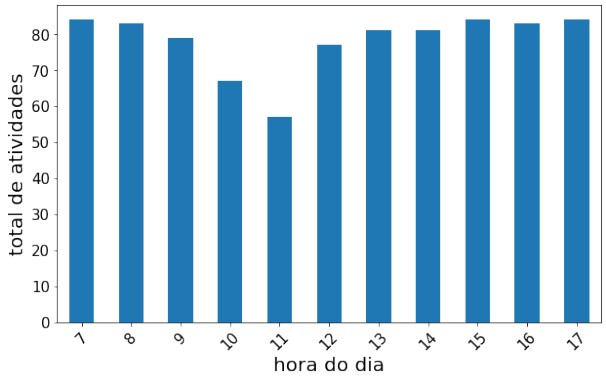
A obtenção de dados rotulados para o treinamento de modelos representa um grande obstáculo para a construção de modelos, particularmente na detecção de anomalias, dada a natureza não supervisionada



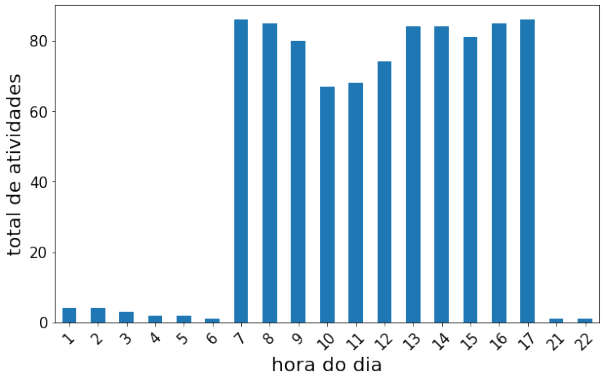
(a) Padrão de atividades do usuário em dias da semana no conjunto de treinamento.



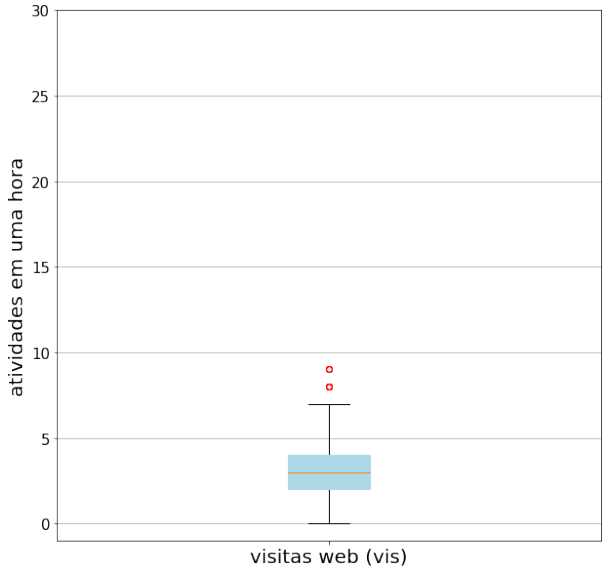
(b) Padrão de atividades do usuário em dias da semana no conjunto de teste.



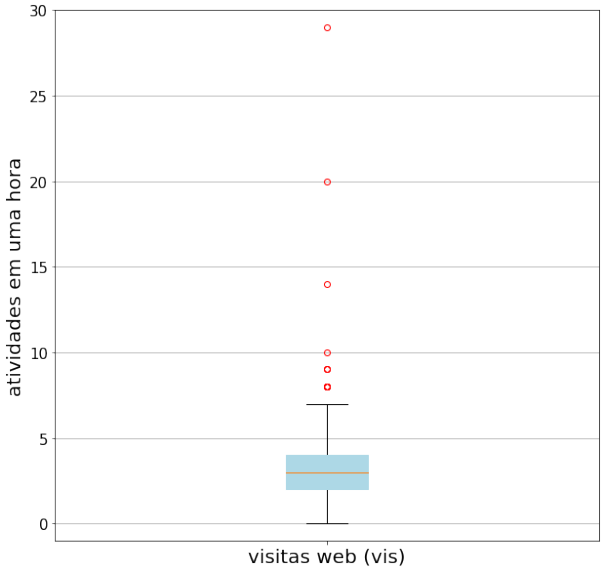
(c) Padrão de atividades do usuário em horários do dia no conjunto de treinamento.



(d) Padrão de atividades do usuário em horários do dia no conjunto de teste.



(e) Boxplot do padrão de visitas web durante o intervalo de uma hora no conjunto de treinamento.



(f) Boxplot do padrão de visitas web durante o intervalo de uma hora no conjunto de teste.

Figura 4.4: Mudanças de comportamento de usuário entre os conjuntos de treinamento e teste.

do problema de detecção de anomalias [3]. Normalmente, são necessários especialistas no domínio do conhecimento para a rotulação dos dados, o que pode levar semanas ou meses para rotular uma quantidade insignificante de dados. Mesmo com especialistas no domínio, às vezes a rotulação de dados pode ser

ambígua, a ponto de não haver um consenso sobre determinado evento.

Neste trabalho, a classificação do comportamento anômalo depende basicamente de uma mensuração da quantidade de desvio de comportamento, o que é algo subjetivo. Portanto, não seria surpreendente se dois especialistas tivessem opiniões diferentes sobre um desvio específico. Para contornar essas limitações, neste trabalho a rotulação dos dados de teste foi realizada por intermédio do Snorkel [42].

O Snorkel tem se destacado tanto na comunidade científica como em grandes organizações por sua capacidade de rotular *datasets* com base em uma supervisão fraca [3]. Para isso, o Snorkel conta com funções de rotulação (*Labeling Functions* – LFs), que são regras programáticas ruidosas que atribuem rótulos a dados não rotulados. Por LFs ruidosas subentende-se regras que não têm boa precisão, que não devem rotular necessariamente todos os dados, que podem às vezes se sobrepor a outras regras, e ainda podem atribuir rótulos diferentes aos mesmos dados. Para fornecer uma rotulação final para os dados, o Snorkel implementa um modelo de rotulação probabilístico denominado *Label Model*, que analisa todos os rótulos canditados obtidos das LFs e então define uma rotulação final com base na matriz de covariância inversa generalizada da árvore de junção do grafo de dependência das LFs [60].

A Figura 4.5 apresenta o processo de rotulação dos dados de teste utilizado para avaliação de desempenho do WEAPON. No primeiro estágio (Ponto A), os padrões de comportamento de cada usuário são construídos com base no conjunto de treinamento, para criar uma tabela de referência contendo os valores de boxplot para cada *feature* e cada usuário. No segundo estágio (Ponto B), foram construídas dezesseis LFs que são aplicadas ao conjunto de teste para então obter uma matriz com dezesseis rótulos candidatos para cada evento do conjunto de teste. Cada LF compara todos os dados do conjunto de teste com os padrões de comportamento do boxplot presentes na tabela de referência para encontrar possíveis mudanças significativas de comportamento de cada usuário, para então definir um rótulo para cada evento com base na LF em questão. No terceiro estágio (Ponto C), é aplicado o modelo de rotulação *Label Model* do Snorkel para a obtenção da rotulação final do conjunto de teste (Ponto D).

As regras de rotulação baseadas em supervisão fraca definidas nas LFs têm por objetivo representar a visão de um especialista de segurança que busca as alterações comportamentais previsíveis. Portanto, não se pode afirmar que a rotulação obtida contempla todos as possíveis situações de anomalias de comportamento de usuário. Todavia, ainda que a rotulação obtida não possa ser considerada perfeita (*gold labels*), ela representa uma forma de avaliação de desempenho da detecção de anomalias de comportamento do WEAPON.

A Figura 4.6 ilustra o processo de criação de uma LF para a detecção de anomalias relacionadas à quantidade de *downloads* do conjunto de teste em função do perfil de comportamento do conjunto de treinamento para um determinado usuário. A análise é realizada a partir do valor de marcador *whisker* superior do boxplot sobre as atividades de *download* do usuário no conjunto de treinamento (Figura 4.6a). Caso, no conjunto de teste, existam eventos com quantidade de downloads superior ao valor desse marcador, esses eventos serão rotulados como anomalia pela LF (Figura 4.6b). Essa rotulação será combinada com todas as outras rotulações obtidas das LFs restantes para a formação da rotulação final a partir do *Label Model* do Snorkel. O algoritmo contendo as regras de formação de padrão de comportamento e as LFs utilizadas para rotular o conjunto de teste são apresentadas no Anexo II.

[3] https://www.snorkel.org/resources/

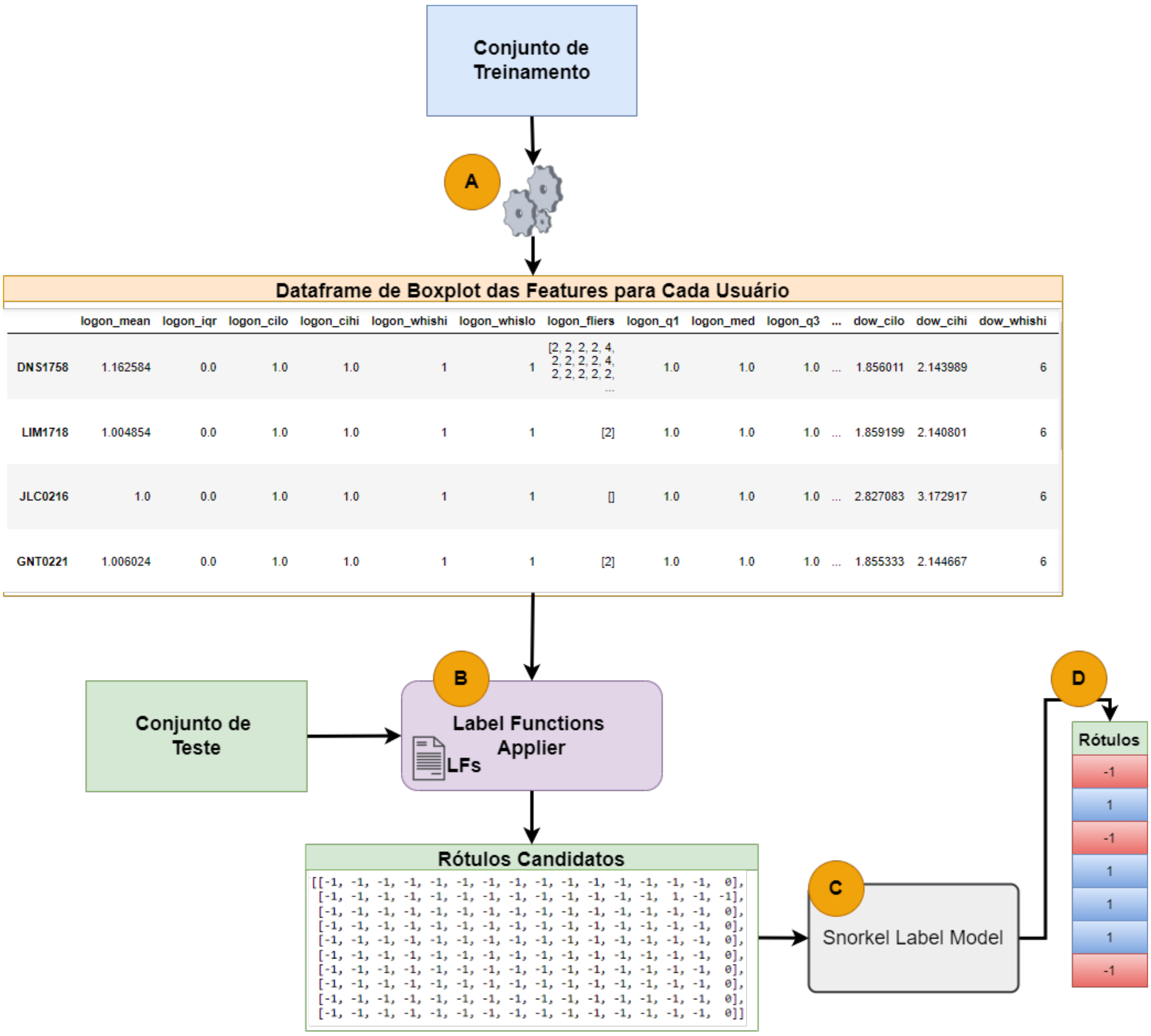
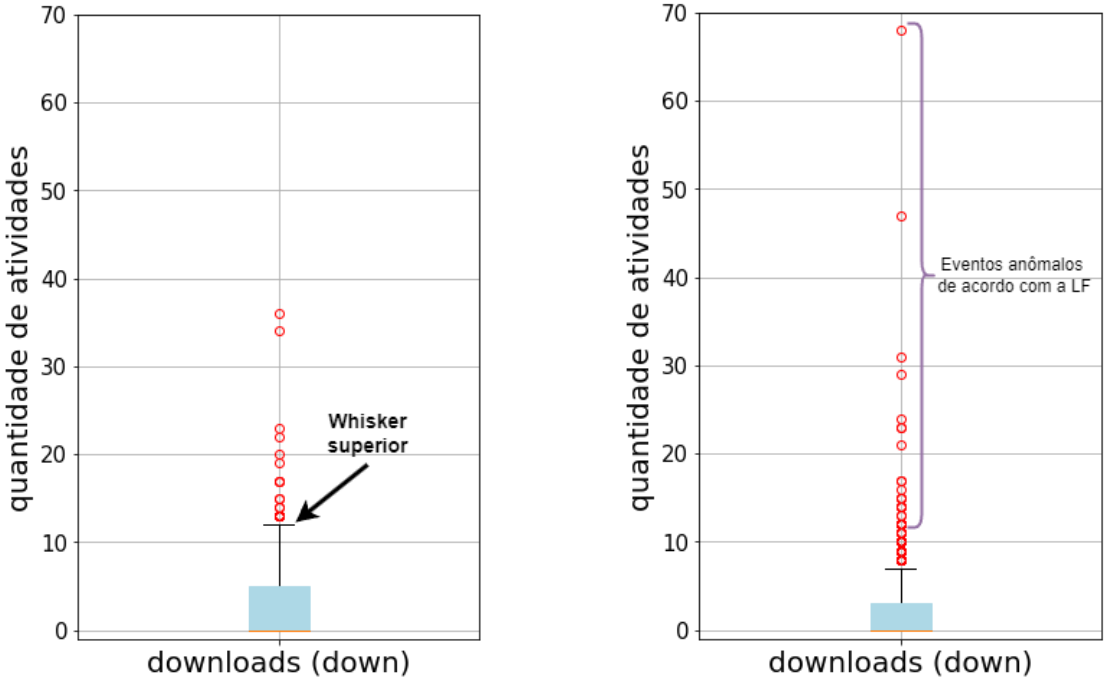


Figura 4.5: Visão Geral do Processo de Rotulação do Snorkel.

4.4 WEAPON

O WEAPON é uma arquitetura de *deep learning* baseada em aprendizado não supervisionado voltada para a detecção de anomalias de comportamento de usuário. Para isso, o WEAPON foi modelado baseado na estratégia de detecção de anomalias com autoencoders, conforme visto na Seção 2.3.3. Contudo, a arquitetura de autoencoder tradicional foi modificada.

A Figura 4.7 apresenta a arquitetura do WEAPON. O WEAPON foi modelado combinando os conceitos de arquitetura de rede neural *wide and deep* [48], autoencoder convolucional [61] e autoencoder LSTM [62]. Como pode ser visto, o WEAPON possui como entradas os dados obtidos das *features* do pré-processamento, os quais são divididos em duas partes: uma parte para a entrada *deep*, compostas das variáveis comportamentais, e outra entrada *wide*, composta dos dados categóricos relativos a hora e dia da semana e identificação do usuário. Essa modificação utiliza o conceito de arquitetura *Wide and Deep* apresentado na Seção 2.1.2. Essa construção permitiu que a vinculação das variáveis comportamentais aos hábitos de utilização de cada usuário durante o treinamento fosse diretamente correlacionada à reconstrução dos dados no autoencoder, associando assim os comportamentos aos respectivos usuários e seus



(a) Boxplot do padrão de comportamento de atividades de *download* de um usuário no conjunto de treinamento.

(b) Boxplot do comportamento de atividades de *download* apresentado pelo usuário no conjunto de teste.

Figura 4.6: Parâmetros de construção da LF para detecção de anomalias relacionadas à quantidade de downloads no conjunto de teste em função do padrão de comportamento demonstrado no conjunto de treinamento. A definição do *whisker* superior define o valor a partir do qual os eventos do conjunto de teste serão declarados como anomalias pela LF.

horários de atividade.

O funcionamento do WEAPON segue a lógica de um autoencoder tradicional. Na primeira etapa da detecção de anomalias de comportamento com o WEAPON, o *encoder* recebe como entrada os dados denominados *deep* (Ponto A), que são as váriaveis comportamentais, compostas de treze *features* obtidas do pré-processamento. A estrutura interna do *encoder* combina uma convolução temporal com redes neurais recorrentes do tipo LSTM. A camada convolucional aprende cento e trinta *kernels*, retornando dez filtros, onde cada filtro de saída é a média de seus treze kernels. Na sequência, as duas camadas LSTM reduzem progressivamente o espaço dimensional a partir da representação latente das características de comportamento para apenas sete dimensões (Ponto B), mantendo as características de memória de curto e longo prazo devido ao tipo de construção das camadas LSTM.

Já o *decoder* recebe os dados de representação latente provenientes do *encoder* e os combinam com a entrada *wide* (Ponto C), que é composta das *features* categóricas representando informações de usuário e de instante de tempo (hora e dia da semana). Um aspecto fundamental para a eficiência do WEAPON é que essas *features* categóricas são matrizes esparsas [4] após a função de codificação *One Hot Encoder*, e com isso, poupam recursos computacionais já que requerem pequena quantidade de memória, e muitos algoritmos de ML já são preparados para otimizarem esse tipo de estrutura de dado. Com isso, o WE-

[4]Matriz na qual a maior parte dos elementos é zero, permitindo sua representação em uma estrutura de dados reduzida que armazena apenas as posições e valores que são diferentes de zero.

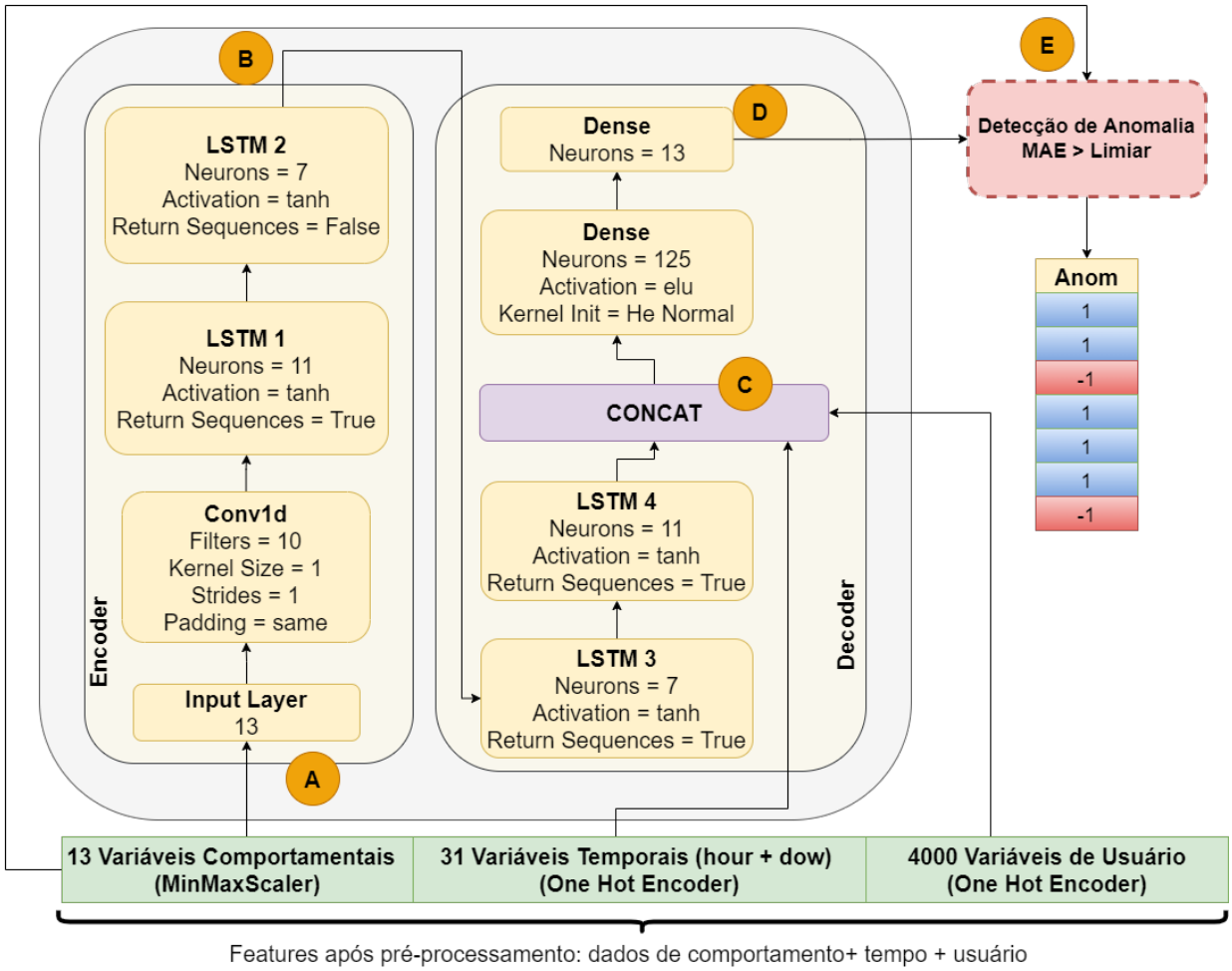


Figura 4.7: Arquitetura do WEAPON.

APON é capaz de processar uma grande quantidade de usuários de forma eficiente, o que lhe assegura escalabilidade para muitos usuários.

A estrutura do *decoder* é composta de duas camadas LSTM reversas às do *encoder*, cuja função é restaurar a dimensionalidade para onze *features*. Uma camada de concatenação realiza a combinação das *features* comportamentais advindas das camadas LSTM com as informações categóricas obtidas da entrada *wide*. Após a concatenação, uma camada neural densa com ativação *Exponencial Linear Unit* (ELU) é encarregada de reduzir a nova dimensionalidade para 125 *features*, seguida de uma camada densa de treze neurônios responsável pela reconstrução das variáveis comportamentais da entrada (Ponto D).

A saída do *decoder* do WEAPON é comparada com as váriaveis comportamentais da entrada (Ponto E), o que permite avaliar o erro de reconstrução segundo a métrica de erro médio absoluto. O erro médio absoluto quantifica o erro de reconstrução dos dados de teste, e é definido por:

$$mae = (\frac{1}{n}) \sum_{i=1}^{n} |y_i - x_i| \tag{4.2}$$

A partir da aplicação de um limiar, o WEAPON está habilitado a realizar a detecção de anomalias de

comportamento.

Para que o WEAPON aprenda os padrões comportamentais que devem ser considerados normais, ele utiliza o conjunto de dados de treinamento como aprendizado auto-supervisionado, de forma que os melhores pesos das camadas de redes neurais sejam selecionados para que sua estrutura de autoencoder seja capaz de reconstruir os dados de entrada com o menor erro possível. Para o ajuste dos melhores pesos de cada camada de sua estrutura de autoencoder, o algoritmo de *backpropagation* está configurado para minimizar o erro médio absoluto (Equação 4.2) da reconstrução. Além disso, o modelo foi configurado para utilizar o otimizador Nadam [63], que combina os otimizadores Adam e Nesterov, atualizando a taxa de aprendizado de forma adaptativa conforme o algoritmo converge para a melhor solução.

O treinamento do WEAPON é realizado de forma rápida, utilizando apenas 10 épocas, com um tamanho de *batch* de 1024, ou seja, a cada 1024 eventos dos dados pré-processados, os pesos são ajustados dentro de uma época de acordo com a taxa de aprendizado adaptativa obtida do otimizador Nadam.

5 AVALIAÇÃO DE DESEMPENHO

Este capítulo apresenta a validação do WEAPON em comparação com outros modelos para detecção de anomalias de comportamento. Para isso, apresentaremos a seguir a configuração dos experimentos, os resultados e sua discussão.

5.1 CONFIGURAÇÃO DO EXPERIMENTO

Além do WEAPON, apresentado na Seção 4.4, esta pesquisa investigou outras possíveis arquiteturas para detecção de anomalias de comportamento. No desenvolvimento dos modelos, um dos requisitos para viabilizar a construção de perfis de comportamento vinculados aos usuários, mesmo que em grande número, foi trabalhar com matrizes esparsas. Esse requisito decorre do fato de codificarmos a informação dos usuários em matrizes com células binárias, técnica conhecida como *one hot encoding*, cada coluna correspondendo a um único usuário, de modo que cada linha de dado pré-processado possuirá apenas a coluna correspondente ao usuário com o valor "1", enquanto as colunas restantes terão o valor "0".

Trabalhar com matrizes esparsas demonstrou ser um grande trunfo do WEAPON, pois permite que, com menor quantidade de memória, o modelo possa aprender comportamento de milhares de usuários, como validado no CERT *Dataset*. Além disso, grande parte dos algoritmos utilizados, em especial os de *deep learning*, são otimizados para trabalhar com matrizes esparsas, favorecendo seu uso.

Assim, os seguintes modelos foram escolhidos para serem comparados com o WEAPON, pois permitem seu uso com requisitos de tempo e memória aceitáveis:

a) *Linear One-Class* SVM utilizando *Stochastic Gradient Descent* (OCSVM-SGD);

b) *Isolation Forest* (IF);

c) *Stacked Autoencoder with Batch Normalization* (SAEBN); e

d) *Wide and Deep Stacked Autoencoder with Batch Normalization* (WDSAEBN)

A manipulação dos dados, desde sua obtenção a partir dos *logs* de eventos e subsequente pré-processamento, até a submissão para os modelos, foi realizada por meio das bibliotecas Pandas [64], Scikit-Learn [53], NumPy [65] e TensorFlow Data API [66]. Para a rotulação dos dados do conjunto de teste, foram utilizadas as bibliotecas PyTorch [67] and Snorkel [42]. Já a construção dos modelos baseados em *shallow learning*, OCSVM-SGD e IF, utilizou a biblioteca Scikit-Learn. Os modelos baseados em *deep learning* utilizaram as bibliotecas TensorFlow [66] e Keras [68].

A busca pelos melhores hiperparâmetros foi realizada utilizando o hyperopt [69]. A partr de um conjunto de possíveis hiperparâmetros, o algorítmo executa uma busca por combinações para encontrar os

melhores hiperparâmetros para o modelo. O processo de hiperparametrização visa otimizar os parâmetros do modelo especializado utilizando o algoritmo Parzen Estimator com estrutura de árvore [70] para estimar a melhor configuração de parâmetros dentro de uma lista de possíveis combinações. É necessário definir uma variável de custo a ser minimizada pelo hyperopt. No presente estudo, como a métrica *Recall* é o valor mais importante a ser maximizado, atribuímos como variável de custo o oposto dessa métrica: $-1 \times Recall$. A listagem completa de todos os valores de hiperparâmetros configurados está disponível no ANEXO III. Os algoritmos desenvolvidos para testar os valores e encontrar as melhores soluções estão disponíveis no ANEXO IV. O código fonte desenvolvido para o desenvolvimento da pesquisa também está disponível em <https://github.com/andre-molina/weapon> sob a licença GPLv3.

O ambiente de execução para gerar os experimentos é composto de um computador com processador i7-11700, 128GB de memória RAM, placa de vídeo NVIDIA RTX-3080, plataforma Anaconda para implementação dos modelos utilizando o Python versão 3.9.7 e Jupyter Notebook.

Para a avaliação dos modelos, o conjunto de treinamento possui um total de 3.294.562 registros, e o conjunto de teste um total de 3.278.931 registros. Esses registros correspondem aos dados pré-processados dos usuários, ou seja, cada instância de dado é um resumo das atividades de cada usuário no intervalo de uma hora. A partir do processo de rotulação realizado com o Snorkel, foram obtidos um total de 6.229 eventos rotulados como anomalias.

A detecção de anomalias ocorre a partir de limiares pré-definidos, conforme apresentado na Seção 2.4. A escolha desses valores deve considerar a necessidade de encontrar o máximo de anomalias possíveis, bem como evitar um grande número de falso positivos. Assim, foram testados os valores de limiar T=1% e T=2%. Esses valores são considerados a taxa de contaminação dos dados, e correspondem a taxas de anomalias esperadas em relação ao total do conjunto de teste. Essas taxas são largamente utilizadas em algoritmos de detecção de anomalias [71].

Para a avaliação do desempenho, foram utilizadas as métricas *Recall*, *Precision*, ROC-AUC, curvas ROC e tempo de execução (*Runtime*). Aggarwal *et al.* [3] indica que a análise a partir das métricas *Precision* x *Recall*, curvas ROC e ROC-AUC tendem a permitir uma melhor compreensão da efetividade de modelos de detecção de anomalias, principalmente em *datasets* desbalanceados. Géron [2] indica que em um cenário de detecção de anomalias indesejáveis, um valor de *Recall* superior tende a indicar um melhor modelo, ainda que sob pena de menor desempenho na métrica *Precision*. Assim, no contexto dessa pesquisa, a escolha dos melhores modelos é favorecida com base nos melhores resultados das métricas *Recall* e ROC-AUC, indicando que o melhor modelo é aquele que mais encontra anomalias reais (melhor *Recall*) e possui melhor resposta de detecção de anomalias em diferentes valores de limiar (melhor ROC-AUC).

5.2 RESULTADOS OBTIDOS

Uma vez encontrados os melhores hiperparâmetros para os modelos, torna-se possível compará-los com o WEAPON. Como o modelo foi treinado com o conjunto de treinamento para formação dos padrões comportamentais dos usuários no período de quatro meses, procede-se à detecção de anomalias de comportamento no conjunto de teste, compreendendo os quatro meses subsequentes. A detecção de anomalias

ocorre quando o erro de reconstrução ultrapassa os limiares de taxa de contaminação escolhidos (T=1% e T=2%).

A Figura 5.1 apresenta o desempenho com relação à métrica *Recall* para os modelos avaliados com o WEAPON, considerando os valores de limiar T=1% e T=2%. Verifica-se que as arquiteturas de *deep learning* (SAEBN, WEAPON e WDSAEBN) têm um desempenho superior aos alcançados pelos algoritmos de *shallow learning* (OCSVM-SGD e IF), com resultados muito próximos em ambos os valores de limiar T=1% e T=2%. Além disso, o WEAPON alcançou o melhor resultado, obtendo 68,87% de *Recall* no melhor caso (T=2%), seguido ao SAEBN com 68,44% e WDSAEBN com 67,41%. Por outro lado, os modelos OCSVM-SGD e IF tiveram um desempenho ruim, alcançando no melhor caso (T=2%) 18,11% e 31,85% de *Recall*, respectivamente. Isso evidencia que os modelos baseados em *deep learning* conseguem aprender melhores representações dos padrões de comportamento dos usuários, e consequentemente, detectar mais anomalias reais em comparação aos modelos baseados em *shallow learning*.

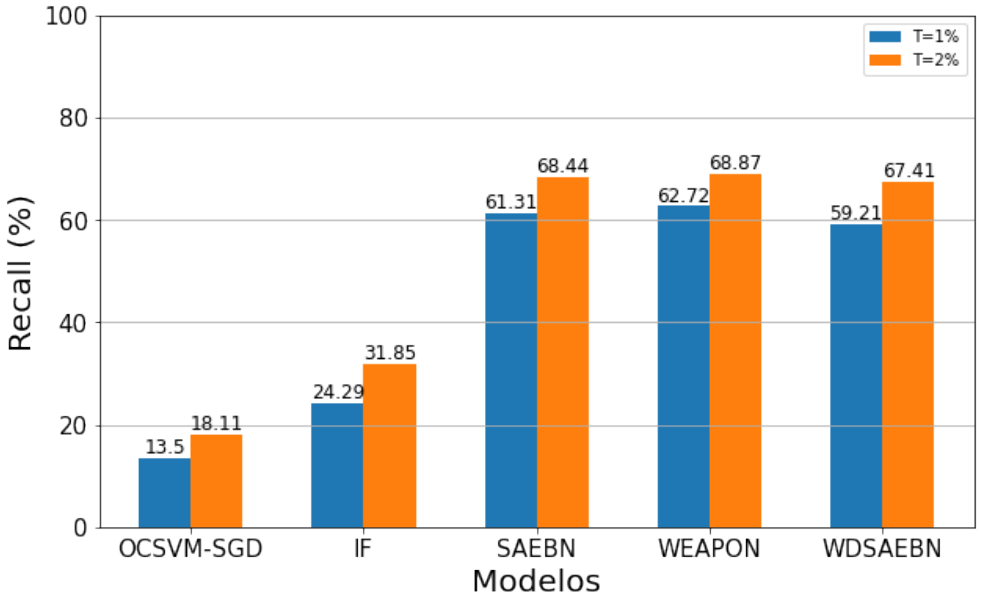


Figura 5.1: Resultados obtidos para a Métrica *Recall*.

A Figura 5.2 apresenta os resultados da métrica ROC-AUC. Pode ser observado que os modelos baseados em *deep learning* novamente se sobressaem em comparação com os modelos baseados em *shallow learning*, pois fornecem uma melhor resposta de detecção de anomalias de comportamento para diferentes valores de limiar. O WEAPON alcançou o melhor resultado, pois quando comparado com o segundo melhor modelo (SAEBN) obteve um ganho de 4,31% e, quando comparado com o pior modelo (OCSVM-SGD), obteve um ganho de 26,60%. Esses resultados demonstram que as arquiteturas de *deep learning* baseadas em autoencoders são mais adequadas à detecção de anomalias de comportamento que as arquiteturas de *shallow learning* OCSVM e IF em diferentes valores de limiar, pois conseguem aprender de forma mais intrínseca os padrões de comportamento dos usuários para compará-los na etapa de detecção.

A Figura 5.3 apresenta o desempenho da métrica *Precision* para os modelos avaliados com o WEAPON, considerando os valores de limiar T=1% e T=2%. Observa-se que o desempenho do WEAPON foi o melhor dentre todos os modelos, alcançando no melhor caso 12%, seguido do SAEBN com 11,64% e

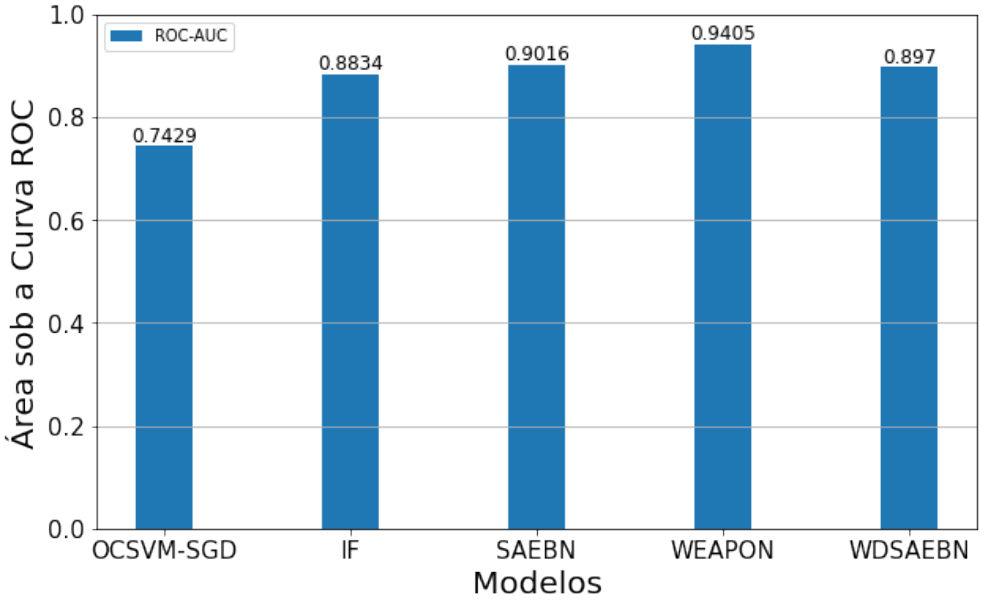


Figura 5.2: Valores de Área sob a Curva ROC (ROC-AUC) para os modelos avaliados.

WDSAEBN com 11,25%. Os modelos OCSVM-SGD e IF tiveram os piores resultados novamente, com 7,23% e 4,68% no melhor caso, respectivamente.

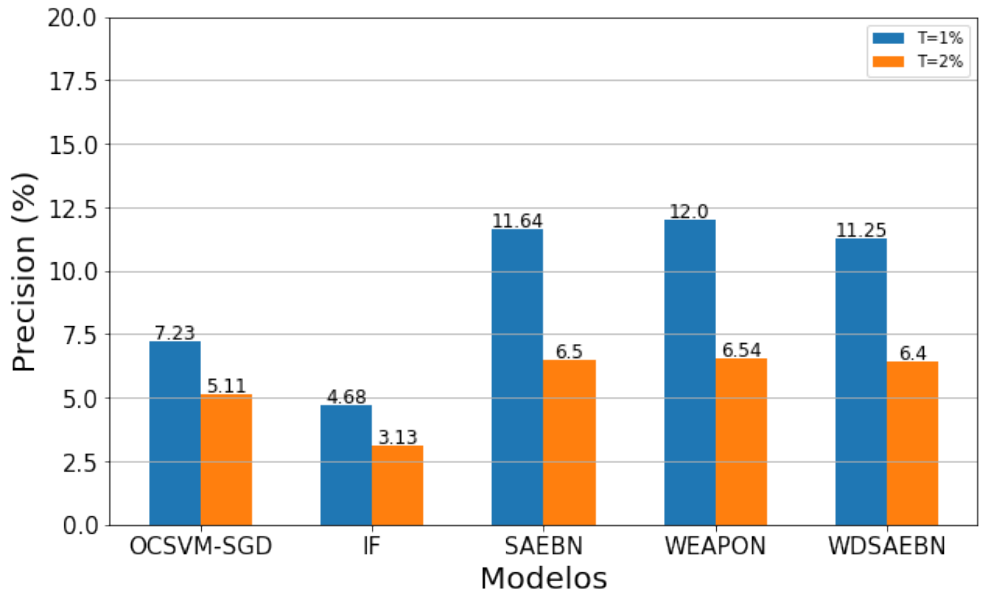


Figura 5.3: Resultados obtidos para a Métrica *Precision*.

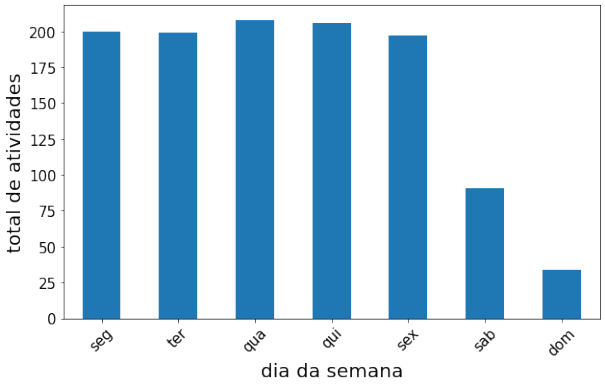
Os valores baixos alcançados por todos os modelos podem ser justificados em razão de o *dataset* de teste ser desbalanceado. O Snorkel rotulou 6.229 anomalias em um conjunto de 3.278.931 registros, o que representa apenas 0,19% dos dados, enquanto os valores de taxa de contaminação testados na avaliação dos modelos são T=1% e T=2%, ou seja, esses percentuais indicam a quantidade dos dados que serão considerados anomalias pelos modelos. Isso nos traz a uma importante reflexão: O Snorkel foi capaz de rotular todas as possíveis anomalias de comportamento existentes do conjunto de teste? Certamente não,

uma vez que as LFs utilizadas pelo Snorkel representam apenas a capacidade analítica de um analista de segurança, desconsiderando situações possivelmente impensáveis ou imprevisíveis. Para demonstrar isso, tomemos um evento de falso positivo detectado pelo WEAPON, que está representado na Tabela 5.1. A anomalia consiste em um registro indicando que o usuário realizou um *logoff* de sua estação às 3h da madrugada de um domingo.

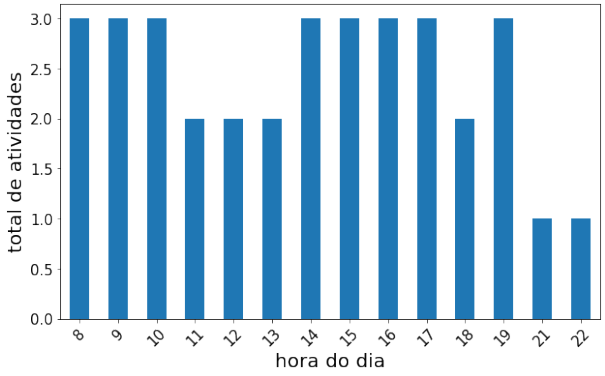
Tabela 5.1: Evento de falso positivo detectado pelo WEAPON no conjunto de teste com valor de limiar T=1%.

User	Date	logon	logoff	down	up	vis	conn	disc	trm	frm	open	write	copy	delete	hour	dow
GDH2500	2010-05-02 03:00:00	0	1	0	0	0	0	0	0	0	0	0	0	0	3	6

Para investigar se esse evento é de fato uma anomalia real, verificamos se esse evento está realmente em consonância com o padrão de comportamento desse usuário no conjunto de treinamento. Para facilitar a análise, a Figura 5.4 apresenta as principais características de comportamento do usuário para essa avaliação. Verificou-se que no conjunto de treinamento, seu padrão de comportamento indica um usuário que tem hábito de trabalhar aos finais de semana (Figura 5.4a). Avalindo seu perfil de trabalho aos domingos (Figura 5.4b), observamos que seu perfil de horário de trabalho varia entre 8h e 22h.



(a) Padrão de atividades do usuário GDH2500 em dias da semana no conjunto de treinamento.



(b) Perfil de horários de trabalho do usuário GDH2500 aos domingos no conjunto de treinamento.

Figura 5.4: Perfis de comportamento do usuário GDH2500 no conjunto de treinamento.

Assim, conclui-se que o evento de *logoff* às 3h da madrugada de domingo detectado pelo WEAPON é claramente uma anomalia de comportamento que o snorkel não encontrou em razão de não ter nenhuma LF programada para tal. Por outro lado, vale salientar que WEAPON detectou este comportamento anômalo porque seu mecanismo de detecção é baseado no erro de reconstrução obtido pelo seu autoencoder modificado, de modo que o processo de detecção de anomalias pode ser muito mais amplo e menos dependente de regras. Portanto, isso significa que WEAPON supera o modelo baseado na detecção LF do Snorkel quando estamos lidando com anomalias não analíticas, em outras palavras, anomalias que um especialista possivelmente não anteciparia.

Outra análise relevante para a avaliação do comportamento dos modelos é a curva ROC, conforme exibida na Figura 5.5. Ela mostra o desempenho da detecção de anomalias de comportamento para diferentes valores limiar em relação à sensibilidade e especificidade, e a linha tracejada representa a curva de resposta de um classificador puramente aleatório. Uma curva ótima é aquela mais próxima da extremidade superior esquerda do gráfico, ou seja, quanto mais alto o verdadeiro positivo e mais baixo o falso positivo. Dos

resultados obtidos, observa-se que o WEAPON é o modelo cuja curva está mais próxima da borda superior esquerda, indicando que para diferentes valores de limiar, ele tem a melhor resposta de detecção com a maior área sob a curva (AUC).

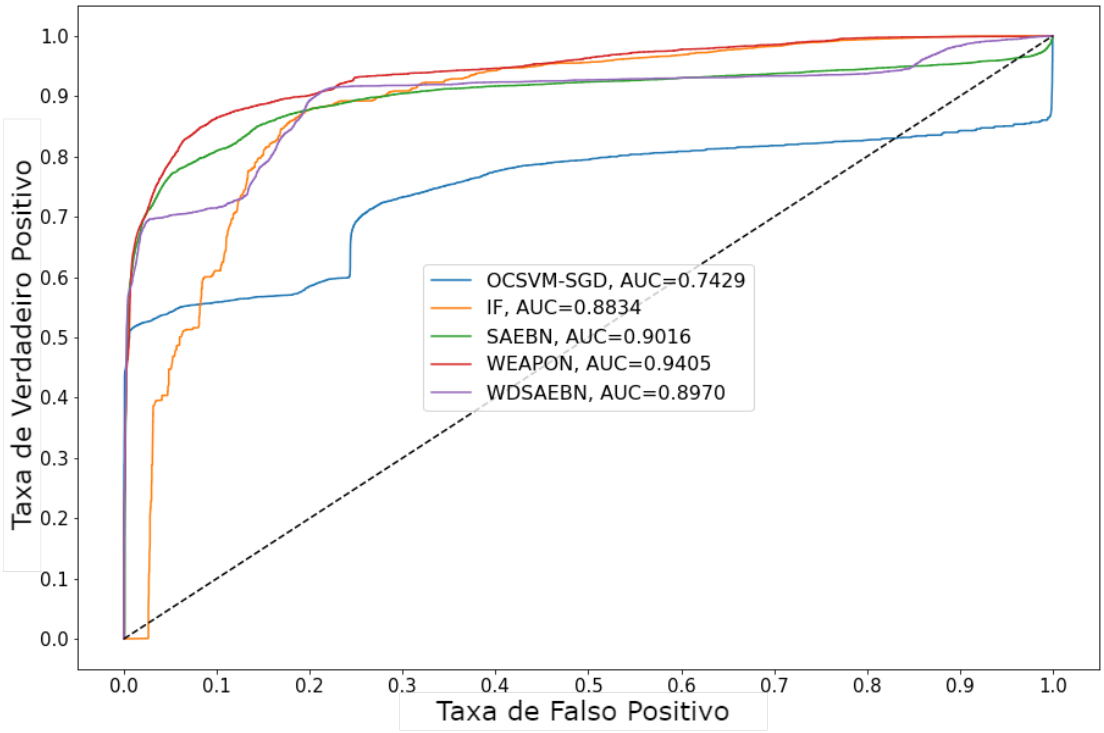


Figura 5.5: Curvas ROC para os modelos avaliados.

Isto ocorre porque WEAPON foi modelado utilizando uma camada convolucional unidimensional (convolução temporal) associada a redes neurais recorrentes do tipo LSTM, o que garantiu que o modelo aprendesse a interpretar melhor os padrões do usuário a partir do conjunto de treinamento. Com isso, o WEAPON conseguiu distinguir anomalias de comportamento e apresentar uma melhor resposta em diferentes valores de limiar. Finalmente, o número reduzido de camadas intermediárias fortalece a interpretação das características latentes do padrão de comportamento, o que é devido principalmente à sua estrutura de autoencoder. Além disso, o uso de entradas *wide and deep* utilizadas no WEAPON garante que a identificação do usuário e seu perfil comportamental associado ao horário de trabalho e dia da semana estejam ligados tanto ao padrão de comportamento aprendido (conjunto de treinamento) quanto ao avaliado na detecção (conjunto de testes).

A avaliação do desempenho computacional do WEAPON também é um requisito importante de ser investigado. Para isso, todos os modelos foram submetidos a verificação dos tempos de execução. A Figura 5.6 apresenta os tempos de execução das etapas de treinamento e teste. O treinamento corresponde à etapa de aprendizado dos padrões de comportamento dos usuários, e o teste, à detecção de anomalias de comportamento de usuários. A análise indica que os algoritmos de *deep learning* possuem um tempo de execução na etapa de treinamento superior aos algoritmos de *shallow learning*. Isso decorre da complexidade dos cálculos de ajuste de pesos do algoritmo *backpropagation*. No treinamento, o WEAPON alcançou um tempo de execução de aproximadamente 4 minutos e 51 segundos, sendo 6,27% mais lento que o segundo melhor modelo para detecção de anomalias, o SAEBN. Já na fase de detecção, o WEAPON

alcançou um tempo de execução de aproximadamente 8,6 segundos, sendo 52,35% mais lento que o SA-EBN. Os melhores resultados de tempo de execução tanto no treinamento como no teste foram obtidos pelo pior modelo, o OCSVM-SGD. Esse modelo implementa uma versão do algoritmo tradicional SVM, porém utilizando uma técnica de aproximação de Kernel por meio do Gradiente Descendente Estocástico [55], o que garante uma complexidade linear em função do número de amostras.

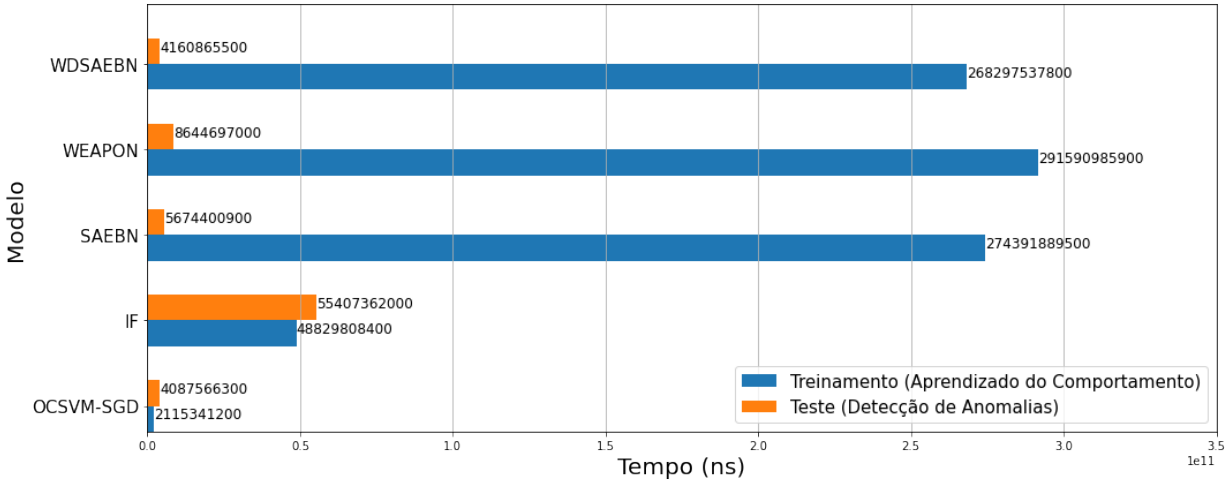


Figura 5.6: Tempos de execução das etapas de treinamento (aprendizado de comportamento) e teste (detecção de anomalias de comportamento).

5.3 DISCUSSÃO

A partir dos resultados obtidos, observamos que o WEAPON é superior aos modelos comparados quanto à capacidade de detecção de anomalias de comportamento de usuários, uma vez que alcançou os maiores valores de *Recall*, *ROC-AUC*, *Precision* e curva ROC. Vale destacar que no contexto deste estudo, as métricas *Recall* e *ROC-AUC* são as mais importantes para a escolha do melhor modelo. A análise de um caso falso positivo indicou que o WEAPON é capaz de encontrar anomalias de comportamento complexas, dificilmente previsíveis por um analista de segurança, o que lhe confere uma capacidade de detecção mais ampla e menos dependente de regras. Com relação ao tempo de execução, considerando que as etapas de treinamento e teste envolveram conjuntos de dados relativamente grandes, com registros associados a atividades de 4.000 usuários durante quatro meses em cada conjunto, consideramos que os tempos de execução alcançados pelo WEAPON são satisfatórios, uma vez que esse modelo possui os melhores resultados para a detecção de anomalias.

6 CONCLUSÃO

O cenário mundial de crescentes ataques cibernéticos indica a clara necessidade de adoção de mecanismos mais amplos de proteção. A detecção de anomalias no comportamento dos usuários tem um papel crucial como medida de alerta contra ataques cibernéticos. Nesse sentido, os avanços computacionais têm permitido que modelos baseados em AM e *big data* sejam aplicados com sucesso para detectar estas anomalias. Entretanto, é fundamental que os modelos tenham aplicação em cenários do mundo real, em que as anomalias de comportamento são consideradas eventos de natureza não supervisionada. Além disso, o perfil de comportamento do usuário precisa ser estabelecido em um horizonte de tempo curto e suficiente para que a detecção de anomalias possa ser realizada o mais rápido possível.

Nesse contexto, este trabalho alcançou seus objetivos propondo o WEAPON, uma arquitetura capaz de detectar anomalias de comportamento de usuários. Para isso, foi utilizada uma estrutura de autoencoder modificada que utiliza a arquitetura *wide and deep*, camada convolucional unidimensional (convolução temporal) e camadas LSTM. Essa configuração de autoencoder permitiu que o WEAPON pudesse aprender as representações latentes das caraterísticas comportamentais dos usuários, e, além disso, distinguir os perfis de comportamento individuais. Para tanto, o WEAPON requer uma quantidade de dados reduzida, pois requer quatro meses de dados para formar os padrões comportamentais dos usuários e então estar habilitado para a detecção de anomalias de comportamento.

A partir do CERT *Dataset* foi possível validar a eficiência do WEAPON em comparação com outros modelos derivados da literatura. O desempenho obtido alcançou um *Recall* de 68,87% e ROC-AUC de 0.9405, demonstrando ser mais eficiente que outros modelos baseados em *shallow learning* e *deep learning*. Além disso, os resultados demostraram que o WEAPON possibilita uma detecção de anomalias de comportamento abrangente, pois identifica anomalias não analíticas que um especialista no domínio possivelmente não conseguiria antecipar para configurar um modelo baseado em regras.

Além disso, o trabalho contribuiu ao estender o uso do CERT *Dataset* para além do seu propósito original de detecção de *insiders*, permitindo que sejam obtidas novas fontes de anomalias de comportamento de usuários. A estratégia para detecção de anomalias utilizada nesta pesquisa implementou um processo de rotulação de anomalias de comportamento de usuários baseado em supervisão fraca usando o Snorkel. Isso permitiu avaliar o desempenho do WEAPON de acordo com rótulos obtidos a partir de um conjunto de funções de rotulação que, combinadas, equivalem à opinião de um especialista no domínio de conhecimento. Essa estratégia de rotulação permite que futuros estudos se beneficiem de uma base comum de metodologia de rotulação, assegurando que possam ser comparados entre si. Até o limite dessa pesquisa, trata-se de uma proposta nova, pois não se encontrou estudo semelhante voltado para rotulação de anomalias de comportamento.

6.1 TRABALHOS FUTUROS

Como pesquisas futuras, destacamos que esse estudo aponta para um grande leque de oportunidades de evolução, dada a grande quantidade de graus de liberdade existente. Inicialmente, a pesquisa utilizou a validação da detecção de anomalias de comportamento no formato de lote (*batch*), validando quatro meses de dados do conjunto de teste. Em um cenário real, abordar a detecção de forma *online* significa habilitar o WEAPON para a detecção de anomalias de comportamento de usuários a cada hora, tão logo os *logs* de eventos de segurança estejam disponíveis para o pré-processamento. Assim, consideramos promissor estudar a adaptação da etapa de detecção de anomalias com base no erro de reconstrução para possilitar essa alternativa.

Outro estudo promissor está relacionado a utilizar as detecções falso positivas do WEAPON para identificar possíveis novas funções de rotulação a serem consideradas na supervisão fraca do Snorkel, e assim, procurar reavaliar a eficiência de WEAPON em relação a outros modelos.

Por fim, um novo estudo pode avaliar a possibilidade de que determinadas variáveis comportamentais possam ter mais importância no processo de detecção de anomalias, como por exemplo, atribuir que alterações significativas de *downloads* possam ter mais importância que alterações significativas de visitas *web*. Isso pode ser avaliado a partir da aplicação de pesos para as *features* comportamentais no procedimento de treinamento.

REFERÊNCIAS BIBLIOGRÁFICAS

1 CHOLLET, F. *Deep Learning with Python*. [S.l.]: Manning, 2017. ISBN 9781617294433.

2 GERON, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd. ed. [S.l.]: O'Reilly Media, Inc., 2019. ISBN 1492032646.

3 AGGARWAL, C. C. *Outlier Analysis*. 2nd. ed. [S.l.]: Springer Publishing Company, Incorporated, 2017. ISBN 3319475770.

4 ALPAYDIN, E. *Introduction to Machine Learning*. [S.l.]: The MIT Press, 2014. ISBN 0262028182.

5 CINCO controles de segurança cibernética para ontem. <https://portal.tcu.gov.br/5-controles-de-seguranca-cibernetica.htm>. Accessado em 24/08/2022.

6 PREPARE for the Future of Remote Work. <https://www.gartner.com/en/human-resources/trends/remote-work-revolution>. Accessado em 23/08/2022.

7 REMOTE Work After COVID-19 | Gartner for HR. <https://www.gartner.com/en/human-resources/trends/remote-work-after-covid>. Accessado em 23/08/2022.

8 GARTNER Survey Reveals a 44% Rise in Workers' Use of Collaboration Tools Since 2019. <https://www.gartner.com/en/newsroom/press-releases/2021-08-23-gartner-survey-reveals-44-percent-rise-in-workers-use-of-collaboration-tools-since-2019>. Accessado em 23/08/2022.

9 COLLABORATION tool security: How to avoid common risks. <https://www.techtarget.com/searchunifiedcommunications/tip/Collaboration-tool-security-How-to-avoid-common-risks>. Accessado em 23/08/2022.

10 HACKERS que invadiram ConecteSUS usam truque antigo para driblar autenticação. <https://tecnoblog.net/noticias/2022/03/29/hackers-que-invadiram-conectesus-usam-truque-antigo-para-driblar-autenticacao>. Accessado em 23/08/2022.

11 LEVERAGING The Cloud During The Pandemic. <https://www.swktech.com/7-reasons-why-mfa-is-critical-to-cloud-hosting>. Accessado em 23/08/2022.

12 CYBER attacks more than double since COVID-19, PwC Thailand says. <https://www.pwc.com/th/en/press-room/press-release/2021/press-release-18-08-21-en.html>. Accessado em 23/08/2022.

13 LEVANTAMENTO mostra que ataques cibernéticos no Brasil cresceram 94%. <https://www.cnnbrasil.com.br/tecnologia/levantamento-mostra-que-ataques-ciberneticos-no-brasil-cresceram-94>. Accessado em 23/08/2022.

14 HARRIS, S.; MAYMI, F. *CISSP All-in-One Exam Guide, Ninth Edition*. McGraw-Hill Education, 2021. ISBN 9781260467376. Disponível em: <https://books.google.com.br/books?id=ZVFdzgEACAAJ>.

15 UNDERSTANDING Targeted Attacks: Goals and Motives. <https://www.trendmicro.com/vinfo/es/security/news/cyber-attacks/understanding-targeted-attacks-goals-and-motives>. Accessado em 23/08/2022.

16 MEHROTRA, K. G.; MOHAN, C. K.; HUANG, H. *Anomaly Detection Principles and Algorithms*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2017. ISBN 3319675249.

17 POKHREL, R.; POKHAREL, P.; Kumar Timalsina, A. Anomaly-Based – Intrusion Detection System using User Profile Generated from System Logs. *International Journal of Scientific and Research Publications (IJSRP)*, v. 9, n. 2, p. p8631, 2019.

18 JIN, Q.; WANG, L. Intranet User-Level Security Traffic Management with Deep Reinforcement Learning. *Proceedings of the International Joint Conference on Neural Networks*, IEEE, v. 2019-July, n. July, p. 1–8, 2019.

19 PANG, G.; SHEN, C.; CAO, L.; HENGEL, A. V. D. Deep learning for anomaly detection: A review. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 54, n. 2, mar 2021. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3439950>.

20 THUDUMU, S.; BRANCH, P.; JIN, J.; SINGH, J. J. A comprehensive survey of anomaly detection techniques for high dimensional big data. *Journal of Big Data*, Springer, v. 7, n. 1, p. 1–30, 2020.

21 PRARTHANA, T. S.; GANGADHAR, N. D. User behaviour anomaly detection in multidimensional data. In: *2017 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*. [S.l.: s.n.], 2017. p. 3–10.

22 Abu Sulayman, I. I.; OUDA, A. User Modeling via Anomaly Detection Techniques for User Authentication. *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference, IEMCON 2019*, IEEE, p. 169–176, 2019.

23 VILAÇA, E. S. C.; VIEIRA, T. P. B.; SOUSA, R. T. de; COSTA, J. P. C. L. da. Botnet traffic detection using RPCA and mahalanobis distance. In: *2019 Workshop on Communication Networks and Power Systems (WCNPS)*. [S.l.: s.n.], 2019. p. 1–6.

24 KIM, J.; PARK, M.; KIM, H.; CHO, S.; KANG, P. Insider threat detection based on user behavior modeling and anomaly detection algorithms. *Applied Sciences (Switzerland)*, v. 9, n. 19, 2019. ISSN 20763417.

25 JIN, Y.; QIU, C.; SUN, L.; PENG, X.; ZHOU, J. Anomaly detection in time series via robust PCA. *2017 2nd IEEE International Conference on Intelligent Transportation Engineering, ICITE 2017*, p. 352–355, 2017.

26 JUNIOR, G.; CARVALHO, L. C.; RODRIGUES, J. R.; PROENçA, M. P. Network anomaly detection using IP flows with principal component analysis and ant colony optimization. *Journal of Network and Computer Applications*, Elsevier, v. 64, n. 1, p. 1–11, April 2016. ISSN 1095-8592.

27 GUPTA, M.; GAO, J.; AGGARWAL, C. C.; HAN, J. Outlier Detection for Temporal Data: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, v. 26, n. 9, p. 2250–2267, 2014. ISSN 10414347.

28 THOMé, M.; PRESTES, A.; GOMES, R.; MOTA, V. Um arcabouço para detecção e alerta de anomalias de mobilidade urbana em tempo real. In: *Anais do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Porto Alegre, RS, Brasil: SBC, 2020. p. 784–797. ISSN 2177-9384. Disponível em: <https://sol.sbc.org.br/index.php/sbrc/article/view/12325>.

29 SHIN, K.; HOOI, B.; KIM, J.; FALOUTSOS, C. Detecting group anomalies in tera-scale multi-aspect data via dense-subtensor mining. *Frontiers in Big Data*, v. 3, 2021. ISSN 2624-909X. Disponível em: <https://www.frontiersin.org/article/10.3389/fdata.2020.594302>.

30 SADIK, M. S.; GRUENWALD, L. Research issues in outlier detection for data streams. *ACM SIGKDD Explorations Newsletter*, v. 15, p. 33–40, 03 2014.

31 GAO, Y.; MA, Y.; LI, D. Anomaly detection of malicious users' behaviors for web applications based on web logs. In: *2017 IEEE 17th International Conference on Communication Technology (ICCT)*. [S.l.: s.n.], 2017. p. 1352–1355.

32 KWON, D.; KIM, H.; KIM, J.; SUH, S. C.; KIM, I.; KIM, K. J. A survey of deep learning-based network anomaly detection. *Cluster Computing*, v. 22, p. 949–961, 2017.

33 AHMED, M.; MAHMOOD, A. N.; HU, J. A survey of network anomaly detection techniques. *J. Netw. Comput. Appl.*, Academic Press Ltd., GBR, v. 60, n. C, p. 19–31, jan 2016. ISSN 1084-8045. Disponível em: <https://doi.org/10.1016/j.jnca.2015.11.016>.

34 AGRAWAL, S.; AGRAWAL, J. Survey on anomaly detection using data mining techniques. *Procedia Computer Science*, Elsevier Masson SAS, v. 60, n. 1, p. 708–713, 2015. ISSN 18770509. Disponível em: <http://dx.doi.org/10.1016/j.procs.2015.08.220>.

35 PRASAD, N. R.; ALMANZA-GARCIA, S.; LU, T. T. Anomaly detection. *Computers, Materials and Continua*, v. 14, n. 1, p. 1–22, 2009. ISSN 15462218.

36 PATCHA, A.; PARK, J. M. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, v. 51, n. 12, p. 3448–3470, 2007. ISSN 13891286.

37 ZHANG, C.; WANG, S.; ZHAN, D.; YU, T.; WANG, T.; YIN, M. Detecting Insider Threat from Behavioral Logs Based on Ensemble and Self-Supervised Learning. *Security and Communication Networks*, v. 2021, 2021. ISSN 19390122.

38 QU, Z.; SU, L.; WANG, X.; ZHENG, S.; SONG, X.; SONG, X. A Unsupervised Learning Method of Anomaly Detection Using GRU. *Proceedings - 2018 IEEE International Conference on Big Data and Smart Computing, BigComp 2018*, IEEE, p. 685–688, 2018.

39 AL-MHIQANI, M. N.; AHMAD, R.; ABIDIN, Z. Z.; ABDULKAREEM, K. H.; MOHAMMED, M. A.; GUPTA, D.; SHANKAR, K. A new intelligent multilayer framework for insider threat detection. *Computers Electrical Engineering*, v. 97, p. 107597, 2022. ISSN 0045-7906. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045790621005322>.

40 ATAQUES de ransomware aumentaram 90% entre 2020 e 2021, alerta SonicWall Capture Labs. <https://tiinside.com.br/21/05/2021/ataques-de-ransomware-aumentaram-90-entre-2020-e-2021-alerta-sonicwall-capture-labs>. Accessado em 23/08/2022.

41 ATAQUES cibernéticos crescem 300%, segundo relatório da NTT. <https://tiinside.com.br/29/05/2021/ataques-ciberneticos-crescem-300-segundo-relatorio-da-ntt>. Accessado em 23/08/2022.

42 RATNER, A.; BACH, S.; EHRENBERG, H.; FRIES, J.; WU, S.; Ré, C. Snorkel: rapid training data creation with weak supervision. *The VLDB Journal*, v. 29, 05 2020.

43 MOLINA, A. L.; GONÇALVES, V. P.; JR, R. T. de S.; GIUNTINI, F. T.; PESSIN, G.; MENEGUETTE, R. I.; FILHO, G. P. R. Weapon: Uma arquitetura para detecção de anomalias de comportamento do usuário. In: SBC. *Anais do XI Brazilian Workshop on Social Network Analysis and Mining*. [S.l.], 2022. p. 121–132.

44 LORENA, A.; FACELI, K.; ALMEIDA, T.; CARVALHO, A. de; GAMA, J. *Inteligência Artificial: uma abordagem de Aprendizado de Máquina (2a edição)*. [S.l.: s.n.], 2021. ISBN 9788521637493.

45 HAYKIN, S. S. *Neural networks and learning machines*. Third. Upper Saddle River, NJ: Pearson Education, 2009.

46 ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, v. 65, n. 6, p. 386–408, 1958. ISSN 0033-295X. Disponível em: <http://dx.doi.org/10.1037/h0042519>.

47 RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning internal representations by error propagation. In: ____. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986. p. 318–362. ISBN 026268053X.

48 CHENG, H.-T.; KOC, L.; HARMSEN, J.; SHAKED, T.; CHANDRA, T.; ARADHYE, H.; ANDERSON, G.; CORRADO, G.; CHAI, W.; ISPIR, M.; ANIL, R.; HAQUE, Z.; HONG, L.; JAIN, V.; LIU, X.; SHAH, H. Wide deep learning for recommender systems. In: *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. New York, NY, USA: Association for Computing Machinery, 2016. (DLRS 2016), p. 7–10. ISBN 9781450347952. Disponível em: <https://doi.org/10.1145/2988450.2988454>.

49 HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural computation*, v. 9, p. 1735–80, 12 1997.

50 LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE*. [s.n.], 1998. v. 86, n. 11, p. 2278–2324. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665>.

51 LIU, F. T.; TING, K. M.; ZHOU, Z.-H. Isolation forest. In: *2008 Eighth IEEE International Conference on Data Mining*. [S.l.: s.n.], 2008. p. 413–422.

52 REGAYA, Y.; FADLI, F.; AMIRA, A. Point-denoise: Unsupervised outlier detection for 3d point clouds enhancement. *Multimedia Tools and Applications*, v. 80, p. 1–17, 07 2021.

53 PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011.

54 CRISTIANINI, N.; RICCI, E. Support vector machines. In: ____. *Encyclopedia of Algorithms*. Boston, MA: Springer US, 2008. p. 928–932. ISBN 978-0-387-30162-4. Disponível em: <https://doi.org/10.1007/978-0-387-30162-4_415>.

55 ZHANG, T. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In: *Proceedings of the Twenty-First International Conference on Machine Learning*. New York, NY, USA: Association for Computing Machinery, 2004. (ICML '04), p. 116. ISBN 1581138385. Disponível em: <https://doi.org/10.1145/1015330.1015332>.

56 BOTTOU, L. Large-scale machine learning with stochastic gradient descent. In: LECHEVALLIER, Y.; SAPORTA, G. (Ed.). *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*. Paris, France: Springer, 2010. p. 177–187. Disponível em: <http://leon.bottou.org/papers/bottou-2010>.

57 COTTRELL, G.; MUNRO, P.; ZIPSER, D. Learning internal representations from gray-scale images: An example of extensional programming. 01 1987.

58 GLASSER, J.; LINDAUER, B. Bridging the gap: A pragmatic approach to generating insider threat data. In: *2013 IEEE Security and Privacy Workshops*. [S.l.: s.n.], 2013. p. 98–104.

59 LINDAUER, B. *Insider Threat Test Dataset*. [S.l.]: Carnegie Mellon University, 2020. <https://kilthub.cmu.edu/articles/dataset/Insider_Threat_Test_Dataset/12841247/1>.

60 RATNER, A.; HANCOCK, B.; DUNNMON, J.; SALA, F.; PANDEY, S.; Ré, C. *Training Complex Models with Multi-Task Weak Supervision*. arXiv, 2018. Disponível em: <https://arxiv.org/abs/1810.02840>.

61 MASCI, J.; MEIER, U.; CIREŞAN, D.; SCHMIDHUBER, J. Stacked convolutional auto-encoders for hierarchical feature extraction. In: HONKELA, T.; DUCH, W.; GIROLAMI, M.; KASKI, S. (Ed.). *Artificial Neural Networks and Machine Learning – ICANN 2011*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 52–59. ISBN 978-3-642-21735-7.

62 SRIVASTAVA, N.; MANSIMOV, E.; SALAKHUTDINOV, R. Unsupervised learning of video representations using lstms. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. [S.l.]: JMLR.org, 2015. (ICML'15), p. 843–852.

63 DOZAT, T. Incorporating Nesterov Momentum into Adam. In: *Proceedings of the 4th International Conference on Learning Representations*. [S.l.: s.n.], 2016. p. 1–4.

64 MCKINNEY Wes. Data Structures for Statistical Computing in Python. In: WALT Stéfan van der; MILLMAN Jarrod (Ed.). *Proceedings of the 9th Python in Science Conference*. [S.l.: s.n.], 2010. p. 56 – 61.

65 HARRIS, C. R.; MILLMAN, K. J.; WALT, S. J. van der; GOMMERS, R.; VIRTANEN, P.; COURNAPEAU, D.; WIESER, E.; TAYLOR, J.; BERG, S.; SMITH, N. J.; KERN, R.; PICUS, M.; HOYER, S.; KERKWIJK, M. H. van; BRETT, M.; HALDANE, A.; RÍO, J. F. del; WIEBE, M.; PETERSON, P.; GÉRARD-MARCHANT, P.; SHEPPARD, K.; REDDY, T.; WECKESSER, W.; ABBASI, H.; GOHLKE, C.; OLIPHANT, T. E. Array programming with NumPy. *Nature*, Springer Science and Business Media LLC, v. 585, n. 7825, p. 357–362, set. 2020. Disponível em: <https://doi.org/10.1038/s41586-020-2649-2>.

66 ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; GOODFELLOW, I.; HARP, A.; IRVING, G.; ISARD, M.; JIA, Y.; JOZEFOWICZ, R.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANé, D.; MONGA, R.; MOORE, S.; MURRAY, D.; OLAH, C.; SCHUSTER, M.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P.; VANHOUCKE, V.; VASUDEVAN, V.; VIéGAS, F.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y.; ZHENG, X. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: <https://www.tensorflow.org/>.

67 PASZKE, A.; GROSS, S.; MASSA, F.; LERER, A.; BRADBURY, J.; CHANAN, G.; KILLEEN, T.; LIN, Z.; GIMELSHEIN, N.; ANTIGA, L.; DESMAISON, A.; KOPF, A.; YANG, E.; DEVITO, Z.; RAISON, M.; TEJANI, A.; CHILAMKURTHY, S.; STEINER, B.; FANG, L.; BAI, J.; CHINTALA, S. Pytorch: An imperative style, high-performance deep learning library. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019. p. 8024–8035. Disponível em: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.

68 CHOLLET, F. et al. *Keras*. 2015. <https://keras.io>.

69 BERGSTRA, J.; KOMER, B.; ELIASMITH, C.; YAMINS, D.; COX, D. D. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science Discovery*, v. 8, n. 1, p. 014008, 2015. Disponível em: <http://stacks.iop.org/1749-4699/8/i=1/a=014008>.

70 BERGSTRA, J.; KOMER, B.; ELIASMITH, C.; YAMINS, D.; COX, D. D. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, IOP Publishing, v. 8, n. 1, p. 014008, 2015.

71 ZHAO, Y.; NASRULLAH, Z.; LI, Z. Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 01 2019.

APÊNDICES

I. CÓDIGO FONTE DO PRÉ-PROCESSAMENTO

```
# Importa a biblioteca pandas
import pandas as pd

# Importa datetime e timedelta para verificar se ha gaps de tempo nos datasets preparados
from datetime import datetime, timedelta

# Importa a biblioteca os
import os

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures, OneHotEncoder, OrdinalEncoder

#Para as figuras
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

#Importa a biblioteca Numpy
import numpy as np

# Carregamento do dataset original, so na primeira execucao!
#Carrega o dataset fazendo atribuicao de tipos para diminuir consumo de memoria
df_http = pd.read_csv("D:/OneDrive/ML/Datasets/Cert/data/r6.2/http.csv", usecols=['date', 'user', 'pc', 'url', 'activity'], infer_datetime_format=True,
                parse_dates=['date'], dtype={"user":"category", "pc":"category","url":"category","activity":"category"})



# So na primeira execucao!
```

```
# Adiciona coluna de valores 0 ou 1 conforme coluna 'activity', para
# depois consolidar as somas das atividades nessas colunas
df_http['down'] = np.where((df_http['activity']== 'WWW Download')
    ↪ ,1,0)
df_http['up'] = np.where((df_http['activity']== 'WWW Upload'),1,0)
df_http['vis'] = np.where((df_http['activity']== 'WWW Visit'),1,0)

# Salva o Dataset em arquivo.
# Nas proximas rodadas, basta carregar o arquivo "df_http_file.pkl".
df_http.to_pickle("df_http_file.pkl")

# Rodar somente na primeira execucao
#Agrupamento para consolidar dados a cada 1 hora, criando baseline
    ↪ temporal por usuario
df_h_1hora = df_http.groupby(['user',pd.Grouper(key='date',freq='1H'
    ↪ )])['down','up','vis'].sum().reset_index().sort_values(['user'
    ↪ ,'date'], ascending=[True, True])


# Rodar somente na primeira execucao
#Criando a coluna para a hora dos eventos
df_h_1hora['hour'] = df_h_1hora['date'].dt.hour

#Criando a coluna dow para dia da semana
df_h_1hora['dow'] = df_h_1hora['date'].dt.dayofweek

## Limpeza das linhas com valores de zero para down up vis
df_h_1hora.drop(df_h_1hora[(df_h_1hora['down'] == 0) & (df_h_1hora['
    ↪ up'] == 0) & (df_h_1hora['vis'] == 0)].index, axis=0, inplace=
    ↪ True)

### Grava em arquivo a versao sem zeros
df_h_1hora.to_pickle("df_h_1hora_file.pkl")
### Carrega do arquivo
df_h_1hora = pd.read_pickle("df_h_1hora_file.pkl")

# Carregamento do dataset original, so na primeira execucao!
#Carrega o dataset fazendo atribuicao de tipos para diminuir consumo
    ↪  de memoria
df_usb = pd.read_csv("D:/OneDrive/ML/Datasets/Cert/data/r6.2/
    ↪ file.csv",
```

```
64                    usecols=['date','user','pc','activity','
                         ↪ to_removable_media','from_removable_media'],
65                    infer_datetime_format=True,parse_dates=['date'],
66                    dtype={"user":"category", "pc":"category","activity
                         ↪ ":"category",
67                       "to_removable_media":"int32","
                            ↪ from_removable_media":"int32"})
68
69 df_usb.rename(columns={'to_removable_media':'trm',
70                     'from_removable_media':'frm'},
71              inplace = True)
72
73 # So na primeira execucao!
74 # Adiciona coluna de valores 0 ou 1 conforme coluna 'activity'
75 df_usb['open'] = np.where((df_usb['activity']== 'File Open'),1,0)
76 df_usb['write'] = np.where((df_usb['activity']== 'File Write'),1,0)
77 df_usb['copy'] = np.where((df_usb['activity']== 'File Copy'),1,0)
78 df_usb['delete'] = np.where((df_usb['activity']== 'File Delete')
      ↪ ,1,0)
79
80 #Retira as colunas 'activity' e 'pc'
81 df_usb.drop(['activity','pc'], axis='columns', inplace=True)
82
83 ## Salva em arquivo para uso posterior
84 df_usb.to_pickle("df_usb_file.pkl")
85
86 ## Carrega arquivo
87 df_usb = pd.read_pickle("df_usb_file.pkl")
88
89 # Rodar somente na primeira execucao
90 #Agrupamento para consolidar dados a cada 1 hora, criando baseline
      ↪ temporal por usuario
91 #df_usb_1hora = df_usb.groupby(['user',pd.Grouper(key='date',freq='1
      ↪ H')])[\
92 # 'open','write','copy','delete','to_removable_media',\
93 # 'from_removable_media'].sum().reset_index().sort_values(['user','
      ↪ date'], ascending=[True, True])
94
95 # Rodar somente na primeira execucao
96 #Criando a coluna para a hora dos eventos
97 #df_usb_1hora['hour'] = df_usb_1hora['date'].dt.hour
98
```

```
#Criando a coluna dow para dia da semana
#df_usb_1hora['dow'] = df_usb_1hora['date'].dt.dayofweek


# Salva os Datasets em arquivos.
# Nas proximas rodadas, basta carregar o arquivo "df_usb_1h_file.pkl "
#df_usb_1hora.to_pickle("df_usb_1h_file.pkl")

#Carrega o dataset fazendo atribuicao de tipos para diminuir consumo de memoria
df_device = pd.read_csv("D:/OneDrive/ML/Datasets/Cert/data/r6.2/device.csv",
        usecols=['date','user','pc','activity'],
        infer_datetime_format=True,parse_dates=['date'],
        dtype={"user":"category", "pc":"category","activity":"category"})

# So na primeira execucao!
# Adiciona coluna de valores 0 ou 1 conforme coluna 'activity'
df_device['conn'] = np.where((df_device['activity']== 'Connect'),1,0)
df_device['disc'] = np.where((df_device['activity']== 'Disconnect'),1,0)

#Retira as colunas 'activity' e 'pc'
df_device.drop(['activity','pc'], axis='columns', inplace=True)

## Salva em arquivo para uso posterior
df_device.to_pickle("df_device_file.pkl")

## Carrega arquivo
df_device = pd.read_pickle("df_device_file.pkl")

# Carregamento do dataset original, so na primeira execucao!
#Carrega o dataset fazendo atribuicao de tipos para diminuir consumo de memoria
df_logon = pd.read_csv("D:/OneDrive/ML/Datasets/Cert/data/r6.2/logon.csv",
        usecols=['date','user','pc','activity'],
        infer_datetime_format=True,parse_dates=['date'],
```

```
132         dtype={"user":"category", "pc":"category","activity":"
                ↪ category"})
133
134 # So na primeira execucao!
135 # Adiciona coluna de valores 0 ou 1 conforme coluna 'activity'
136 df_logon['logon'] = np.where((df_logon['activity']== 'Logon'),1,0)
137 df_logon['logoff'] = np.where((df_logon['activity']== 'Logoff'),1,0)
138
139 #Retira as colunas 'activity' e 'pc'
140 df_logon.drop(['activity','pc'], axis='columns', inplace=True)
141
142 ## Salva em arquivo para uso posterior
143 df_logon.to_pickle("df_logon_file.pkl")
144
145 ## Carrega arquivo
146 df_logon = pd.read_pickle("df_logon_file.pkl")
147
148 #Agregacao dos datasets
149 df_logon = df_logon = pd.read_pickle("df_logon_file.pkl")
150 df_http = pd.read_pickle("df_http_file.pkl")
151 df_device = pd.read_pickle("df_device_file.pkl")
152 df_usb = pd.read_pickle("df_usb_file.pkl")
153
154 df_http.drop(['pc','url','activity'], axis='columns', inplace=True)
155
156 df_lhud = pd.concat([df_logon, df_http],ignore_index=True)
157 df_lhud = pd.concat([df_lhud, df_device],ignore_index=True)
158 df_lhud = pd.concat([df_lhud, df_usb],ignore_index=True)
159 df_lhud.fillna(0, inplace=True)
160 df_lhud.sort_values(['user','date'], ascending=[True, True], inplace
      ↪ =True)
161 df_lhud["logon"] = df_lhud['logon'].astype('int32')
162 df_lhud["logoff"] = df_lhud['logoff'].astype('int32')
163 df_lhud["down"] = df_lhud['down'].astype('int32')
164 df_lhud["up"] = df_lhud['up'].astype('int32')
165 df_lhud["vis"] = df_lhud['vis'].astype('int32')
166 df_lhud["conn"] = df_lhud['conn'].astype('int32')
167 df_lhud["disc"] = df_lhud['disc'].astype('int32')
168 df_lhud["trm"] = df_lhud['trm'].astype('int32')
169 df_lhud["frm"] = df_lhud['frm'].astype('int32')
170 df_lhud["open"] = df_lhud['open'].astype('int32')
171 df_lhud["write"] = df_lhud['write'].astype('int32')
```

```
172 df_lhud["copy"] = df_lhud['copy'].astype('int32')
173 df_lhud["delete"] = df_lhud['delete'].astype('int32')
174
175 ### Grava em arquivo
176 df_lhud.to_pickle("df_lhud_file.pkl")
177
178 ### Carrega do arquivo
179 df_lhud = pd.read_pickle("df_lhud_file.pkl")
180
181 # Gera dataset df_lhud_1hora que e a consolidacao das atividades a
      ↪ cada hora.
182
183 # Rodar somente na primeira execucao
184 #Agrupamento para consolidar dados a cada 1 hora, criando baseline
      ↪ temporal por usuario
185 df_lhud_1hora = df_lhud.groupby(
186    ['user',pd.Grouper(key='date',freq='1H')]
187 )['logon','logoff','down','up','vis','conn','disc','trm','frm','open
      ↪ ','write','copy','delete'
188  ].sum().reset_index().sort_values(['user','date'], ascending=[True,
       ↪  True])
189
190 #Criando a coluna para a hora dos eventos
191 df_lhud_1hora['hour'] = df_lhud_1hora['date'].dt.hour
192
193 #Criando a coluna dow para dia da semana
194 df_lhud_1hora['dow'] = df_lhud_1hora['date'].dt.dayofweek
195
196 ### Grava em arquivo
197 df_lhud_1hora.to_pickle("df_lhud_1hora_file.pkl")
198
199 ### Carrega do arquivo
200 df_lhud_1hora = pd.read_pickle("df_lhud_1hora_file.pkl")
```

II. REGRAS DE FORMAÇÃO DE COMPORTAMENTO E FUNÇÕES DE ROTULAÇÃO - LFS UTILIZADAS PARA ROTULAÇÃO DO CONJUNTO DE TESTE PELO SNORKEL

```
# Importa a biblioteca pandas
import pandas as pd

# Importa a biblioteca os
import os

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
    ↪ , OneHotEncoder, OrdinalEncoder

#Para as figuras
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from matplotlib.cbook import boxplot_stats

#Importa a biblioteca Numpy
import numpy as np

#importa bibliotecas snorkel
import snorkel
from snorkel.labeling import labeling_function
from snorkel.labeling import PandasLFApplier
from snorkel.labeling.apply.dask import DaskLFApplier
from snorkel.labeling.apply.dask import PandasParallelLFApplier

## Carrega os dados processados
# Logon + HTTP + USB + Device
#df_lhud_1hora = pd.read_pickle("df_lhud_1hora_file.pkl")
df_lhud_1hora = pd.read_hdf("df_lhud_1hora_file.hdf",'df')
```

```
df_lhud_1hora.sort_values('date', ascending=True, inplace = True)
df_lhud_1hora.reset_index(inplace = True, drop=True)

## Separa 4 primeiros meses de dados
df_lhud_1hora4m = df_lhud_1hora[(df_lhud_1hora['date'] <= '
    ↪ 2010-05-02')]

train_index = df_lhud_1hora4m.index[-1]
test_index = df_lhud_1hora[(df_lhud_1hora['date'] <= '2010-09-02')]
    ↪ .index[-1]


## Conjunto de teste
df_lhud_1hora_test = df_lhud_1hora.iloc[train_index:test_index]


# For clarity, we define constants to represent the class labels for
    ↪  normal, anomaly, and abstaining.
ABSTAIN = -1
NORMAL = 0
ANOMALY = 1

## Label Functions General -- Funcoes de rotulacao (LFs)
## In order to label the dataset, weak supervision is applied here.
    ↪ Every feature
## is analysed, for every user. The behavior pattern is obtained
    ↪ from boxplot_stats
## applied in the training data (first 4 months). This behavior is
    ↪ saved in a
## DataFrame so it can be used after, during the weak supervision,
    ↪ for
## checking if the current value of the feature analysed is an
    ↪ anomaly or not.


## logon and logoff - Anomalies only occur when user behavior
    ↪ indicates more
## rates in one hour compared to fliers in training data
@labeling_function()
def logon_high(x):
    user = x['user']
    max_logon = users_stats.loc[user]['logon_whishi']
```

```
    # Return a label of ANOMALY if logon rate is too high in 1 hour,
    # otherwise ABSTAIN
    return ANOMALY if x['logon'] > max_logon else ABSTAIN

@labeling_function()
def logoff_high(x):
    user = x['user']
    max_logoff = users_stats.loc[user]['logoff_whishi']
    # Return a label of ANOMALY if logoff rate is too high in 1 hour,
    # otherwise ABSTAIN
    return ANOMALY if x['logoff'] > max_logoff else ABSTAIN

## Web - Anomalies only occur when user behavior indicates
@labeling_function()
def down_high(x):
    user = x['user']
    max_down = users_stats.loc[user]['down_whishi']
    # Return a label of ANOMALY if download rate is too high in 1
       ↪ hour,
    # otherwise ABSTAIN
    return ANOMALY if x['down'] > max_down else ABSTAIN

@labeling_function()
def up_high(x):
    user = x['user']
    max_up = users_stats.loc[user]['up_whishi']
    # Return a label of ANOMALY if upload rate is too high in 1 hour,
    # otherwise ABSTAIN
    return ANOMALY if x['up'] > max_up else ABSTAIN

@labeling_function()
def vis_high(x):
    user = x['user']
    max_vis = users_stats.loc[user]['vis_whishi']
    # Return a label of ANOMALY if visit rate is too high in 1 hour,
    # otherwise ABSTAIN
    return ANOMALY if x['vis'] > max_vis else ABSTAIN

@labeling_function()
def conn_high(x):
    user = x['user']
    max_conn = users_stats.loc[user]['conn_whishi']
```

```
106     # Return a label of ANOMALY if connection rate is too high in 1
            ↪ hour,
107     # otherwise ABSTAIN
108     return ANOMALY if x['conn'] > max_conn else ABSTAIN
109 
110 @labeling_function()
111 def disc_high(x):
112     user = x['user']
113     max_disc = users_stats.loc[user]['disc_whishi']
114     # Return a label of ANOMALY if disconnection rate is too high in
            ↪ 1 hour,
115     # otherwise ABSTAIN
116     return ANOMALY if x['disc'] > max_disc else ABSTAIN
117 
118 @labeling_function()
119 def trm_high(x):
120     user = x['user']
121     max_trm = users_stats.loc[user]['trm_whishi']
122     # Return a label of ANOMALY if TRM rate is too high in 1 hour,
123     # otherwise ABSTAIN
124     return ANOMALY if x['trm'] > max_trm else ABSTAIN
125 
126 @labeling_function()
127 def frm_high(x):
128     user = x['user']
129     max_frm = users_stats.loc[user]['frm_whishi']
130     # Return a label of ANOMALY if FRM rate is too high in 1 hour,
131     # otherwise ABSTAIN
132     return ANOMALY if x['frm'] > max_frm else ABSTAIN
133 
134 @labeling_function()
135 def open_high(x):
136     user = x['user']
137     max_open = users_stats.loc[user]['open_whishi']
138     # Return a label of ANOMALY if Open rate is too high in 1 hour,
139     # otherwise ABSTAIN
140     return ANOMALY if x['open'] > max_open else ABSTAIN
141 
142 @labeling_function()
143 def write_high(x):
144     user = x['user']
145     max_write = users_stats.loc[user]['write_whishi']
```

```
146     # Return a label of ANOMALY if Write rate is too high in 1 hour,
147     # otherwise ABSTAIN
148     return ANOMALY if x['write'] > max_write else ABSTAIN
149
150 @labeling_function()
151 def copy_high(x):
152     user = x['user']
153     max_copy = users_stats.loc[user]['copy_whishi']
154     # Return a label of ANOMALY if Copy rate is too high in 1 hour,
155     # otherwise ABSTAIN
156     return ANOMALY if x['copy'] > max_copy else ABSTAIN
157
158 @labeling_function()
159 def delete_high(x):
160     user = x['user']
161     max_delete = users_stats.loc[user]['delete_whishi']
162     # Return a label of ANOMALY if Delete rate is too high in 1 hour,
163     # otherwise ABSTAIN
164     return ANOMALY if x['delete'] > max_delete else ABSTAIN
165
166 @labeling_function()
167 def hour_not_in(x):
168     user = x['user']
169     # Return a label of ANOMALY if hour is in previous data,
170     # otherwise ABSTAIN
171     return ANOMALY if
172         x['hour'] not in users_stats.loc[user]['hour_unique'] else
            ↪ ABSTAIN
173
174 @labeling_function()
175 def dow_not_in(x):
176     user = x['user']
177     # Return a label of ANOMALY if Day of Week is in previous data,
178     # otherwise ABSTAIN
179     return ANOMALY if
180         x['dow'] not in users_stats.loc[user]['dow_unique'] else
            ↪ ABSTAIN
181
182 @labeling_function()
183 def normal_behav(x):
184     user = x['user']
185     max_logon = np.max(users_stats.loc[user]['logon_fliers']) if
```

```
186         users_stats.loc[user]['logon_fliers'].size != 0 else
187         users_stats.loc[user]['logon_whishi']
188     max_logoff = np.max(users_stats.loc[user]['logoff_fliers']) if
189         users_stats.loc[user]['logoff_fliers'].size != 0 else
190             users_stats.loc[user]['logoff_whishi']
191     max_down = np.max(users_stats.loc[user]['down_fliers'])*1.5 if
192         users_stats.loc[user]['down_fliers'].size != 0 else
193         users_stats.loc[user]['down_whishi']*1.5
194     max_up = np.max(users_stats.loc[user]['up_fliers'])*1.5 if
195         users_stats.loc[user]['up_fliers'].size != 0 else
196         users_stats.loc[user]['up_whishi']*1.5
197     max_vis = np.max(users_stats.loc[user]['vis_fliers'])*3 if
198         users_stats.loc[user]['vis_fliers'].size != 0 else
199         users_stats.loc[user]['vis_whishi']*3
200     max_conn = np.max(users_stats.loc[user]['conn_fliers'])*2 if
201         users_stats.loc[user]['conn_fliers'].size != 0 else
202         users_stats.loc[user]['conn_whishi']*2
203     max_disc = np.max(users_stats.loc[user]['disc_fliers'])*2 if
204         users_stats.loc[user]['disc_fliers'].size != 0 else
205         users_stats.loc[user]['disc_whishi']*2
206     max_trm = np.max(users_stats.loc[user]['trm_fliers'])*2 if
207         users_stats.loc[user]['trm_fliers'].size != 0 else
208         users_stats.loc[user]['trm_whishi']*2
209     max_frm = np.max(users_stats.loc[user]['frm_fliers'])*2 if
210         users_stats.loc[user]['frm_fliers'].size != 0 else
211         users_stats.loc[user]['frm_whishi']*2
212     max_open = np.max(users_stats.loc[user]['open_fliers'])*2 if
213         users_stats.loc[user]['open_fliers'].size != 0 else
214         users_stats.loc[user]['open_whishi']*2
215     max_write = np.max(users_stats.loc[user]['write_fliers'])*2 if
216         users_stats.loc[user]['write_fliers'].size != 0 else
217         users_stats.loc[user]['write_whishi']*2
218     max_copy = np.max(users_stats.loc[user]['copy_fliers'])*2 if
219         users_stats.loc[user]['copy_fliers'].size != 0 else
220         users_stats.loc[user]['copy_whishi']*2
221     max_delete = np.max(users_stats.loc[user]['delete_fliers'])*2 if
222         users_stats.loc[user]['delete_fliers'].size != 0 else
223         users_stats.loc[user]['delete_whishi']*2
224
225     # Return a label of ANOMALY if Day of Week is in previous data,
226     # otherwise ABSTAIN
```

```
227     return NORMAL if ((x['logon'] <= max_logon) and (x['logoff'] <=
        ↪ max_logoff) and
228             (x['down'] <= max_down) and (x['up'] <= max_up) and
229             (x['vis'] <= max_vis) and
230             (x['conn'] <= max_conn) and
231             (x['disc'] <= max_disc) and
232             (x['trm'] <= max_trm) and
233             (x['frm'] <= max_frm) and
234             (x['open'] <= max_open) and
235             (x['write'] <= max_write) and
236             (x['copy'] <= max_copy) and
237             (x['delete'] <= max_delete) and
238             (x['hour'] in users_stats.loc[user]['hour_unique']) and
239             (x['dow'] in users_stats.loc[user]['dow_unique'])) else
                ↪ ABSTAIN
240
241
242 lfs = [logon_high,logoff_high,down_high,up_high,vis_high,conn_high,
    ↪ disc_high,
243        trm_high,frm_high,open_high,write_high,copy_high,delete_high,
244        hour_not_in,dow_not_in,normal_behav]
245
246
247
248 ## Aplica as LFs ao dataset de test e obtem as rotulacoes candidatas
249 applier = PandasLFApplier(lfs=lfs)
250
251 L_test = applier.apply(df=df_1hud_1hora_test)
252 ## Salva em arquivo
253 L_test_g_pd = pd.DataFrame(L_test)
254 L_test_g_pd.to_hdf("L_test_g_pd.hdf",key='df')
255 L_test_g_pd = pd.read_hdf("L_test_g_pd.hdf",'df')
256 L_test_g = np.array(L_test_g_pd)
257
258 ### Aplica o Label Modelo para obter o modelo de rotulao final
259 from snorkel.labeling.model import LabelModel
260
261 label_model = LabelModel(cardinality=2, verbose=True)
262 label_model.fit(L_test_g, n_epochs=500, log_freq=100, seed=123)
263
264 ### Obtm rotulao final a partir do modelo construdo
265 labels_g = label_model.predict(L_test_g)
```

```
266
267 ## Salva em arquivo
268 labels_g_pd = pd.DataFrame(labels_g)
269 labels_g_pd.to_hdf("labels_g_pd.hdf",key='df')
270 labels_g_pd = pd.read_hdf("labels_g_pd.hdf",'df')
271 labels_g = np.array(labels_g_pd)
```

III. HIPERPARÂMETROS TESTADOS NA CONFIGURAÇÃO DO EXPERIMENTO

Tabela III.1: Configuração de valores possíveis para os hiperparâmetros dos modelos OCSVM-SGD e IF.

Modelo	Hiperparâmetros	Valores Possíveis
OCSVM-SGD	nu	[**0.01, 0.02**]
	fit_intercept	[**True**]
	max_iter	[10.000, 20.000, **30.000**, ..., 100.000]
	tol	[0.00001, 0.00005, 0.0001,..., **0.001**]
	learning_rate	[**"optimal"**]
IF	n_estimators	[50, **100**, 150, 200, ..., 1000]
	max_samples	[50, 100, 150, ..., **500**, ..., 600]
	contamination	[**0.01, 0.02**]
	max_features	[**1.0**]
	bootstrap	[**False**]
	n_jobs	[**-1**]

Tabela III.2: Configuração de valores possíveis para os hiperparâmetros do modelo SAEBN.

Modelo	Hiperparâmetros	Valores Possíveis
SAEBN	Input	[**4044**]
	Dense 1	Units: [250,270,290,...,**370**,...,750] Kernel initializer: [**"he_normal"**] Activation: [**"elu"**]
	Batch Normalization	Present: [**yes**]
	L1 Regularization	Factor: [0,1e-05,5e-05,1e-04,...,**5e-3**,1e-2]
	Dense 2	Units: [50,75,100,...,**275**,...,300] Kernel initializer: [**"he_normal"**] Activation: [**"elu"**]
	Batch Normalization	Present: [**yes**]
	Dense 3	Units: [20,30,**40**,50,60] Kernel initializer: [**"he_normal"**] Activation: [**"elu"**]
	Batch Normalization	Present: [**yes**]
	Dense 4	Units: **Idêntico a Dense 2**
	Batch Normalization	Present: [**yes**]
	Dense 5	Units: **Idêntico a Dense 1**
	Batch Normalization	Present: [no,**yes**]
	Dense 6	Units: [**13**]
	Parâmetros de treinamento	Loss: [**"mae"**, "mse", "Hubber"] Metrics: ["mean_absolute_error",**"mean_squared_error"**] Optimizer: [**"nadam"**,"adam"] Batch size: [128,256, 512, **1024**]

Tabela III.3: Configuração de valores possíveis para os hiperparâmetros do modelo WEAPON.

Modelo	Hiperparâmetros	Valores Possíveis
WEAPON	Input	deep:[**13**], wide:[**4031**]
	Convolutional-1D	Filters: [1...**7**...13] Kernel size: [**1**...10]
	LSTM 1	Units: [7...**11**...14]
	LSTM 2	Units: [4...**7**...11]
	LSTM 3	Units: **Idêntico a LSTM 2**
	LSTM 4	Units: **Idêntico a LSTM 1**
	CONCAT	[**4031 + 11**]
	Dense 1	Units: [100...**125**...500]
	Batch Normalization	Present: [**no**,yes]
	L1 Regularization	Factor: [**0**...1e-2]
	Dropout	Dropout rate: [**0.0**...0.2]
	Dense 2	Units: [**13**]
	Parâmetros de treinamento	Loss: [**"mae"**, "mse", "Hubber"] Metrics: ["mean_absolute_error",**"mean_squared_error"**] Optimizer: [**"nadam"**,"adam"] Batch size: [128,256, 512, **1024**]

Tabela III.4: Configuração de valores possíveis para os hiperparâmetros do modelo WDSAEBN.

Modelo	**Hiperparâmetros**	**Valores Possíveis**
WDSAEBN	Input deep	deep:[**13**], wide:[**4031**]
	Dense 1	Units: [7,8,9,10,11,**12**,13,14] Kernel initializer: [**"he_normal"**] Activation: [**"elu"**]
	Batch Normalization	Present: [**yes**]
	Dense 2	Units: [3,4,5,6,**7**,8] Kernel initializer: [**"he_normal"**] Activation: [**"elu"**]
	Batch Normalization	Present: [**yes**]
	Dense 3	Units: **Idêntico a Dense 2**
	Batch Normalization	Present: [**yes**]
	Dense 4	Units: **Idêntico a Dense 1**
	Batch Normalization	Present: [**yes**]
	CONCAT	[**4031 + 11**]
	Dense 5	Units: [100,125,...,**275**,...,500] Kernel initializer: [**"he_normal"**] Activation: [**"elu"**]
	Batch Normalization	Present: [no,**yes**]
	L1 Regularization	Factor: [0,1e-05,5e-05,**1e-04**,...,1e-2]
	Dropout	Dropout rate: [0.0, 0.05, 0.1, **0.15**, 0.2]
	Dense 6	Units: [**13**]
	Parâmetros de treinamento	Loss: [**"mae"**, "mse", "Hubber"] Metrics: ["mean_absolute_error",**"mean_squared_error"**] Optimizer: [**"nadam"**,"adam"] Batch size: [128,256, 512, **1024**]

IV. CÓDIGO FONTE DA HIPERPARAMETRIZAÇÃO PARA ENCONTRAR MELHORES MODELOS

IV.1 ALGORITMO PARA ENCONTRAR MELHOR MODELO OCSVM-SGD

```
# Python >=3.5 is required
import sys
assert sys.version_info >= (3, 5)

from ipynb.fs.full.Cert_Aux_Functions2 import *

# Importa a biblioteca pandas
import pandas as pd

# Importa datetime e timedelta
from datetime import datetime, timedelta

# Importa a biblioteca os
import os

## Bibliotecas sklearn
from sklearn.compose import ColumnTransformer
from sklearn.compose import make_column_selector as selector
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
    ↪ , OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder, MinMaxScaler,
    ↪ FunctionTransformer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import TimeSeriesSplit
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor

# One Class SVM
# https://scikit-learn.org/stable/auto_examples/svm/
    ↪ plot_oneclass.html
from sklearn import svm
from sklearn import linear_model
```

```
31
32 #Para as figuras
33 %matplotlib inline
34 import matplotlib as mpl
35 import matplotlib.pyplot as plt
36 from mpl_toolkits import mplot3d
37
38 #Importa bibliotecas Numpy
39 import numpy as np
40
41 from hyperopt import hp, fmin, tpe, STATUS_OK, Trials
42
43 import warnings
44 warnings.filterwarnings("ignore")
45
46 # Logon + HTTP + USB + Device
47 df_lhud_1hora = pd.read_pickle("df_lhud_1hora_file.pkl")
48 df_lhud_1hora.sort_values('date', ascending=True, inplace = True)
49 df_lhud_1hora.reset_index(inplace = True, drop=True)
50
51 ## Separa 4 primeiros meses de dados
52 df_lhud_1hora4m = df_lhud_1hora[(df_lhud_1hora['date'] <= '
      ↪ 2010-05-02')]
53
54 train_index = df_lhud_1hora4m.index[-1]
55 test_index = df_lhud_1hora[(df_lhud_1hora['date'] <= '2010-09-02')]
      ↪ .index[-1]
56
57
58 ## Conjunto de teste
59 df_lhud_1hora_test = df_lhud_1hora.iloc[train_index:test_index]
60
61 ### Transformacoes
62
63 #Nomes das features numericas de acordo com o dataset
64 lhud_numeric_features = ['logon', 'logoff','down','up','vis',
65                 'conn','disc','trm','frm','open',
66                 'write','copy','delete']
67
68 #Nomes das features categoricas - comum a todos os datasets
69
70 #numeric_transformer = StandardScaler()
```

```
71 numeric_transformer = MinMaxScaler()
72 #numeric_transformer = SimpleImputer()
73
74 hour_categories = np.arange(0, 24)
75 dow_categories = np.arange(0, 7)
76 user_categories = df_lhud_1hora4m.user.unique()
77
78 #categorical_features = ['user','hour', 'dow']
79 categorical_features = ['hour', 'dow']
80 categorical_transformer = OneHotEncoder(
81 # categories = [user_categories, hour_categories, dow_categories]
82    categories = [hour_categories, dow_categories]
83 )
84
85 user_feature = ['user']
86 #user_transformer = OrdinalEncoder(categories = [user_categories])
87 user_transformer = OneHotEncoder(categories = [user_categories])
88
89 lhud_preprocessor = ColumnTransformer(
90    transformers=[
91       ('lhud_num', numeric_transformer, lhud_numeric_features),
92       ('lhud_cat', categorical_transformer, categorical_features),
93       ('lhud_user', user_transformer, user_feature),
94    ])
95
96 #Transformacoes
97 columns = lhud_numeric_features + categorical_features +
      ↪ user_feature
98 trans_lhud_4m = lhud_preprocessor.fit_transform(df_lhud_1hora4m[
      ↪ columns])
99 trans_lhud_test = lhud_preprocessor.transform(df_lhud_1hora_test[
      ↪ columns])
100 trans_lhud = lhud_preprocessor.transform(df_lhud_1hora[columns])
101
102 ### Carrega Labels preditos do Snorkel
103 labels_g_pd = pd.read_hdf("labels_g_pd.hdf",'df')
104 labels_g_pd['anom'] = np.where((labels_g_pd[0]== 1),-1,1)
105 labels_g_pd[labels_g_pd['anom'] == -1].shape, labels_g_pd[
      ↪ labels_g_pd['anom'] == 1].shape
106
107 def objective(space):
108
```

```
    clf_SGDOC=linear_model.SGDOneClassSVM(
        nu=space['nu_s'], fit_intercept=space['fiti_s'], max_iter=
            ↪ space['maxi_s'],
        tol=space['to_s'], #eta0=0.0,
        learning_rate=space['lr_s']
    )


    clf_SGDOC.fit(trans_lhud_4m)

    pred1 = clf_SGDOC.predict(trans_lhud_test)
    dec1 = clf_SGDOC.decision_function(trans_lhud_test)

    anomalyScoresSGDOC = df_lhud_1hora_test
    anomalyScoresSGDOC['scores']=dec1
    anomalyScoresSGDOC['anom']=pred1

    a,p,r,f,cm,auc_sc = benchmark_snorkel(labels_g_pd,
        ↪ anomalyScoresSGDOC)

    global index
    scores_df.loc[index,:]=np.array([index,space,a,p,r,f,auc_sc,
        ↪ np.reshape(cm,(4))],dtype=object)

    index=index+1

    print(space,r)

    return {'loss': -r, 'status': STATUS_OK, 'space': space,
          'model': clf_SGDOC, 'f1_score': f,'auc_sc': auc_sc,
         'precision': p, 'recall': r, 'c_matrix': cm}

space ={'nu_s': hp.choice('nu_s', [0.01, 0.02]),
       'fiti_s': hp.choice('fiti_s', [True]),
       'maxi_s': hp.choice('maxi_s', np.arange(10000,100000,10000)),
       'to_s': hp.choice('to_s', [1e-3, 5e-4, 1e-4, 5e-5, 1e-5]),
       'lr_s' : hp.choice('lr_s', ["optimal"]),
    }


scores_df = pd.DataFrame(columns=["Model","Params","Accuracy","
    ↪ Precision","Recall","F1-Score","ROC-AUC", "CM"])
```

```
index=0

trials = Trials()
best = fmin(fn=objective,
          space=space,
          algo=tpe.suggest,
          max_evals=300,
          trials=trials)

## Melhor modelo pelas metricas Recall e ROC-AUC
scores_df.sort_values(['Recall','ROC-AUC'], ascending=[False,False])
```

IV.2 ALGORITMO PARA ENCONTRAR MELHOR MODELO IF

```
# Python >=3.5 is required
import sys
assert sys.version_info >= (3, 5)

from ipynb.fs.full.Cert_Aux_Functions2 import *

# Importa a biblioteca pandas
import pandas as pd

# Importa datetime e timedelta
from datetime import datetime, timedelta

# Importa a biblioteca os
import os

## Bibliotecas sklearn
from sklearn.compose import ColumnTransformer
from sklearn.compose import make_column_selector as selector
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
    ↪ , OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder, MinMaxScaler,
    ↪ FunctionTransformer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import TimeSeriesSplit
```

```
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor

# One Class SVM
# https://scikit-learn.org/stable/auto_examples/svm/plot_oneclass.html
from sklearn import svm
from sklearn import linear_model

#Para as figuras
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

#Importa bibliotecas Numpy
import numpy as np

from hyperopt import hp, fmin, tpe, STATUS_OK, Trials

import warnings
warnings.filterwarnings("ignore")

# Logon + HTTP + USB + Device
df_lhud_1hora = pd.read_pickle("df_lhud_1hora_file.pkl")
df_lhud_1hora.sort_values('date', ascending=True, inplace = True)
df_lhud_1hora.reset_index(inplace = True, drop=True)

## Separa 4 primeiros meses de dados
df_lhud_1hora4m = df_lhud_1hora[(df_lhud_1hora['date'] <= '2010-05-02')]

train_index = df_lhud_1hora4m.index[-1]
test_index = df_lhud_1hora[(df_lhud_1hora['date'] <= '2010-09-02')].index[-1]


## Conjunto de teste
df_lhud_1hora_test = df_lhud_1hora.iloc[train_index:test_index]

### Transformacoes

```

```
#Nomes das features numericas de acordo com o dataset
lhud_numeric_features = ['logon', 'logoff','down','up','vis',
                'conn','disc','trm','frm','open',
                'write','copy','delete']

#Nomes das features categoricas - comum a todos os datasets

#numeric_transformer = StandardScaler()
numeric_transformer = MinMaxScaler()
#numeric_transformer = SimpleImputer()

hour_categories = np.arange(0, 24)
dow_categories = np.arange(0, 7)
user_categories = df_lhud_1hora4m.user.unique()

#categorical_features = ['user','hour', 'dow']
categorical_features = ['hour', 'dow']
categorical_transformer = OneHotEncoder(
# categories = [user_categories, hour_categories, dow_categories]
   categories = [hour_categories, dow_categories]
)

user_feature = ['user']
#user_transformer = OrdinalEncoder(categories = [user_categories])
user_transformer = OneHotEncoder(categories = [user_categories])

lhud_preprocessor = ColumnTransformer(
   transformers=[
       ('lhud_num', numeric_transformer, lhud_numeric_features),
       ('lhud_cat', categorical_transformer, categorical_features),
       ('lhud_user', user_transformer, user_feature),
   ])

#Transformacoes
columns = lhud_numeric_features + categorical_features +
   ↪ user_feature
trans_lhud_4m = lhud_preprocessor.fit_transform(df_lhud_1hora4m[
   ↪ columns])
trans_lhud_test = lhud_preprocessor.transform(df_lhud_1hora_test[
   ↪ columns])
trans_lhud = lhud_preprocessor.transform(df_lhud_1hora[columns])

```

```
### Carrega Labels preditos do Snorkel
labels_g_pd = pd.read_hdf("labels_g_pd.hdf",'df')
labels_g_pd['anom'] = np.where((labels_g_pd[0]== 1),-1,1)
labels_g_pd[labels_g_pd['anom'] == -1].shape, labels_g_pd[
    ↪ labels_g_pd['anom'] == 1].shape

def objective(space):

    clf_Isol=IsolationForest(n_estimators=space["ne"], max_samples=
        ↪ space["maxi_s"], contamination=space["cont"],
                    max_features=1.0, bootstrap=False, n_jobs=space["
                        ↪ n_jobs"], verbose=0)
    )


    clf_Isol.fit(trans_lhud_4m)

    pred1 = clf_Isol.predict(trans_lhud_test)
    dec1 = clf_Isol.decision_function(trans_lhud_test)

    anomalyScoresIsol = df_lhud_1hora_test
    anomalyScoresIsol['scores']=dec1
    anomalyScoresIsol['anom']=pred1

    a,p,r,f,cm,auc_sc = benchmark_snorkel(labels_g_pd,
        ↪ anomalyScoresIsol)

    global index
    scores_df.loc[index,:]=np.array([index,space,a,p,r,f,auc_sc,
        ↪ np.reshape(cm,(4))],dtype=object)

    index=index+1

    print(space,r)

    return {'loss': -r, 'status': STATUS_OK, 'space': space,
          'model': clf_Isol, 'f1_score': f,'auc_sc': auc_sc,
         'precision': p, 'recall': r, 'c_matrix': cm}

space ={'ne': hp.choice('ne', np.arange(50,1000,50)),
       'maxi_s': hp.choice('maxi_s', np.arange(50,600,50)),
       'cont': hp.choice('cont', [0.01, 0.02]),
```

```
        'n_jobs' : hp.choice('n_jobs', [-1])
    }

scores_df = pd.DataFrame(columns=["Model","Params","Accuracy","
    ↪ Precision","Recall","F1-Score","ROC-AUC", "CM"])
index=0

trials = Trials()
best = fmin(fn=objective,
          space=space,
          algo=tpe.suggest,
          max_evals=300,
          trials=trials)

## Melhor modelo pelas metricas Recall e ROC-AUC
scores_df.sort_values(['Recall','ROC-AUC'], ascending=[False,False])
```

IV.3 ALGORITMO PARA ENCONTRAR MELHOR MODELO SAEBN

```
# Python >=3.5 is required
import sys
assert sys.version_info >= (3, 5)

from ipynb.fs.full.Cert_Aux_Functions2 import *

# Importa a biblioteca pandas
import pandas as pd

# Importa datetime e timedelta
from datetime import datetime, timedelta

# Importa a biblioteca os
import os
from pathlib import Path

## Bibliotecas sklearn
from sklearn.compose import ColumnTransformer
from sklearn.compose import make_column_selector as selector
from sklearn.pipeline import Pipeline
```

```
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
    ↪ , OneHotEncoder, LabelEncoder
from sklearn.preprocessing import OrdinalEncoder, MinMaxScaler,
    ↪ FunctionTransformer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.impute import SimpleImputer

#Para as figuras
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

#Importa bibliotecas Numpy
import numpy as np

# TensorFlow >=2.0 is required
import tensorflow as tf
from tensorflow import keras
assert tf.__version__ >= "2.0"

from keras import optimizers, Sequential
from keras.models import Model
#from keras.utils import plot_model
from keras.layers import Dense, LSTM, RepeatVector, TimeDistributed
from keras.callbacks import ModelCheckpoint, TensorBoard

from hyperopt import hp, fmin, tpe, STATUS_OK, Trials

# Logon + HTTP + USB + Device
df_lhud_1hora = pd.read_pickle("df_lhud_1hora_file.pkl")
df_lhud_1hora.sort_values('date', ascending=True, inplace = True)
df_lhud_1hora.reset_index(inplace = True, drop=True)

## Separa 4 primeiros meses de dados
df_lhud_1hora4m = df_lhud_1hora[(df_lhud_1hora['date'] <= '
    ↪ 2010-05-02')]

```

```
train_index = df_lhud_1hora4m.index[-1]
test_index = df_lhud_1hora[(df_lhud_1hora['date'] <= '2010-09-02')]
    ↪ .index[-1]


## Conjunto de teste
df_lhud_1hora_test = df_lhud_1hora.iloc[train_index:test_index]

### Transformacoes

#Nomes das features numericas de acordo com o dataset
lhud_numeric_features = ['logon', 'logoff','down','up','vis',
                'conn','disc','trm','frm','open',
                'write','copy','delete']

#Nomes das features categoricas - comum a todos os datasets

#numeric_transformer = StandardScaler() ## removing the mean and
    ↪ scaling to unit variance.
numeric_transformer = MinMaxScaler() ## Varia de 0 a 1
#numeric_transformer = SimpleImputer()

hour_categories = np.arange(0, 24)
dow_categories = np.arange(0, 7)
user_categories = df_lhud_1hora4m.user.unique()

#categorical_features = ['user','hour', 'dow']
categorical_features = ['hour', 'dow']
categorical_transformer = OneHotEncoder(
# categories = [user_categories, hour_categories, dow_categories]
    categories = [hour_categories, dow_categories]
)

user_feature = ['user']
user_transformer = OrdinalEncoder(categories = [user_categories])
user_transformer_wd = OneHotEncoder(categories = [user_categories])

lhud_preprocessor = ColumnTransformer(
    transformers=[
        ('lhud_num', numeric_transformer, lhud_numeric_features),
        ('lhud_cat', categorical_transformer, categorical_features),
        ('lhud_user', user_transformer_wd, user_feature),
```

```
100     ])
101 
102 lhud_preprocessor_deep = ColumnTransformer(
103     transformers=[
104         ('lhud_num', numeric_transformer, lhud_numeric_features),
105     ])
106 
107 lhud_preprocessor_wide = ColumnTransformer(
108     transformers=[
109         ('lhud_cat', categorical_transformer, categorical_features),
110         ('lhud_user_wd', user_transformer_wd, user_feature),
111     ])
112 
113 #Transformacoes simples, com dados todos juntos
114 columns = lhud_numeric_features + categorical_features +
      ↪ user_feature
115 trans_lhud_4m = lhud_preprocessor.fit_transform(df_lhud_1hora4m[
      ↪ columns])
116 trans_lhud_test = lhud_preprocessor.transform(df_lhud_1hora_test[
      ↪ columns])
117 trans_lhud = lhud_preprocessor.transform(df_lhud_1hora[columns])
118 trans_lhud_4m_13 = trans_lhud_4m[:,:13].toarray()
119 trans_lhud_test_13 = trans_lhud_test[:,:13].toarray()
120 
121 #Transformacoes deep
122 columns_deep = lhud_numeric_features# + categorical_features +
      ↪ user_feature
123 trans_lhud_4m_deep = lhud_preprocessor_deep.fit_transform(
      ↪ df_lhud_1hora4m[columns_deep])
124 trans_lhud_test_deep = lhud_preprocessor_deep.transform(
      ↪ df_lhud_1hora_test[columns_deep])
125 
126 #Transformacoes wide
127 columns_wide = categorical_features + user_feature
128 trans_lhud_4m_wide = lhud_preprocessor_wide.fit_transform(
      ↪ df_lhud_1hora4m[columns_wide])
129 trans_lhud_test_wide = lhud_preprocessor_wide.transform(
      ↪ df_lhud_1hora_test[columns_wide])
130 
131 ### Carrega Labels preditos do Snorkel
132 labels_g_pd = pd.read_hdf("labels_g_pd.hdf",'df')
133 labels_g_pd['anom'] = np.where((labels_g_pd[0]== 1),-1,1)
```

```
labels_g_pd[labels_g_pd['anom'] == -1].shape, labels_g_pd[
    ↪ labels_g_pd['anom'] == 1].shape

# PAra RNA com entradas 4044

ds_train = tf.data.Dataset.from_tensor_slices(
    ↪ convert_sparse_matrix_to_sparse_tensor(trans_lhud_4m))
output_train = tf.data.Dataset.from_tensor_slices(trans_lhud_4m_13)
ds_train = tf.data.Dataset.zip(({"input": ds_train}, {'output':
    ↪ output_train}))

ds_test = tf.data.Dataset.from_tensor_slices(
    ↪ convert_sparse_matrix_to_sparse_tensor(trans_lhud_test))
output_test = tf.data.Dataset.from_tensor_slices(trans_lhud_test_13)
ds_test = tf.data.Dataset.zip(({"input": ds_test}, {'output':
    ↪ output_test}))

ds_train = ds_train.batch(1024).cache().prefetch(4)
ds_test = ds_test.batch(1024).cache().prefetch(4)

ds_train.element_spec, ds_test.element_spec


def objective(space):

    keras.backend.clear_session()

    input_feat = keras.layers.Input(shape=[4044], name="input",
        ↪ sparse=True, dtype=tf.float64)

    sae = keras.models.Sequential([

        #keras.layers.BatchNormalization(),

        keras.layers.Dense(space['first'],
                           kernel_initializer=space['kernel_init']),

        keras.layers.BatchNormalization(),
        keras.layers.Activation(space['activ']),

        keras.layers.ActivityRegularization(space['l1_reg']),

```

```
170         keras.layers.Dense(space['second'],
171                            kernel_initializer=space['kernel_init']),
172 
173         keras.layers.BatchNormalization(),
174         keras.layers.Activation(space['activ']),
175 
176         keras.layers.Dense(space['middle'],
177                            kernel_initializer=space['kernel_init']),
178 
179         keras.layers.BatchNormalization(),
180         keras.layers.Activation(space['activ']),
181 
182         keras.layers.Dense(space['second'],
183                            kernel_initializer=space['kernel_init']),
184 
185         keras.layers.BatchNormalization(),
186         keras.layers.Activation(space['activ']),
187 
188         keras.layers.Dense(space['first'],
189                            kernel_initializer=space['kernel_init']),
190 
191         keras.layers.BatchNormalization(),
192         keras.layers.Activation(space['activ']),
193 
194     ])
195 
196     sae = sae(input_feat)
197 
198     output = keras.layers.Dense(13, name="output")(sae)
199 
200     model = keras.models.Model(inputs=[input_feat], outputs=[output])
201 
202     model.compile(loss=space['loss_ob'], optimizer=space['optimizer'
        ↪ ], metrics=space['metrics'])
203 
204     history = model.fit(ds_train, epochs=space['epochs'], verbose=0)
205 
206     y_pred = model.predict(ds_test)
207 
208     anomalyScores, den_thres = dnn_tf_anomScores(
209     y_pred, trans_1hud_test_13, df_1hud_1hora_test, 98.0)
210 
```

```
    a,p,r,f,cm,auc_sc = benchmark_snorkel(labels_g_pd,anomalyScores)

    global index
    scores_df.loc[index,:]=np.array([index,space,a,p,r,f,auc_sc,
       ↪ np.reshape(cm,(4))],dtype=object)

    index=index+1

    print(space,r)

    return {'loss': -r, 'status': STATUS_OK, 'space': space,
          'model': model, 'f1_score': f,'auc_sc': auc_sc,
         'precision': p, 'recall': r, 'c_matrix': cm}


space ={'first': hp.choice("first", np.arange(250, 500, 20)),
       'second': hp.choice('second', np.arange(50, 300, 25)),
       'middle': hp.choice('middle', np.arange(20, 60, 10)),
       'kernel_init' : hp.choice('kernel_init', ["he_normal"]),
       'activ' : hp.choice('activ', ["elu"]),
       #'dropout' : hp.choice('dropout', [0.0, 0.05, 0.1, 0.15, 0.2])
          ↪ ,
       'loss_ob' : hp.choice('loss_ob', ["mae"]),
       'optimizer' : hp.choice('optimizer', ["nadam"]),
       'metrics' : hp.choice('metrics', ["mean_squared_error"]),
       'epochs' : hp.choice('epochs', [10]),
       'l1_reg' : hp.choice('l1_reg', [0.00001, 0.00005, 0.0001,0
          ↪ .0005,0.001,0.005,0.01])
    }

trials = Trials()
best = fmin(fn=objective,
          space=space,
          algo=tpe.suggest,
          max_evals=300,
          trials=trials)

## Melhor modelo pelas metricas Recall e ROC-AUC
scores_df.sort_values(['Recall','ROC-AUC'], ascending=[False,False])
```

IV.4 ALGORITMO PARA ENCONTRAR MELHOR MODELO WEAPON

```
# Python >=3.5 is required
import sys
assert sys.version_info >= (3, 5)

from ipynb.fs.full.Cert_Aux_Functions2 import *

# Importa a biblioteca pandas
import pandas as pd

# Importa datetime e timedelta
from datetime import datetime, timedelta

# Importa a biblioteca os
import os
from pathlib import Path

## Bibliotecas sklearn
from sklearn.compose import ColumnTransformer
from sklearn.compose import make_column_selector as selector
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
    ↪ , OneHotEncoder, LabelEncoder
from sklearn.preprocessing import OrdinalEncoder, MinMaxScaler,
    ↪ FunctionTransformer

#Para as figuras
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

#Importa bibliotecas Numpy
import numpy as np

# TensorFlow >=2.0 is required
import tensorflow as tf
from tensorflow import keras
assert tf.__version__ >= "2.0"

```

```
38 from keras import optimizers, Sequential
39 from keras.models import Model
40 #from keras.utils import plot_model
41 from keras.layers import Dense, LSTM, RepeatVector, TimeDistributed
42 from keras.callbacks import ModelCheckpoint, TensorBoard
43
44 from hyperopt import hp, fmin, tpe, STATUS_OK, Trials
45
46
47
48 # Logon + HTTP + USB + Device
49 df_lhud_1hora = pd.read_pickle("df_lhud_1hora_file.pkl")
50 df_lhud_1hora.sort_values('date', ascending=True, inplace = True)
51 df_lhud_1hora.reset_index(inplace = True, drop=True)
52
53 ## Separa 4 primeiros meses de dados
54 df_lhud_1hora4m = df_lhud_1hora[(df_lhud_1hora['date'] <= '
      ↪ 2010-05-02')]
55
56 train_index = df_lhud_1hora4m.index[-1]
57 test_index = df_lhud_1hora[(df_lhud_1hora['date'] <= '2010-09-02')]
      ↪ .index[-1]
58
59
60 ## Conjunto de teste
61 df_lhud_1hora_test = df_lhud_1hora.iloc[train_index:test_index]
62
63 ### Transformacoes
64
65 #Nomes das features numericas de acordo com o dataset
66 lhud_numeric_features = ['logon', 'logoff','down','up','vis',
67                 'conn','disc','trm','frm','open',
68                 'write','copy','delete']
69
70 #Nomes das features categoricas - comum a todos os datasets
71
72 #numeric_transformer = StandardScaler() ## removing the mean and
      ↪ scaling to unit variance.
73 numeric_transformer = MinMaxScaler() ## Varia de 0 a 1
74 #numeric_transformer = SimpleImputer()
75
76 hour_categories = np.arange(0, 24)
```

```
77 dow_categories = np.arange(0, 7)
78 user_categories = df_lhud_1hora4m.user.unique()
79
80 #categorical_features = ['user','hour', 'dow']
81 categorical_features = ['hour', 'dow']
82 categorical_transformer = OneHotEncoder(
83 # categories = [user_categories, hour_categories, dow_categories]
84    categories = [hour_categories, dow_categories]
85 )
86
87 user_feature = ['user']
88 user_transformer = OrdinalEncoder(categories = [user_categories])
89 user_transformer_wd = OneHotEncoder(categories = [user_categories])
90
91 lhud_preprocessor = ColumnTransformer(
92    transformers=[
93       ('lhud_num', numeric_transformer, lhud_numeric_features),
94       ('lhud_cat', categorical_transformer, categorical_features),
95       ('lhud_user', user_transformer_wd, user_feature),
96    ])
97
98 lhud_preprocessor_deep = ColumnTransformer(
99    transformers=[
100       ('lhud_num', numeric_transformer, lhud_numeric_features),
101    ])
102
103 lhud_preprocessor_wide = ColumnTransformer(
104    transformers=[
105       ('lhud_cat', categorical_transformer, categorical_features),
106       ('lhud_user_wd', user_transformer_wd, user_feature),
107    ])
108
109 #Transformacoes simples, com dados todos juntos
110 columns = lhud_numeric_features + categorical_features +
      ↪ user_feature
111 trans_lhud_4m = lhud_preprocessor.fit_transform(df_lhud_1hora4m[
      ↪ columns])
112 trans_lhud_test = lhud_preprocessor.transform(df_lhud_1hora_test[
      ↪ columns])
113 trans_lhud = lhud_preprocessor.transform(df_lhud_1hora[columns])
114 trans_lhud_4m_13 = trans_lhud_4m[:,:13].toarray()
115 trans_lhud_test_13 = trans_lhud_test[:,:13].toarray()
```

```
116
117 #Transformacoes deep
118 columns_deep = lhud_numeric_features# + categorical_features +
      ↪ user_feature
119 trans_lhud_4m_deep = lhud_preprocessor_deep.fit_transform(
      ↪ df_lhud_1hora4m[columns_deep])
120 trans_lhud_test_deep = lhud_preprocessor_deep.transform(
      ↪ df_lhud_1hora_test[columns_deep])
121
122 #Transformacoes wide
123 columns_wide = categorical_features + user_feature
124 trans_lhud_4m_wide = lhud_preprocessor_wide.fit_transform(
      ↪ df_lhud_1hora4m[columns_wide])
125 trans_lhud_test_wide = lhud_preprocessor_wide.transform(
      ↪ df_lhud_1hora_test[columns_wide])
126
127 ### Carrega Labels preditos do Snorkel
128 labels_g_pd = pd.read_hdf("labels_g_pd.hdf",'df')
129 labels_g_pd['anom'] = np.where((labels_g_pd[0]== 1),-1,1)
130 labels_g_pd[labels_g_pd['anom'] == -1].shape, labels_g_pd[
      ↪ labels_g_pd['anom'] == 1].shape
131
132 # PAra RNN com entradas deep 13 e wide 4031
133
134 n_steps = 1
135 wind = n_steps
136 batch_size = 1024
137
138 ds_deep_train = tf.data.Dataset.from_tensor_slices(
      ↪ trans_lhud_4m_deep)
139 ds_wide_train = tf.data.Dataset.from_tensor_slices(
      ↪ convert_sparse_matrix_to_sparse_tensor(trans_lhud_4m_wide))
140
141 ds_deep_train = ds_deep_train.window(wind, shift=n_steps,
      ↪ drop_remainder=True
142                                 ).flat_map(lambda window: window.batch(
                                        ↪ wind)
143                                 )
144
145 ds_wide_train = ds_wide_train.window(wind, shift=n_steps,
      ↪ drop_remainder=True
```

```
                                ).flat_map(lambda window: window.batch(
                                    ↪ wind)
                                )

ds_train = tf.data.Dataset.zip(({"deep_input": ds_deep_train, "
    ↪ wide_input": ds_wide_train}, ds_deep_train))

ds_train = ds_train.batch(batch_size).cache().prefetch(1)


ds_deep_test = tf.data.Dataset.from_tensor_slices(
    ↪ trans_lhud_test_deep)
ds_wide_test = tf.data.Dataset.from_tensor_slices(
    ↪ convert_sparse_matrix_to_sparse_tensor(trans_lhud_test_wide))

ds_deep_test = ds_deep_test.window(wind, shift=n_steps,
    ↪ drop_remainder=True
                                ).flat_map(lambda window: window.batch(
                                    ↪ wind)
                                )

ds_wide_test = ds_wide_test.window(wind, shift=n_steps,
    ↪ drop_remainder=True
                                ).flat_map(lambda window: window.batch(
                                    ↪ wind)
                                )


ds_test = tf.data.Dataset.zip(({"deep_input": ds_deep_test, "
    ↪ wide_input": ds_wide_test}, ds_deep_test))

ds_test = ds_test.batch(batch_size).cache().prefetch(1)

ds_train.element_spec, ds_test.element_spec

def objective(space):

    input_feat = keras.layers.Input(shape=[None, 13], name="
        ↪ deep_input")
    input_time_user = keras.layers.Input(shape=[None, 4031], name="
        ↪ wide_input")

```

```
177     conv1d_ae = keras.models.Sequential([
178     keras.layers.Conv1D(filters=space['filters'], kernel_size=space['
            ↪ kernel_s'], strides=1, padding="same",
179                         activation="selu", input_shape=[None, 13]),
180     keras.layers.LSTM(space['first'], activation='tanh',
            ↪ return_sequences=True),
181     keras.layers.LSTM(space['intermediate'], activation='tanh',
            ↪ return_sequences=False),
182     keras.layers.RepeatVector(wind),
183     keras.layers.LSTM(space['intermediate'], activation='tanh',
            ↪ return_sequences=True),
184     keras.layers.LSTM(space['first'], activation='tanh',
            ↪ return_sequences=True),
185     ])
186
187     conv1d_ae = conv1d_ae(input_feat)
188
189     concat = keras.layers.concatenate([input_time_user, conv1d_ae])
190
191     #last = keras.models.Sequential()
192     #last.add(keras.layers.Dense(space['last'], kernel_initializer=
            ↪ space['kernel_init']))
193     #last.add(keras.layers.BatchNormalization())
194     #last.add(keras.layers.Activation("elu"))
195     #last.add(keras.layers.Dropout(space['dropout']))
196
197     last = keras.models.Sequential()
198     last.add(keras.layers.Dense(space['last'],
199                                 kernel_initializer=space['kernel_init']))
200     if space['last_batch_layer']:
201         last.add(keras.layers.BatchNormalization())
202     last.add(keras.layers.Activation("elu"))
203     if space['l1reg_layer']:
204         last.add(keras.layers.ActivityRegularization(space['l1_reg']))
205     if space['dropout_layer']:
206         last.add(keras.layers.Dropout(space['dropout']))
207     last = last(concat)
208
209     output = keras.layers.Dense(13, name="output")(last)
210
211     model = keras.models.Model(inputs=[input_feat, input_time_user],
            ↪ outputs=[output])
```

```
212
213     model.compile(loss=space['loss_ob'], optimizer=space['optimizer'
            ↪ ], metrics=space['metrics'])
214
215     history = model.fit(ds_train, epochs=space['epochs'], verbose=0)
216
217     y_pred = model.predict(ds_test)
218
219     #anomalyScores, den_thres = anomScores_Snorkel(
220     anomalyScores, den_thres = dnn_tf_anomScores(
221     y_pred, trans_lhud_test_deep, df_lhud_1hora_test, 99.0)
222
223     a,p,r,f,cm,auc_sc = benchmark_snorkel(labels_g_pd,anomalyScores)
224
225     global index
226     scores_df.loc[index,:]=np.array([index,space,a,p,r,f,auc_sc,
            ↪ np.reshape(cm,(4))],dtype=object)
227
228     index=index+1
229
230     print(space,r)
231
232     return {'loss': -r, 'status': STATUS_OK, 'space': space,
233             'model': model, 'f1_score': f,'auc_sc': auc_sc,
234            'precision': p, 'recall': r, 'c_matrix': cm}
235
236 space ={'filters': hp.choice('filters', [1, 2, 3, 4, 5, 6, 7, 8, 9,
        ↪ 10, 11, 12, 13]),
237         'kernel_s': hp.choice('kernel_s', [1, 2, 3, 4, 5, 6, 7, 8, 9,
                ↪ 10]),
238         'first': hp.choice('first', np.arange(7,14,1)),
239         'intermediate': hp.choice('intermediate', np.arange(4, 11, 1))
                ↪ ,
240         'last' : hp.choice('last', np.arange(100,500,25)),
241         'kernel_init' : hp.choice('kernel_init', ["he_normal"]),
242         'activ' : hp.choice('activ', ["elu"]),
243         'loss_ob' : hp.choice('loss_ob', ["mae", "mse", "Huber"])
244         'optimizer' : hp.choice('optimizer', ["nadam", "adam"]),
245         'metrics' : hp.choice('metrics', ["mean_absolute_error","
                ↪ mean_squared_error"]),
246         'epochs' : hp.choice('epochs', [10]),
```

```
247         'l1_reg' : hp.choice('l1_reg', [0.00001, 0.00005, 0.0001,0
                ↪ .0005,0.001,0.005,0.01]),
248         'dropout' : hp.choice('dropout', [0.0, 0.05, 0.1, 0.15, 0.2]),
249         'last_batch_layer': hp.choice('last_batch_layer', [True, False
                ↪ ]),
250         'l1reg_layer': hp.choice('l1reg_layer', [True, False]),
251         'dropout_layer':hp.choice('dropout_layer', [True, False])
252     }
253
254
255 scores_df = pd.DataFrame(columns=["Model","Params","Accuracy","
        ↪ Precision","Recall","F1-Score","ROC-AUC", "CM"])
256 index=0
257
258 trials = Trials()
259 best = fmin(fn=objective,
260           space=space,
261           algo=tpe.suggest,
262           max_evals=300,
263           trials=trials)
264
265 ## Melhor modelo pelas metricas Recall e ROC-AUC
266 scores_df.sort_values(['Recall','ROC-AUC'], ascending=[False,False])
```

IV.5 ALGORITMO PARA ENCONTRAR MELHOR MODELO WDSAEBN

```
1
2 # Python >=3.5 is required
3 import sys
4 assert sys.version_info >= (3, 5)
5
6 from ipynb.fs.full.Cert_Aux_Functions2 import *
7
8 # Importa a biblioteca pandas
9 import pandas as pd
10
11 # Importa datetime e timedelta
12 from datetime import datetime, timedelta
13
```

```
# Importa a biblioteca os
import os
from pathlib import Path

## Bibliotecas sklearn
from sklearn.compose import ColumnTransformer
from sklearn.compose import make_column_selector as selector
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
    ↪ , OneHotEncoder, LabelEncoder
from sklearn.preprocessing import OrdinalEncoder, MinMaxScaler,
    ↪ FunctionTransformer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.impute import SimpleImputer

#Para as figuras
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

#Importa bibliotecas Numpy
import numpy as np

# TensorFlow >=2.0 is required
import tensorflow as tf
from tensorflow import keras
assert tf.__version__ >= "2.0"

from keras import optimizers, Sequential
from keras.models import Model
#from keras.utils import plot_model
from keras.layers import Dense, LSTM, RepeatVector, TimeDistributed
from keras.callbacks import ModelCheckpoint, TensorBoard

from hyperopt import hp, fmin, tpe, STATUS_OK, Trials

import warnings
```

```
54 warnings.filterwarnings("ignore")
55
56 # Logon + HTTP + USB + Device
57 df_lhud_1hora = pd.read_pickle("df_lhud_1hora_file.pkl")
58 df_lhud_1hora.sort_values('date', ascending=True, inplace = True)
59 df_lhud_1hora.reset_index(inplace = True, drop=True)
60
61 ## Separa 4 primeiros meses de dados
62 df_lhud_1hora4m = df_lhud_1hora[(df_lhud_1hora['date'] <= '
      ↪ 2010-05-02')]
63
64 train_index = df_lhud_1hora4m.index[-1]
65 test_index = df_lhud_1hora[(df_lhud_1hora['date'] <= '2010-09-02')]
      ↪ .index[-1]
66
67
68 ## Conjunto de teste
69 df_lhud_1hora_test = df_lhud_1hora.iloc[train_index:test_index]
70
71 ### Transformacoes
72
73 #Nomes das features numericas de acordo com o dataset
74 lhud_numeric_features = ['logon', 'logoff','down','up','vis',
75                 'conn','disc','trm','frm','open',
76                 'write','copy','delete']
77
78 #Nomes das features categoricas - comum a todos os datasets
79
80 #numeric_transformer = StandardScaler() ## removing the mean and
      ↪ scaling to unit variance.
81 numeric_transformer = MinMaxScaler() ## Varia de 0 a 1
82 #numeric_transformer = SimpleImputer()
83
84 hour_categories = np.arange(0, 24)
85 dow_categories = np.arange(0, 7)
86 user_categories = df_lhud_1hora4m.user.unique()
87
88 #categorical_features = ['user','hour', 'dow']
89 categorical_features = ['hour', 'dow']
90 categorical_transformer = OneHotEncoder(
91 # categories = [user_categories, hour_categories, dow_categories]
92    categories = [hour_categories, dow_categories]
```

```
)

user_feature = ['user']
user_transformer = OrdinalEncoder(categories = [user_categories])
user_transformer_wd = OneHotEncoder(categories = [user_categories])

lhud_preprocessor = ColumnTransformer(
    transformers=[
        ('lhud_num', numeric_transformer, lhud_numeric_features),
        ('lhud_cat', categorical_transformer, categorical_features),
        ('lhud_user', user_transformer_wd, user_feature),
    ])

lhud_preprocessor_deep = ColumnTransformer(
    transformers=[
        ('lhud_num', numeric_transformer, lhud_numeric_features),
    ])

lhud_preprocessor_wide = ColumnTransformer(
    transformers=[
        ('lhud_cat', categorical_transformer, categorical_features),
        ('lhud_user_wd', user_transformer_wd, user_feature),
    ])

#Transformacoes simples, com dados todos juntos
columns = lhud_numeric_features + categorical_features +
    ↪ user_feature
trans_lhud_4m = lhud_preprocessor.fit_transform(df_lhud_1hora4m[
    ↪ columns])
trans_lhud_test = lhud_preprocessor.transform(df_lhud_1hora_test[
    ↪ columns])
trans_lhud = lhud_preprocessor.transform(df_lhud_1hora[columns])
trans_lhud_4m_13 = trans_lhud_4m[:,:13].toarray()
trans_lhud_test_13 = trans_lhud_test[:,:13].toarray()

#Transformacoes deep
columns_deep = lhud_numeric_features# + categorical_features +
    ↪ user_feature
trans_lhud_4m_deep = lhud_preprocessor_deep.fit_transform(
    ↪ df_lhud_1hora4m[columns_deep])
trans_lhud_test_deep = lhud_preprocessor_deep.transform(
    ↪ df_lhud_1hora_test[columns_deep])
```

```
129
130 #Transformacoes wide
131 columns_wide = categorical_features + user_feature
132 trans_lhud_4m_wide = lhud_preprocessor_wide.fit_transform(
      ↪ df_lhud_1hora4m[columns_wide])
133 trans_lhud_test_wide = lhud_preprocessor_wide.transform(
      ↪ df_lhud_1hora_test[columns_wide])
134
135
136 ### Carrega Labels preditos do Snorkel
137 labels_g_pd = pd.read_hdf("labels_g_pd.hdf",'df')
138 labels_g_pd['anom'] = np.where((labels_g_pd[0]== 1),-1,1)
139 labels_g_pd[labels_g_pd['anom'] == -1].shape, labels_g_pd[
      ↪ labels_g_pd['anom'] == 1].shape
140
141 # Para RNA com entradas deep 13 e wide 4031
142
143 ds_deep_train = tf.data.Dataset.from_tensor_slices(
      ↪ trans_lhud_4m_deep)
144 ds_wide_train = tf.data.Dataset.from_tensor_slices(
      ↪ convert_sparse_matrix_to_sparse_tensor(trans_lhud_4m_wide))
145
146 ds_train = tf.data.Dataset.zip(({"deep_input": ds_deep_train, "
      ↪ wide_input": ds_wide_train}, ds_deep_train))
147
148 ds_deep_test = tf.data.Dataset.from_tensor_slices(
      ↪ trans_lhud_test_deep)
149 ds_wide_test = tf.data.Dataset.from_tensor_slices(
      ↪ convert_sparse_matrix_to_sparse_tensor(trans_lhud_test_wide))
150
151 ds_test = tf.data.Dataset.zip(({"deep_input": ds_deep_test, "
      ↪ wide_input": ds_wide_test}, ds_deep_test))
152
153 ds_train = ds_train.batch(1024).cache().prefetch(4)
154 ds_test = ds_test.batch(1024).cache().prefetch(4)
155
156 ds_train.element_spec, ds_test.element_spec
157
158 def objective(space):
159
160     #keras.backend.clear_session()
161
```

```
162     input_feat = keras.layers.Input(shape=[13], name="deep_input")
163     input_time_user = keras.layers.Input(shape=[4031], name="
            ↪ wide_input")
164
165     sae = keras.models.Sequential([
166
167         #keras.layers.BatchNormalization(),
168
169         keras.layers.Dense(space['first'],
170                            kernel_initializer=space['kernel_init'],
171                            activation=space['activ']),
172
173         keras.layers.BatchNormalization(),
174        # keras.layers.Activation(space['activ']),
175
176         #keras.layers.ActivityRegularization(space['l1_reg']),
177
178         keras.layers.Dense(space['second'],
179                            kernel_initializer=space['kernel_init'],
180                            activation=space['activ']),
181
182         keras.layers.BatchNormalization(),
183         #keras.layers.Activation(space['activ']),
184
185         #keras.layers.Dense(space['middle'],
186         # kernel_initializer=space['kernel_init']),
187
188         #keras.layers.BatchNormalization(),
189         #keras.layers.Activation(space['activ']),
190
191         keras.layers.Dense(space['second'],
192                            kernel_initializer=space['kernel_init'],
193                            activation=space['activ']),
194
195         keras.layers.BatchNormalization(),
196         #keras.layers.Activation(space['activ']),
197
198         keras.layers.Dense(space['first'],
199                            kernel_initializer=space['kernel_init'],
200                            activation=space['activ']),
201
202         keras.layers.BatchNormalization(),
```

```
203         #keras.layers.Activation(space['activ']),
204 
205     ])
206 
207     sae = sae(input_feat)
208 
209     concat = keras.layers.concatenate([input_time_user, sae])
210 
211     last = keras.models.Sequential()
212     last.add(keras.layers.Dense(space['last'],
213                         kernel_initializer=space['kernel_init']))#,
214                         #activation=space['activ']))
215     if space['last_batch_layer']:
216         last.add(keras.layers.BatchNormalization())
217     last.add(keras.layers.Activation("elu"))
218     if space['l1reg_layer']:
219         last.add(keras.layers.ActivityRegularization(space['l1_reg']))
220     if space['dropout_layer']:
221         last.add(keras.layers.Dropout(space['dropout']))
222     last = last(concat)
223 
224     output = keras.layers.Dense(13, name="output")(last)
225 
226     model = keras.models.Model(inputs=[input_feat, input_time_user],
        ↪ outputs=[output])
227 
228     model.compile(loss=space['loss_ob'], optimizer=space['optimizer'
        ↪ ], metrics=space['metrics'])
229 
230     history = model.fit(ds_train, epochs=space['epochs'], verbose=0)
231 
232     y_pred = model.predict(ds_test)
233 
234     anomalyScores, den_thres = dnn_tf_anomScores(
235     y_pred, trans_lhud_test_deep, df_lhud_1hora_test, 98.0)
236 
237     a,p,r,f,cm,auc_sc = benchmark_snorkel(labels_g_pd,anomalyScores)
238 
239     global index
240     scores_df.loc[index,:]=np.array([index,space,a,p,r,f,auc_sc,
        ↪ np.reshape(cm,(4))],dtype=object)
241 
```

```
    index=index+1

    print(space,r)

    return {'loss': -r, 'status': STATUS_OK, 'space': space,
          'model': model, 'f1_score': f,'auc_sc': auc_sc,
         'precision': p, 'recall': r, 'c_matrix': cm}


space ={'first': hp.choice("first", np.arange(7, 14, 1)),
       'second': hp.choice('second', np.arange(3, 8, 1)),
       'last' : hp.choice('last', np.arange(100,500,25)),
       'kernel_init' : hp.choice('kernel_init', ["he_normal"]),
       'activ' : hp.choice('activ', ["elu"]),
       'loss_ob' : hp.choice('loss_ob', ["mae", "mse", "Hubber"]),
       'optimizer' : hp.choice('optimizer', ["nadam","adam"]),
       'metrics' : hp.choice('metrics', [mean_absolute_error", "
          ↪ mean_squared_error"]),
       'epochs' : hp.choice('epochs', [10]),
       'l1_reg' : hp.choice('l1_reg', [0.00001, 0.00005, 0.0001,0
          ↪ .0005,0.001,0.005,0.01]),
       'dropout' : hp.choice('dropout', [0.0, 0.05, 0.1, 0.15, 0.2]),
       'last_batch_layer': hp.choice('last_batch_layer', [True, False
          ↪ ]),
       'l1reg_layer': hp.choice('l1reg_layer', [True, False]),
       'dropout_layer':hp.choice('dropout_layer', [True, False])
    }

scores_df = pd.DataFrame(columns=["Model","Params","Accuracy","
    ↪ Precision","Recall","F1-Score","ROC-AUC", "CM"])
index=0
#scores_df.info()

trials = Trials()
best = fmin(fn=objective,
         space=space,
         algo=tpe.suggest,
         max_evals=300,
         trials=trials)

## Melhor modelo pelas metricas Recall e ROC-AUC
scores_df.sort_values(['Recall','ROC-AUC'], ascending=[False,False])
```

IV.6 ALGORITMO DAS MÉTRICAS E CÁLCULO DE ERRO

```
import numpy as np
import pandas as pd
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix
# TensorFlow >=2.0 is required
import tensorflow as tf
from tensorflow import keras
assert tf.__version__ >= "2.0"

def benchmark_snorkel(gold_anom, pred_anom):
    # Anomalies preditas para o usuario

    # Calcula acuracia
    n_correct = sum(pred_anom.iloc[:,-1].values == gold_anom.iloc
        ↪ [:,-1].values)
    accuracy = n_correct / len(gold_anom)

    #Confusion Matrix
    cm = confusion_matrix(gold_anom.iloc[:,-1].values,
                    pred_anom.iloc[:,-1].values,labels = [1,-1])

    precision = precision_score(gold_anom.iloc[:,-1].values,
                    pred_anom.iloc[:,-1].values,
                    pos_label=-1)

    recall = recall_score(gold_anom.iloc[:,-1].values,
                    pred_anom.iloc[:,-1].values,
                    pos_label=-1)

    f1 = f1_score(gold_anom.iloc[:,-1].values,
                    pred_anom.iloc[:,-1].values,
                    pos_label=-1)


    fpr, tpr, thresholds = roc_curve(gold_anom.iloc[:,-1].values,
```

```
                    pred_anom.iloc[:,-2].values, pos_label=-1)

    auc_score = auc(fpr, tpr)

    return accuracy, precision, recall, f1, cm, auc_score

def convert_sparse_matrix_to_sparse_tensor(X):
    coo = X.tocoo()
    indices = np.mat([coo.row, coo.col]).transpose()
    return tf.SparseTensor(indices, coo.data, coo.shape)

def maeScores(originalDF, reducedDF):
    loss = np.mean(np.abs(np.array(originalDF)-np.array(reducedDF)),
        ↪ axis=1)
    loss = pd.Series(data=loss)
    loss = (loss-np.min(loss))/(np.max(loss)-np.min(loss))
    return loss


### Retorna os scores de erro e se e anomalia de acordo com
    ↪ threshold
def dnn_tf_anomScores(y_pred, trans_lhud_test, df_lhud_test, thres):
    # Se o modelo e RNN, precisa mudar a dimensionalidade de 3d para
        ↪ 2d
    if len(y_pred.shape) == 3:
        y_pred = np.reshape(y_pred,(y_pred.shape[0]*y_pred.shape[1],
            ↪ y_pred.shape[2]))
    #Calculo dos scores de anomalias e aplicacao do threshold para
        ↪ detecao de anomalias
    anomalyScores = maeScores(trans_lhud_test, y_pred)
    density_threshold = np.percentile(anomalyScores, thres)
    #print(density_threshold)
    anomalyScores = anomalyScores.to_frame().reset_index(drop=True)
    anomalyScores.set_index(df_lhud_test.index, inplace=True)
    anomalyScores['anom'] = np.where((anomalyScores[0] >
        ↪ density_threshold),-1,1)
    anomalyScores.columns = ['scores','anom']
    anomalyScores = pd.concat([df_lhud_test,anomalyScores], axis=1)
    return anomalyScores, density_threshold
```