UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

INAÊ SOARES DE FIGUEIREDO

SOBRE DETECÇÃO DE INTRUSÃO COM APRENDIZADO DE MÁQUINA: ESTUDO DA APLICAÇÃO DE ENSEMBLE LEARNING E CLUSTERIZAÇÃO NA MELHORIA DE DESEMPENHO

BAURU
Janeiro/2023

INAÊ SOARES DE FIGUEIREDO

SOBRE DETECÇÃO DE INTRUSÃO COM APRENDIZADO DE MÁQUINA: ESTUDO DA APLICAÇÃO DE ENSEMBLE LEARNING E CLUSTERIZAÇÃO NA MELHORIA DE DESEMPENHO

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.
Orientador: Prof. Dr. Kelton A. P. da Costa

BAURU
Janeiro/2023

F475d

Figueiredo, Inaê Soares de

Sobre Detecção de Intrusão com Aprendizado de Máquina: Estudo da Aplicação de Ensemble Learning e Clusterização na Melhoria de Desempenho / Inaê Soares de Figueiredo. -- Bauru, 2023

63 p. : tabs.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (Unesp), Faculdade de Ciências, Bauru

Orientador: Kelton Augusto Pontara da Costa

1. Sistemas de detecção de intrusão. 2. Aprendizado de máquina. 3. Redes de computadores. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca da Faculdade de Ciências, Bauru. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Inaê Soares de Figueiredo

Sobre Detecção de Intrusão com Aprendizado de Máquina: Estudo da Aplicação de Ensemble Learning e Clusterização na Melhoria de Desempenho

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Prof. Dr. Kelton A. P. da Costa
Orientador
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Profa. Dra. Simone Domingues Prado
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Prof. Dr. Aparecido Nilceu Marana
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Bauru, _____ de _____ de _____.

Dedico este trabalho aos meus pais, que sempre me apoiaram e incentivaram na busca por conhecimento.

Agradecimentos

Agradeço aos meus pais pelo constante incentivo aos meus estudos e por permitirem que eu me dedicasse a eles integralmente.

Agradeço ao meu orientador, Prf. Dr. Kelton da Costa por todo o auxílio e direcionamento, essenciais para o desenvolvimento e finalização deste trabalho.

Agradeço aos professores do Departamento de Computação da UNESP-Bauru por tudo que me ensinaram para que eu pudesse alcançar este momento de finalização da graduação.

"Não há nada como procurar, quando se quer encontrar alguma coisa. Você certamente encontrará algo, embora nem sempre seja aquilo que estava buscando."

J.R.R. Tolkien, O Hobbit

Resumo

Sistemas de detecção de intrusão em redes de computadores têm uma grande importância na manutenção da segurança de uma rede e são foco constante de pesquisas que buscam aprimorar IDSs já existentes ou desenvolver novos ainda melhores. Este trabalho apresenta um estudo dos modelos de aprendizado de máquina autoencoder e *Restricted Boltzmann Machine*, comprovadamente efetivos na detecção de intrusão. Os modelos são comparados e busca-se melhorá-los por meio da aplicação de técnicas de clusterização e *ensemble learning* (*majority-voting*). As métricas mostraram-se pouco afetadas pela aplicação das técnicas de melhoria individualmente, mas bons resultados foram obtidos ao combiná-las, alçando 84,71% de acurácia e 91,31% F1-*score* no *dataset* KDD99.

Palavras-chave: Sistema de detecção de intrusão, aprendizado de máquina, redes de computadores.

Abstract

Intrusion detection systems in computer networks are of great importance in maintaining the security of a network and are a constant focus of research that seeks to improve existing IDSs or develop even better ones. This work presents a study of the autoencoder and Restricted Boltzmann Machine machine learning models, proven effective in intrusion detection. The models are compared, and an attempt is made to improve them by applying clustering techniques and ensemble learning (majority voting). The metrics were little affected by the application of improvement techniques individually, but good results were obtained when combining them, reaching 84.71% accuracy and 91.31% F1-score on the KDD99 dataset.

Keywords: Intrusion detection systems, machine learning, computer networks.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

AE	*Autoencoder*
API	*Application Programming Interface*
CSE-CIC	*Communications Security Establishment & Canadian Institute of Cybersecurity Dataset*
DBN	*Deep Belief Networks*
DoS	*Denial of Service*
DDoS	*Distributed Denial of Service*
DL	*Deep Learning*
ISCX	*Information Security Center of Excellence*
IDS	*Intrusion Detection System*
IoT	*Internet of Things*
KDD	*Third International Knowledge Discovery and Data Mining Tools Competition Dataset*
MAE	*Mean Absolute Error*
ML	*Machone Learning*
MSE	*Mean Squared Error*
MV	*Majority Voting*
ND	*Novelty Detection*
RAM	*Random Access Memory*
RBM	*Restricted Boltzmann Machine*
RDP	*Windows Remote Desktop Protocol*
ReLU	*Rectified Linear Unit*
SVM	*Support Vector Machine*
USDL	*Unsupervised Deep Learning*

VAE *Variational Autoencoders*

VPN *Virtual Private Network*

Sumário

1 Introdução

Desde seu surgimento, o uso da internet não para de crescer. Segundo dados da União Internacional de Telecomunicação (do inglês *International Telecommunication Union* ITU), estima-se que em 2019 cerca de 51% da população mundial tinha acesso à internet (UNION, 2019a), chegando a 93% da população em determinados países, de acordo com dados da mesma fonte (UNION, 2019b).

Com a ocorrência da pandemia de COVID19, houve uma drástica mudança nos padrões de uso de internet mundialmente, com quase todos os setores de serviços migrando para o ambiente digital de forma rápida e às vezes sem o preparo necessário (ALASHHAB et al., 2021).

A migração do trabalho e do estudo presencial para atividades remotas se refletiu também no aumento de 200% no uso de VPNs, no início de 2020, uma tendência que não diminuiu com o passar do tempo (FELDMANN et al., 2021).

Além disso, refletindo a tendência dos últimos dois anos, o início de 2022 apresentou crescimento no número de violações de dados, sendo que 92% deles se deve a ataques cibernéticos. A tendência é que o número de ataques cresça ainda mais durante o ano (HENRIQUEZ, 2022).

Considerando a quantidade de pessoas conectadas atualmente por meio da internet e o propósito principal da rede como sendo o de transmitir dados entre pontos conectados à ela (SHAUKAT et al., 2020), nota-se uma certa urgência no investimento em mais e melhores sistemas de proteção que possam guardar estes dados contra ataques cibernéticos e também proteger as redes e computadores de invasores mal intencionados.

1.1 Justificativa

A detecção de intrusos, ou de comportamentos indesejados dentro de uma rede se dá por análise de assinatura do comportamento ou pela detecção de anomalias. A primeira forma se mostra como mais eficiente nos quesitos de velocidade e da emissão de menos alarmes falsos (REBELLO et al., 2016). No entanto, pesquisadores têm investido cada vez mais em buscar um método eficiente de detecção de anomalias com o auxílio de algoritmos de aprendizado de máquina, com o objetivo de solucionar a maior fraqueza do método de detecção por assinatura, que é sua incapacidade de detectar comportamentos malignos desconhecidos.

Nos últimos dois anos, diversas publicações científicas foram feitas sobre o assunto, propondo a aplicação de diversos métodos de aprendizado de máquina em busca do melhor sistema de detecção de anomalias.

Como exemplo, pode-se citar o artigo *Into the Unknown: Unsupervised Machine Learning Algorithms for Anomaly-Based Intrusion Detection*, no qual Zoppi, Ceccarelli e Bondavalli propõem uma abordagem com algoritmo de aprendizado não supervisionado (TOMMASO; CECCARELLI; BONDAVALLI, 2020), e o trabalho de Pu, Wang, Shen e Dong, em que aprofundam este tema utilizando uma abordagem de aprendizado não supervisionado baseado em *clusters*, ou agrupamentos (PU et al., 2021).

Um dos trabalhos recentes na área, publicado em 2020, traz uma abordagem também com aprendizado não supervisionado, aplicando *Auto Encoders* como método de detecção de anomalias. As simulações deste estudo (DAWOUD et al., 2020) produziram o resultado bastante positivo de aproximadamente 99% de acurácia na detecção de intrusões.

Na busca por resultados com ainda melhores acurácias, este projeto se propõe a contribuir com esta pesquisa ao analisar a aplicação de técnicas inteligentes que não foram utilizadas pelos pesquisadores.

Algumas técnicas, como os algoritmos de *ensemble learning* e clusterização são utilizados com o propósito de melhorar cada vez mais o desempenho dos modelos de aprendizado de máquina e podem ter efeitos positivos quando aplicados sobre o modelo proposto em *Internet of Things Intrusion Detection: A Deep Learning Approach*.

Este trabalho apresenta uma análise da utilização destas técnicas mencionadas no aprimoramento da detecção de intrusão em redes de computadores.

1.2 Objetivos

De forma geral, os objetivos deste trabalho são os de aprofundar os conhecimentos da pesquisadora nos temas de segurança de redes e aprendizado de máquina e o de buscar trazer uma contribuição positiva para as áreas estudadas, aperfeiçoando um método já existente de detecção de intrusão com *machine learning*.

1.2.1 Objetivos Específicos

Os objetivos específicos do projeto resumem-se em formas de alcançar estes objetivos gerais, especificados anteriormente:

- Documentar o estado-da-arte do conteúdo estudado por meio de uma revisão de literatura;
- Entender as tendências de utilização de métodos de *machine learning* na criação de sistemas de detecção de intrusão em redes;
- Estudar a metodologia para detecção de intrusão proposta por Dawoud et al. (2020), no artigo *Internet of Things Intrusion Detection: A Deep Learning Approach*;

- Estudar as técnicas de aprendizagem de máquina a serem implementadas, sendo estas os modelos de ensemble learning e clusterização, para definir a melhor a ser aplicada;
- Aprender e utilizar a ferramenta *TensorFlow*;
- Aplicar as técnicas da metodologia de Dawoud et al. (2020) em conjunto com as técnicas de melhoria de aprendizado;
- Treinar o sistema modificado sobre os conjuntos de dados selecionados;
- Observar os resultados dos treinamentos e modificar o sistema em busca de melhorias em desempenho e resultados;
- Observar os resultados obtidos e propor as considerações finais e as propostas de aplicação futuras do algoritmo desenvolvido.

2 Fundamentação Teórica

Neste capítulo são apresentadas as teorias por trás das técnicas aplicadas no desenvolvimento deste projeto de pesquisa, assim como são juntificados os usos de cada uma.

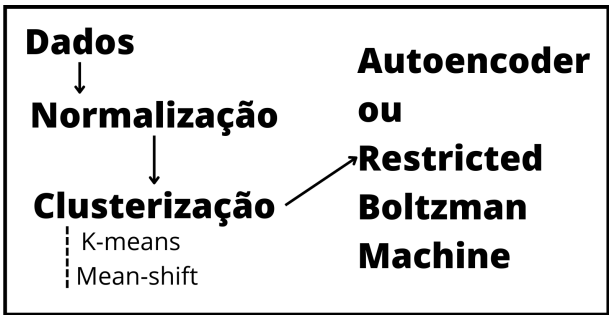
Ao final, os conjuntos de dados utilizados no treinamento dos algoritmos de aprendizado também são analizados, e suas caracteísticas são explicadas para facilitar o entendimento de etapas futuras de desenvolvimento.

2.1 Framework de Dawoud et al. (2020)

Internet of Things Intrusion Detection: A Deep Learning Approach (DAWOUD et al., 2020) é um dos artigos científicos que embasou e motivou esta pesquisa.

Ele descreve os experimentos dos autores em busca de desenvolver um *framework* de aprendizado de máquina para detecção de intrusão com um alto nível de acurácia (Figura 1).

Figura 1 – Representação gráfica do*Framework* proposto por Dawoud et al. (2020).



Fonte: A Autora

O foco principal do estudo é comparar, para o desenvolvimento de um *framework* com *deep learning*, o desempenho de *Restricted Boltzmann Machines* (RBMs) e autoencoders, ambos algoritmos de DL não supervisionado, considerados, pelos autores, os mais capazes de detectar ataques novos à rede.

A primeira etapa realizada pelos autores foi a construção da rede neural com autoencoder, com a definição dos pesos e *biases* para os *encoder* e *decoder*.

A rede foi então treinada com o uso de funções de ativação como *logits* e *ReLU*.

A próxima etapa foi comparar os dados originais com o *output* produzido na etapa anterior, usando então uma função de custo para calcular a perda e em seguida minimizando o custo.

O mesmo processo foi seguido no desenvolvimento da *RBM*, com a construção do modelo sem a especificação das funções ideais para os melhores resultados.

Para completar o *framework*, uma etapa anterior ao processamento com redes neurais foi adicionada, a de clusterização. Duas técnicas são comparadas, K-means e MeanShift.

Por último, os resultados foram comparados com auxílio de matrizes de confusão e diversas métricas foram analisadas, para cada um dos *frameworks* (com autoencoders e com RBMs), para indicar a combinação de técnicas e o número ideal de dados para treinamento e obtenção dos melhores resultados.

Durante o desenvolvimento do *framework* os pesquisadores testam diversas combinações de funções de ativação e de perda e técnicas de otimização. No entanto, ao descrever os resultados obtidos, os autores não especificam qual foi a combinação final que gerou os resultados obtidos, dificultando a reprodução dos modelos.

2.2 Aprendizado não supervisionado

Aprendizado de máquina não supervisionado é a aplicação de algoritmos de *machine learning* para analisar conjuntos de dados não rotulados, de forma a encontrar padrões ou agrupamentos dentro do conjunto sem que exista a necessidade de trabalho manual humano (IBM, 2020).

No entanto, algoritmos de *deep learning* não supervisionado (USDL - *Unsupervised Deep Learning*) não são eficientes como sistemas independentes de detecção de intrusão por que assumem que as anomalias ocorrem com menor frequência do que dados normais, podendo com isso gerar um alto número de detecções falsas quando analisa dados de ataques de *Distributed Denial of Service* (DDoS), por exemplo (DAWOUD et al., 2020).

Para os propósitos do *framework* desenvolvido, Dawoud e os outros pesquisadores trabalharam com dois modelos de USDL: *Restricted Boltzmann Machines* e autoencoders.

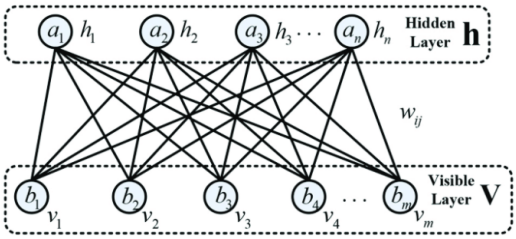
Restricted Boltzmann Machines (RBM) é um modelo probabilístico que utiliza uma camada escondida de unidades binárias para classificar uma camada visível de variáveis binárias ou reais, dependendo do modelo utilizado (Figura 2).

Bernoulli RBMs foram criadas para aplicações binárias e não respondem bem a dados com valores reais. Uma adaptação do modelo, a RBM Gaussiana, utiliza o desvio padrão associado às camadas visíveis para processar dados não binários (YAMASHITA et al., 2014)

Estudos de 2012 mostram que uma RBM pode ser utilizada como um modelo preditivo bastante poderoso e, embora o uso principal desta ferramenta seja como ativação para outros algoritmos, é possível criar um modelo completo a partir dela (LAROCHELLE et al., 2012).

Autoencoders por sua vez realizam um processo de compressão dos dados para dimensões

Figura 2 – Estrutura básica e uma RBM.

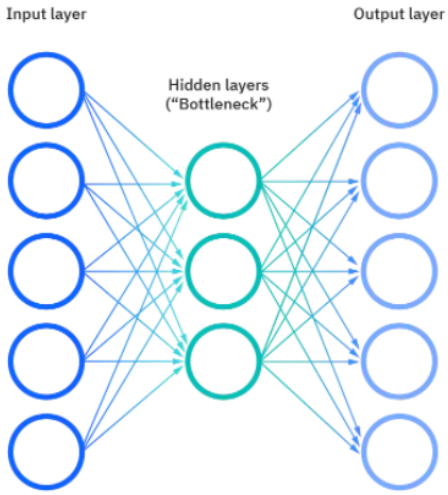


Fonte: Chu et al. (2018)

menores que a original, em camadas ocultas (etapa de *encoding*, ou codificação), e depois executam o proceso oposto (etapa de *decoding*, ou decodificação), gerando como saída uma representação reconstruída dos dados de entrada (IBM, 2020).

O algoritmo de autoencoder se utiliza de *backpropagation*, igualando o valor do dado de entrada ao de saída e tentando então aprender uma aproximação da função identidade para que o *output* seja semelhante ao *input* (STANFORD, 2022) (Figura 3).

Figura 3 – Representação visual do processo executado por um autoencoder.



Fonte: (IBM, 2020)

2.3 Funções de Ativação

Funções de ativação são utilizadas para obter o *output* de um nó de uma rede neural, mapeando os valores resultantes entre 0 e 1, ou -1 e 1, dependendo da função utilizada (SHARMA, 2017).

Funções de ativação podem ser lineares ou não-lineares, sendo o segundo grupo o mais utilizado, por facilitarem a adaptação ao conjunto de dados.

Em Dawoud et al. (2020), são utilizadas duas funções de ativação, uma como ativação da etapa de *forward pass* do autoencoder e outra logo em seguida, para reconstruir os dados. A primeira é especificada como sendo a função *logits*. A segunda não é informada, assim como a utilizada na versão do *framework* com RBM, mas como os autores mencionam testes com as funções *Sigmoid* e *ReLU*, elas serão testadas em busca do melhor resultado.

Assim como a *Logit*, a função Sigmoid retorna valores entre 0 e 1 (TEAM, 2017), sendo aplicada especialmente em situações em que se deseja prever uma probabilidade (SHARMA, 2017).

Rectified Linear Unit, ou *ReLU*, é uma função de ativação muito utilizada em *deep learning* (SHARMA, 2017). Ela retorna 0 para qualquer *input* negativo que receber, mas para qualquer valor positivo, ela o retorna. Por isso pode ser escrita como f(x) = max(0,x) (DANSBECKER, 2018).

2.4 Funções de Custo

Loss functions, ou funções de custo, são utilizadas para computar a diferença entre os valores de *input* e *output* de uma rede neural (DAWOUD et al., 2020).

Dawoud et al. utiliza a função *mean squared error* (MSE) para calcular a perda de dados, uma das funções mais utilizadas em regressões. Ela é uma boa função para se usar quando é importante destacar dados que se diferenciem muito da média obtida, já que oferece como predição ótima a média dos valores de erro e destaca valores muito altos ao aplicar a potenciação na diferença entre o esperado e o obtido (PELTARION, 2022).

A minimização do custo (erro) é a minimização da função de custo, igualando o valor da sua segunda diferencial a zero. A forma mais comum de fazer esse processo é utilizando o algoritmo do gradiente descendente (*gradient descent*) (RAWAT, 2021).

No *framework* proposto, os pesquisadores aplicaram o algoritmo de otimização chamado Adam, que é uma combinação de duas metodologias de gradiente descendente, acelerando o processo de convergência para o ponto mínimo ao utilizar a média do gradiente como base de cálculo e ao aplicar *root mean square propagation*, que utiliza a média móvel exponencial dos gradientes (PRAK, 2020).

2.5 Ensemble Learning

Algoritmos de *ensemble learning* são uma maneira de abordar problemas de aprendizado de máquina com o objetivo de obter um grande número de previsões para um mesmo problema.

Nestes tipo de implementação, diversos modelos de aprendizado são treinados para resolver um mesmo problema, construindo ao final do processo um conjunto de hipóteses que são combinadas e recebem pesos para que possa ser calculada uma previsão final levando todos os resultados em consideração (ZHOU, 2009).

Estudos já mostraram que métodos de *ensemble* frequentemente apresentam uma acurácia maior do que algoritmos que lidam com apenas uma previsão (DIETTERICH, 2002).

Existem diversos algoritmos de *ensemble* diferentes, que utilizam combinações distintas de modelos de previsão para melhorar o processo de aprendizado.

Voting, ou Votação, é um método de *ensemble* que se baseia na comparação das previsões realizadas por cada modelo que compõe o conjunto.

Este método analisa as previsões dos modelos e define uma como correta de acordo com a maioria dos votos (*Majority Voting*) ou de acordo com uma verificação de pesos atribuídos a cada previsão (*Soft Voting*). (BEYELER, 2019)

Como modelos de aprendizado podem aprender características diferentes dos dados e gerar previsões distintas, espera-se que, trabalhando com bons modelos e com a votação, seja possível evitar as previsões incorretas.

2.6 Normalização

Para que os dados estejam prontos para serem processados por um algoritmo de aprendizado de máquina é importante tratá-los de forma que fiquem em uma escala semelhante e permitam que a performance e estabilidade do algoritmo seja a melhor possível.

Existem algumas técnicas de normalização, e uma das mais comuns é a de escalar os dados para um mesmo intervalo padrão, permitindo que os limites superiores e inferiores de valor dos dados sejam previsíveis e tornando os dados mais uniformes (GOOGLE, 2021).

Esta é a técnica utilizada por Dawoud et al. (2020) para o desenvolvimento do *framework* que esta pesquisa busca reproduzir.

Os autores colocam todos os valores numéricos do *dataset* no intervalo entre 0 e 1 (DAWOUD et al., 2020).

A função *normalize* (*sklearn.preprocessing.normalize()*), disponibilizada na biblioteca SciKit-Learn, realiza a normalização elemento a elemento, de acordo com a norma do vetor de *input*. Isso pode ser feito por *sample* (linha, *axis*=1) ou *feature* (coluna, *axis*=0) (DEVELOPERS, 2022a). A saída da função é o vetor (ou matrix) de entrada completamente normalizado no eixo definido.

Como observado durante os experimentos, os valores normalizados são, em maioria, menores que 10^{-4}.

MinMaxScaler também é uma função de normalização, mas diferente da *normalize*, ela comprime os dados dentro de um intervalo definido (0-1, por padrão, mas pode ser modificado).

Os cálculos realizados pela função são apresentados nas Equações 2.1 e 2.2:

$$Xstd = (X - X.min(axis = 0))/(X.max(axis = 0) - X.min(axis = 0)) \tag{2.1}$$

$$Xscaled = Xstd * (max - min) + min \tag{2.2}$$

Esta função requer treinamento prévio com o conjunto de dados a ser normalizado para que possa computar os máximos e mínimos para posterior escalonamento das entradas (DEVELOPERS, 2022b).

Ambas as funções foram utilizadas no pré-processamento dos dados. Detalhes desta etapa do projeto encontram-se na Seção 4.5 deste documento.

2.7 Clusterização

O objetivo de métodos de clusterização é dividir os objetos analisados em grupos homogêneos. Tradicionalmente, estes métodos são não-supervisionados, ou seja, os grupos (*clusters*) são definidos a partir da observação das características dos dados (BAIR, 2013).

K-means é um algoritmo considerado simples e efetivo no tratamento de dados de tráfego de redes (SALO et al., 2019).

Como explica Salo et al. (2019), o K-means utiliza uma métrica de distância para determinar o distanciamento de cada instância de dado ao centro de cada grupo. Os dados são então atribuídos ao agrupamento mais próximo.

A inicialização do algoritmo é aleatória e requer uma pré-definição da quantidade de agrupamentos a serem estruturados.

Uma maneira considerada boa para definição do número n de agrupamentos a serem utilizados é a análise do coeficiente de silhueta (equação 2.3), que oferece uma análise de similaridade do dado dentro do grupo em que se encontra se comparado com os outros grupos.

$$S(i) = (b(i) - a(i))/max(a(i), b(i)) \tag{2.3}$$

Sendo S(i) a silhueta, a(i) a ditância média do dado aos outros do agrupamento em que se encontra, b(i) a distância do dado a todos os outros agrupamentos aos quais ele não pertence (BANERJI, 2022).

Por iniciar os agrupamentos com centroides aleatórios, é possível que o algoritmo acabe convergindo para um mínimo local. Por isso, considera-se positivo aplicá-lo mais de uma vez sobre o mesmo conjunto.

Outro problema apresentado pelo K-means pode aparecer quando dados possuem muitas dimensões, devido à maior dificuldade de interpretar distâncias em espaços de grande dimensionalidade.

A clusterização MeanShift é um algoritmo não supervisonado de clusterização baseado em centroides. A proposta dele é encontrar pontos de alta concentração de dados por meio da movimentação de janelas de seleção sobre o conjunto analisado (DEVELOPERS, 2020).

Como a movimentação é realizada repetidamente, este algoritmo apresenta um custo computacional alto. Além disso, a técnica também é afetada negativamente por dados de alta dimensionalidade.

A biblioteca SciKit-Learn [1] disponibiliza as implementações completa dos algoritmos K-means (utilizando distância euclidiana para definir os clustres e MeanShift).

2.8 Análise dos resultados

Uma análise compreensível dos resultados apresentados por um modelo de aprendizado de máquina precisa indicar com clareza as capacidades preditivas do modelo. Portanto, a utilização de métricas diversas permite uma visão mais ampla dos resultados obtidos.

Nas definições a seguir, valores *Negative* se referem a valores com *label* 0, ou seja, os dados normais de tráfego de rede. Já os valores *Positive* referem-se aos dados com *label* 1, os dados de tráfego anômalo da rede. Desta forma, métricas como precisão e recall mostram diretamente a capacidade dos modelos de detectar os dados anormais.

Os modelos desenvolvidos são analisados de acordo com os valores a seguir:

- *False Negative* (FN): indica quantos valores positivos o modelo previu incorretamente como negativos (BAJAJ, 2022);
- *False Positive* (FP): indica quantos valores negativos o modelo previu incorretamente como positivos (BAJAJ, 2022);
- *True Negative* (TN): indica quantos valores negativos o modelo previu corretamente como negativos (BAJAJ, 2022);
- *True Positive* (TP): indica quantos valores positivos o modelo previu corretamente como positivos (BAJAJ, 2022).

[1] Disponível em: <https://scikit-learn.org>. Acesso em 18 dez. 2022.

- Acurácia: número de exemplos categorizados corretamente (TP e TN) dividido pelo total de exemplos (WOOD, 2019);
- F1-*score*: média harmômica entre precisão e *recall* (BAJAJ, 2022);
- Precisão: a proporção de identificações corretas de valores positivos (TP) em relação a todos os valores previstos como positivos (TP e FP) (GOOGLE, 2020);
- *Recall*: proporção de previsões positivas (TP) em relação ao total de valores positivos verdadeiros (TP e FN) (BAJAJ, 2022);

3 Trabalhos Correlatos

Zavrak e İskefiyeli (2020) reconhecem a importância de aplicações capazes de detectar ataques desconhecidos (*zero-day attacks*) em redes e comparam o desempenho de autoencoders (AE), *variational* autoencoders (VAE) e *One-Class Support Vector Machines* na detecção de anomalias com *streams* de dados. O *dataset* utilizado para as comparações é o NSL-KDD, uma variante revisada do KDD99.

Os modelos de AE e VAE não são profundos, contando com apenas 5 camadas escondidas, e presentam bons resultados para detecção multiclasse.

Al-Qatf et al. (2018) também aplicam autoencoders na construção de IDSs, em conjunto com *Support Vector Machines*. Nesta abordagem, o AE é utilizado para gerar novas representações das características dos dados analisados, que são então alimentados na SVM para que sejam feitam as previsões.

A metodologia proposta mostra melhorias em comparação com outros modelos analisados pelos pesquisadores e com relação à utilização da SVM apenas. Os resultados são obtidos com o conjunto de dados KDD99.

Song, Hyun e Cheong (2021) tratam de temas semelhantes aos pesquisadores anteriores, mas com foco em arquiteturas da Internet das Coisas (IoT) que abrangem dispositivos com pouco poder computacional. A conclusão apresentada é a de que modelos simples de AE podem apresentar resultados positivos (F1-score de 0.895 no *dataset* NSL-KDD, por exemplo). Além disso, os experimentos demnstram que a quantidade de parâmetros treináveis e o tamanho da camada central do AE apresentam impacto nos resultados obtidos, enquanto que não foram observados efeitos resultantes das modificações da profundidade do modelo.

Ainda na área de IoT, Zhao et al. (2021) utilizam autoencoders no desenvolvimento de um modelo de detecção de intrusão em redes que exige pouco poder de processamento computacional e alcança boas métricas.

No modelo apresentado, o AE é combinado com uma *Logical Neural Network* e uma função própria de cálculo de perda.

Os conjuntos de dados utilizados, KDD99 e UNSW-NB15, são normalizados e passam por processo de redução de dimencionalidade utilizando *Principal Component Analysis*, e os resultados obtidos para cada *dataset* são de F1-score = 95,6% e F1-score = 98,5%, respectivamente.

Ieracitano et al. (2020) propõem um *framework* utilizando uma combinação de análise estatística de dados e AE para extrair características e melhorar o reconhecimento de dados anômalos em uma rede. O modelo, também avaliado no conjunto NSL-KDD, contém diversas

etapas de pré-processamento dos dados (análise de *outliers*, normalização, *one-hot encoding* e seleção de características) e apresenta bons resultados: F1-score acima de 70% para classificação binária e acima de 65% para classificação multiclasse.

Farahnakian e Heikkonen (2018) apresentam um modelo de AE profundo para classificação binária de anomalias. Conduzem experimentos para determinar as melhores dimensões de camadas ocultas para o modelo, que conta com etapa de pré-treinamento e duas etapas de *fine-tuning*, uma não supervisionada e outra supervisionada. Assim como os outros pesquisadores citados, trabalham também com o *dataset* KDD99. As etapas de pré-processamento dos dados incluem mapeamento dos valores discretos em valores numéricos, normalização e remoção de redundâncias.

Farahnakian e Heikkonen (2018) obtém acurácias de 96.53% e 94.71% para classificação binária e multiclasse, respectivamente.

Alom e Taha (2017) aplicam e comparam autoencoders e RBMs no sistema que porpõem. As duas técnicas são utilizadas para extração de características e redução de dimensionalidade. Sobre os dados redimensionados (3 *features*) é então aplicado um algoritmo de clusterização e o resultado é aplicado sobre um modelo de *Unsupervised Extreme Learning Machine* para a classificação. O modelo com AE atinge 91.86% de acurácia, enquanto que o modelo com RBM alcança 92.12%.

Trabalhando com *Restricted Boltzmann Machines*, Zhang e Chen (2017) e Alrawashdeh e Purdy (2016) as utilizam em conjunto com DBNs.

Zhang e Chen (2017) combinam o modelo com SVM também, criando dois algoritmos híbridos. A combinação de modelos *energy-based* apresenta melhor performance do que a SVM, o que os autores atribuem principalmente ao aprendizado não supervisionado realizado pela RBM.

Alrawashdeh e Purdy (2016) por sua vez, combinam as RBM e DBN com uma camada de regressão logística.

Nos modelos propostos, as principais características do *dataset* KDD99 são extraídas pela RBM e pasadas para o próximo algoritmo do conjunto, para que este possa realizar as previsões.

Seo, Park e Kim (2016) também utilizam a RBM como ferramenta para destacar características do *dataset* utilizado e não como o modelo de previsão. O estudo apresentado mostra que este tipo de aplicação melhora o desempenho do classificador utilizado.

De forma diferenciada, Aldwairi, Perera e Novotny (2018) desenvolvem um detector de intrusão utilizando uma RBM como classificador, avaliado no conjunto de dados *Information Security Center of Excellence (ISCX)*. Segundo afirmam os pesquisadores, o modelo sofre com grande facilidade de chegar no problema de *overfitting* e precisa que os parâmetros sejam muito

estudados e testados. Os valores de *learning rate* e número de iterações apresentaram maior efeito sobre os resultados ao serem modificados. Com acurácia de mais de 88%, o modelo proposto mostra-se competitivo se comparado a outros IDS atuais.

4 Metodologia

Neste capítulo são explicados em maior detalhe os materiais e ferramentas utilizados no desenvolvimento desta pesquisa. É também detalhado o processo seguido para o desenvolvimento dos testes aplicados para definir o melhor modelo de aprendizado.

4.1 Datasets

Para o treinamento da inteligência artificial para detecção de intrusão, são utilizados dois conjuntos de dados distintos, um deles fornecido pelo *Canadian Institute for Cybersecurity* e o outro disponibilizado pela *University of California*, Irvine (UCI).

O IDS2018[1], oferece informações não tratadas sobre sete tipos de ataques distintos: força bruta, *Heartbleed*, *Botnet*, DoS, DDoS, ataques *Web* e infiltrações na rede de dentro dela, e os dados estão divididos pelos dias em que foram feitas as capturas.

Ele é fruto de uma colaboração entre o *Communications Security Establishment* (CSE) e o *Canadian Institute for Cybersecurity* (CIC) e busca resolver a dificuldade dos pesquisadores da área de segurança de encontrar dados atualizados com os novos tipos de ataques que são criados ao longo dos anos.

O KDD-99[2], disponibilizado pela UCI em 1999, é o conjunto utilizado por Dawoud et al. (2020). Por este motivo, é coerente aplicar as técnicas estudadas sobre o mesmo agrupamento de dados em busca de melhorias perceptíveis.

Este conjunto de dados é relativamente antigo, atualizado pela última vez em 1999, mas se destaca por apresentar uma quantidade muito grande de dados para teste e treinamento. Estes dados são apresentados em versão classificada e não classificada, o que permite também o treinamento de redes tanto supervisionadas quanto não supervisionadas.

Além disso, o KDD-99 disponibiliza dados sobre 22 tipos distintos de ataques, incluindo 6 formas diferentes de DoS e 4 de *probe*.

A comparação entre os resultados obtidos com cada um dos conjuntos de dados, por serem distintos e de épocas relativamente distantes, pode mostrar também a adaptabilidade do sistema criado a novos tipos de ataques e às novidades nos tráfegos de redes.

[1] Disponível em: <https://www.unb.ca/cic/datasets/ids-2018.html> Acesso em: 4 ago. 2021

[2] Disponível em: <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html> Acesso em: 7 ago. 2021.

4.2 Recursos Utilizados

4.2.1 Ambientes de Desenvolvimento

Para facilidade de trabalho e por necessidade de um computador que oferecesse um bom desempenho na execução de algoritmos custosos e no processamento de grandes quantidades de dados, esta pesquisa foi desenvolvida parcialmente em uma máquina remota, com acesso virtual via RDP. Na máquina roda o sistema operacional Windows Server 2019 Datacenter e ela conta com 32gB de memória RAM e 2 *core* para processamento.

Para parte dos experimentos foi utilizada a plataforma Kaggle [3], que oferece máquinas virtuais nas quais é possível editar e executar *notebooks* Python e R. A plataforma também oferece a possibilidade de utilização gratuita, dentro de uma cota mensal, de GPUs e TPUs para acelerar o processamento.

4.2.2 Python

O Python [4] é uma linguagem de programação de uso gratuito e *open source*.

Entre suas diversas aplicações, destaca-se sua utilidade para desenvolvimento de algoritmos de aprendizado de máquina por melhorar a performance e a produtividade por meio do uso de bibliotecas fáceis de entender e de APIs de alto nível (RASCHKA; PATTERSON; NOLET, 2020).

Além disso, o Python conta com diversas bibliotecas e frameworks próprios para o trabalho com *machine learning*, como é o caso do *TensorFlow*, explicado na Seção 4.2.3.

4.2.3 TensorFlow

O TensorFlow é uma plataforma de código aberto para desenvolvimento de projetos de aprendizagem de máquina.

Ele oferece diversas ferramentas, bibliotecas e outros recursos para que desenvolvedores de sistemas de *machine learning* possam criar seus programas com facilidade (DEVELOPERS, 2021).

O TensorFlow é estrutrado sobre a API Keras [5], utilizando-se de suas funções e ambiente de desenvolvimento simples e bem documentado.

A plataforma é utilizada por Dawoud et al. (2020) na pesquisa que apoia este projeto, portanto, será utilizada também para a reprodução do ambiente de trabalho proposto.

[3] Disponível em: <https://www.kaggle.com/>. Acesso em 20 dez. 2022.
[4] Disponível em: <https://www.python.org/>. Acesso em 20 dez. 2022.
[5] Disponível em: <https://keras.io/> Acesso em 20 dez. 2022.

4.2.4 SciKit-Learn

SciKit-Learn [6] é um conjunto de ferramentas para análise de dados em linguagem Python. Dentre os módulos que ela disponibiliza está o módulo de normalização de dados *sklearn.normalize*. Este módulo inclui duas funções que apresentam o mesmo propósito de normalização mas que a realizam de formas distintas.

A SciKit-Learn oferece também um módulo de clusterização que inclui a implementação dos algoritmos K-means e MeanShift.

4.2.5 Learnergy

Learnergy (RODER; ROSA; PAPA, 2020) é uma biblioteca compatível com Python 3.6+, desenvolvida por pesquisadores da UNESP - Bauru.

Esta biblioteca permite a fácil implementação de modelos de ML *energy-based* por meio de funções pré-definidas. Ela oferece funções prontas para treinar e testar Restricted Boltzmann Machines e Deep Delief Networks (DBN) e funções que possibilitam estruturar os modelos de acordo com a preferência do desenvolvedor.

4.2.6 Orange *Data Mining*

O Orange [7] é uma ferramenta de código aberto para aprendizado de máquina e exploração de dados de forma visual, sem a necessidade de escrever códigos.

O *framework* funciona por meio de *widgets* diversos que permitem ao usuário visualizar e manipular dados, carregá-los em modelos de predição, avaliar o desempenho de classificações e algoritmos de regressão, realizar processos de aprendizado não supervisionado e de mineração de informação em textos.

Para utilizar as funcionalidades, basta adicionar os *widgets* desejados ao *canvas* e abri-lo para configurar suas propriedades.

Para este projeto de pesquisa, o ponto atrativo do Orange é a possibilidade de visualização organizada dos dados e das etapas de processamento dos algoritmos a serem implementados.

Um *widget* importante que o Orange oferece é o que permite carregar trechos de código em Python. Com isso é possível executar na ferramenta os modelos reproduzidos na pesquisa e aproveitar as facilidades que ela oferece para a análise dos dados e resultados obtidos.

[6] Disponível em: <scikit-learn.org/> Acesso em: 9 nov. 2022.
[7] Disponível em: <https://orangedatamining.com/> Acesso em: 17 junh. 2022.

4.3 Etapas do Desenvolvimento

Com o objetivo de desenvolver um sistema robusto de detecção de intrusão em redes de computadores e tomando como base os resultados obtidos no paper *Internet of Things Intrusion Detection: A Deep Learning Approach* (DAWOUD et al., 2020), a proposta desta pesquisa é de aplicação de algoritmos de aprendizado não supervisionado (autoencoders e Restricted Boltzmann Machines) para estudar a possibilidade de aplicá-los em IDSs.

Em Dawoud et al. (2020), os pesquisadores apresentam resultados positivos de aproximadamente 99% de acurácia aplicando AE e RBM para detecção de intrusão. Buscando resultados ainda melhores considerou-se para esta pesquisa a aplicação de técnicas de melhoramento de aprendizado sobre os modelos de AE e RBM, como *ensemble learning* e clusterização.

O projeto busca também verificar se o bom funcionamento dos algoritmos se mantém mesmo com a utilização de conjuntos diferentes de dados e para isso utiliza e compara os desempenhos dos programas ao processarem dois *datasets* distintos: KDD99 e IDS18.

A primeira etapa realizada foi buscar reproduzir o ambiente criado por DAWOUD et al. (2020), seguindo as etapas ilustradas na Figura 1 e listadas a seguir, de acordo com o artigo publicado em 2020:

- Normalização dos dados, convertendo todo valor literal em um correspondente numérico e depois escalando os dados de forma que todos se encontrem em um intervalo fechado [0,1];
- Construção do autoencoder, utilizando a biblioteca para *Deep Learning* oferecida pelo Tensor Flow;
- Construção da Restricted Boltzmann Machine, utilizando a biblioteca para *Deep Learning* oferecida pelo Tensor Flow;
- Teste do desempenho de diferentes funções de ativação, perda e otimização para encontrar a que mostre melhores resultados e configurar os parâmetros de cálculo;
- Aplicação de clusterização K-means sobre os dados e executar os modelos com os *datasets* modificados;
- Aplicação de clusterização MeanShift sobre os dados e executar os modelos com os *datasets* modificados;
- Comparação dos resultados obtidos por cada modelo.

Após a recriação do ambiente, como proposto, foi verificado o desempenho do *framework* no processamento dos *subconjuntos* dos dois conjuntos de dados elencados.

Em seguida, foi aplicada a técnica de *ensemble learning Majority Voting*, tanto sem quanto em conjunto com as técnicas de clusterização, e os resultados obtidos foram comparados com os dos modelos anteriores.

5 Experimentação e Análise dos Resultados

Neste capítulo são detalhados e explicados os experimentos realizados durante o desenvolvimento desta pesquisa. Os resultados obtidos serão discutidos na Seção 5.6.

5.1 Pre-processamento de dados

Foram selecionados dois conjuntos de dados para o treinamento dos modelos, o CSE-CIC-IDS2018 e o KDD-99.

Cada um destes conjuntos apresenta características bastante distintas dos dados, mas são semelhantes no que se refere ao tipo dos dados: ambos contam com valores numéricos contínuos e discretos e com valores literais.

Um exemplo de registro do KDD-99 é como se vê a seguir:

```
0,tcp,http,SF,210,7061,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,8,8,0.00,
0.00,0.00,0.00, 1.00,0.00,0.00,8,255,1.00,0.00,0.12,0.03,0.00,
0.00,0.00,0.00,normal
```

Um exemplo de registro do CSE-CIC-IDS2018 é como se vê a seguir:

```
53,17,28/02/2018,04:30:20,13924,1,1,37,186,37,37,37,0,186,186,
186,0,16015.51278,143.636886,13924,0,13924,13924,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,8,8,71.81844298, 71.81844298,37,186,86.66666667,
86.02519011,7400.333333,0,0,0,0,0,0,0,0,1,130,37,186,0,0,0,0,0,
0,1,37,1,186,-1,-1,0,8,0,0,0,0,0,0,0,0,benign
```

Seguindo o procedimento proposto no *framework* de Dawoud et al. (2020), os valores literais foram substituídos por valores numéricos, utilizando a técnica de dicionário.

O dataset CSE-CIC-IDS2018 apresenta as *features* de data e *timestamp*, as quais foram removidas para a normalização. Além disso, mais uma etapa foi necessária para o tratamento deste conjunto: algumas entradas de dados apresentavam valores *infinity* e Nan nos campos *Flow Bytes* e *Flow Pkts*. As entradas com valores Nan foram removidas do conjunto (RAVIKUMAR, 2021), enquanto os campos com *Infinity* foram substituídos pelo maior valor da coluna+1 (LEEVY; KHOSHGOFTAAR, 2020).

Nas etapas seguintes, alguns procedimentos divergem levemente do descrito no artigo a ser reproduzido. Estas divergências buscam uma maior facilidade de processamento com a ferramenta *TensorFlow* utilizada.

Além disso, como não é especificada a técnica de normalização utilizada e apenas uma entrada de dados é mostrada, decidiu-se por utilizar mais de uma técnica e escolher a que apresentasse melhores resultados para a aplicação final.

No *framework* de Dawoud et al. (2020), os *Lables* de cada instância de dado foram trocadas para 10 (tráfego normal) ou 01 (tráfego anormal). Nesta pesquisa, eles foram trocados por valores 0 (tráfego normal) e 1 (tráfego anormal), para maior agilidade na programação dos modelos.

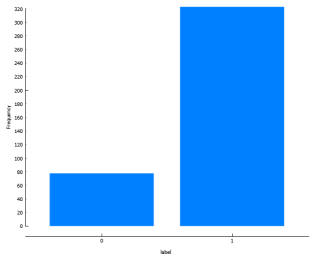
Os dados foram então normalizados utilizando duas técnicas diferentes, as funções *minMaxScaler()* e *normalize()*.

Nos experimentos conduzidos por Dawoud et al. (2020), os testes são realizados com 3 diferentes *subsets* do KDD-99, um com 400 instâncias de dados, um com 800 e outro com 1.300.

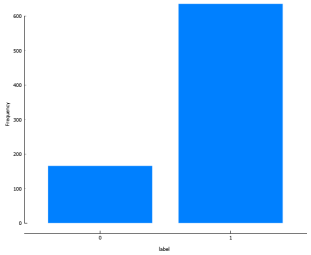
Foram então separados 3 conjuntos de cada *dataset*, com as quantidades de dados mencionadas. A separação foi feita de forma aleatória.

Para verificar a aplicabilidade do projeto em quantidades maiores de dados, um quarto conjunto com 5.000 dados foi estruturado da mesma forma, para cada *dataset*.

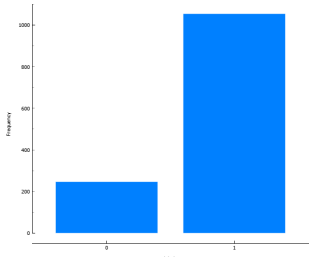
Figura 4 – Distribuição das amostras dos subconjuntos do *dataset* KDD99 de acordo com *labels*.



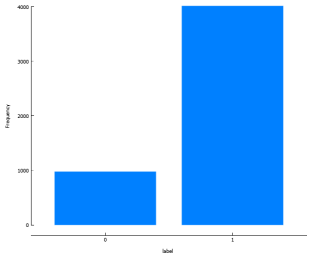
(a) *Subset* de 400 *samples*



(b) *Subset* de 800 *samples*



(c) *Subset* de 1300 *samples*



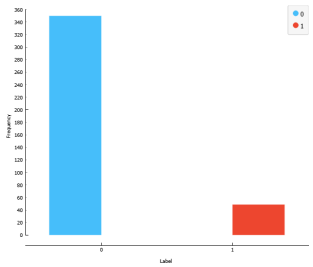
(d) *Subset* de 5000 *samples*

Fonte: A Autora

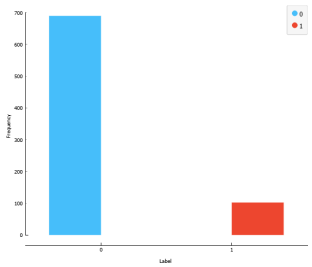
Os dados de cada subconjunto, definido aleatoriamente, estão distribuídos entre *labels* normal (0) e anormal (1) como mostram os gráficos nas Figuras 4 e 5.

No total, após os processos de normalização com as funções *Normalize()* e *MinMax()*, obteve-se 16 conjuntos:

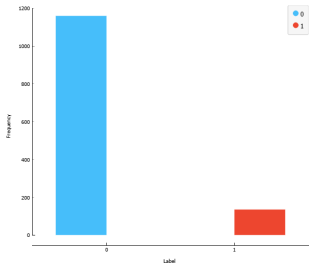
Figura 5 – Distribuição das amostras dos subconjuntos do *dataset* CSE-CIC de acordo com *labels*.



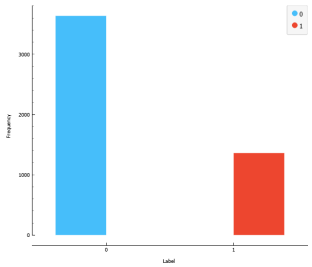
(a) *Subset* de 400 *samples*



(b) *Subset* de 800 *samples*



(c) *Subset* de 1300 *samples*



(d) *Subset* de 5000 *samples*

Fonte: A Autora

1. KDD-400-MinMaxScaler;
2. KDD-800-MinMaxScaler;
3. KDD-1300-MinMaxScaler;
4. KDD-5000-MinMaxScaler;
5. CICAWS-400-MinMaxScaler;
6. CICAWS-800-MinMaxScaler;
7. CICAWS-1300-MinMaxScaler;
8. CICAWS-5000-MinMaxScaler;
9. KDD-400-Normalize;
10. KDD-800-Normalize;
11. KDD-1300-Normalize;
12. KDD-5000-Normalize;
13. CICAWS-400-Normalize;
14. CICAWS-800-Normalize;
15. CICAWS-1300-Normalize;

16. CICAWS-5000-Normalize.

Todos estes conjuntos foram utilizados nos testes de cada modelo reproduzido. Os resultados obtidos com o autoencoder são descritos na Seção 5.8.

5.2 Autoencoder

O autoencoder utilizado foi desenvolvido com base no modelo para detecção de anomalia proposto na plataforma do Tensor Flow [1].

O AE é linear e composto por camadas densas com ativação *ReLU*. A camadas densas recebem como parametro seu tamanho e a função de ativação.

O modelo é dividido em duas partes, o codificador e o decodificador.

Três diferentes profundidades foram testadas para o modelo: 6 camadas (3 do codificador e 3 do decodificador), 10 camadas (5 do codificador e 5 do decodificador) e 20 (10 do codificador e 10 do decodificador).

Testes foram realizados com apenas a ativação *ReLU* em todas as camadas e também combinando-a com a aplicação da função de ativação *Sigmoid* na última camada de previsão (a camada final do *decoder*).

Na declaração do modelo, são definidos os parâmetros otimizador e função de perda.

Os otimizadores analisados foram o Adam e o Stochastic Gradient Descent. As funções de perda comparadas foram MAE, MSE e categorical cross entropy.

Batches de tamanhos 100 e 64 foram testados, juntamente com *epochs* de 100 e 200 iterações.

As combinações de funções e parâmetros geraram 72 modelos de teste. Para obter resultados mais representativos, cada modelo foi executado um total de 10 vezes, com cada um dos 16 *subsets* de dados.

Os resultados apresentados de cada teste são uma média dos resultados obtidos nas 10 execuções.

Os resultados apresentados por cada modelo são comparados na Seção 5.8 Análise de Resultados, e as combinações com melhores desempenhos são mostrados.

5.3 Restricted Boltzmann Machine

Dois modelos foram testados para o desenvolvimento do modelo de RBM: Bernoulli RBM e Gaussian RBM.

[1] https://www.tensorflow.org/tutorials/generative/autoencoder

A Bernoulli RBM foi utilizada da biblioteca SciKit-Learn.

Ela recebe como parâmetros o número de camadas ocultas, o *learning rate*, o tamanho dos batches e o número de epochs.

Este modelo foi descartado logo nos primeiros testes por aparentar não conseguir aprender com os dados.

Após treinamento, com quantidades variadas de dados e diferentes parâmetros, em todas as instâncias as previsões eram sempre de que todas as entradas de dados eram positivas.

Isso resultou em modelos de alta acurácia, mas incapazes de prever corretamente, como pode ser observado ao gerar a matriz de confusão e comparar valores de *true negative*, *true positive*, *false negative* e *false positive*.

Nesta análise, todos os dados se encontravam distribuídos nos valores de *true positive* e *false positive*.

A análise feita foi de que, como Bernoulli RBMs são ideais para trabalhar com dados em formato binário, o modelo não era o melhor para tratar os *datasets* utilizados.

Outra forma de implementar a RBM foi encontrada na biblioteca Learnergy.

Desta vez o modelo escolhido foi uma Gaussian RBM.

Como parâmetros, foram testadas 200 e 400 camadas ocultas, 10 e 50 *batches* de treinamento, 100 e 50 *epochs* e *learning rates* de 0.01 e 0.004.

As combinações de funções e parâmetros geraram 16 modelos de teste. Para obter resultados mais representativos, cada modelo foi executado um total de 10 vezes, com cada um dos 16 *subsets* de dados.

Os resultados apresentados de cada teste são uma média dos resultados obtidos nas 10 execuções.

Mesmo com estes modelos distintos, a Gaussian RBM apresentou o mesmo problema da Bernoulli RBM, não parecendo ser capaz de aprender com os dados e realizando previsões iguais para qualquer dado alimentado no modelo para teste.

Por restrições de tempo, não foi possível estudar e compreender o problema encontrado mais a fundo, e o modelo foi descartado como opção para o melhor modelo, não sendo aplicado com as técnicas de melhorias para comparação.

5.4 Clusterização - Kmeans

Uma metodologia de clusterização utilizada nesta pesquisa é a K-means.

O número de agrupamentos para que o algoritmo K-means possa realizar a classificação deve ser informado pelo programador. Para definir o valor ideal, a ferramenta Orange Data

Mining se mostrou bastante útil.

Aplicando o *widget* K-means sobre os dados, a ferramenta calcula o coeficiente de silhueta para o *dataset*, considerando valores diferentes para o número de agrupamentos.

Tabela 1 – Relação de coeficientes de silhueta e número ideal de agrupamentos.

Dataset	Maior Silhueta	Agrupamentos
KDD99 400	0,857	15
KDD99 800	0,857	26
KDD99 1300	0,866	21
KDD99 5000	0,627	2
CSE-CIC 400	0,956	6
CSE-CIC800	0,936	28
CSE-CIC1300	0,925	4
CSE-CIC 5000	0,916	29

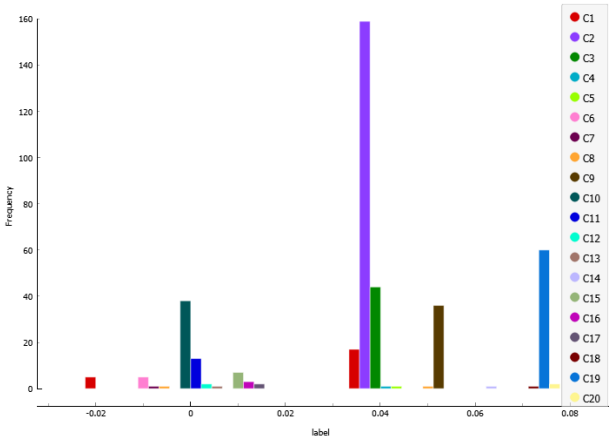
Fonte: A Autora

Para cada subconjunto de dados o valor ideal de agrupamentos sugerido foi diferente (Tabela 1), portanto foi necessário escolher uma quantidade de agrupamentos intermediária que, mesmo não sendo ideal, funcionasse bem para todos. O valor escolhido foi de 20 grupos.

Esta nova análise também foi realizada por meio do Orange, gerando uma visualização gráfica dos resultados da clusterização de acordo com os grupos e *labels*.

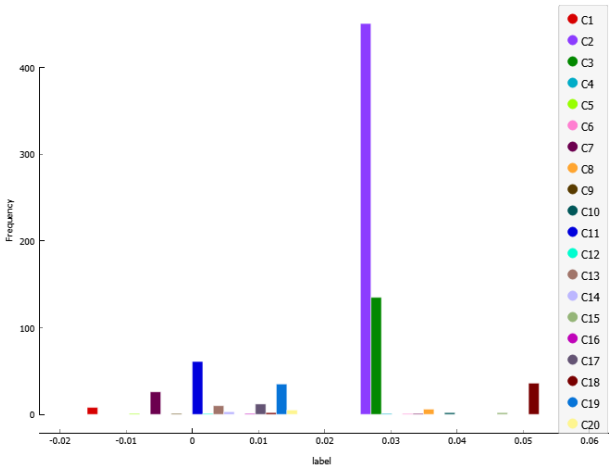
Com o valor escolhido pode-se observar que nos *subsets* do KDD99 os algoritmos de clusterização são capazes de separar de forma satisfatória os valores pertencentes a cada *label*, com mínima ocorrência de valores normais e anormais no mesmo grupo (Figuras 06 a 09).

Figura 6 – Distribuição em agrupamentos dos dados do *subset* de 400 *samples* do dataset KDD.



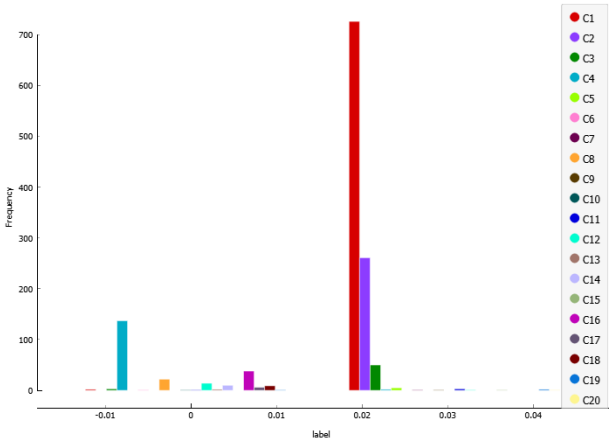
Fonte: A Autora

Figura 7 – Distribuição em agrupamentos dos dados do *subset* de 800 *samples* do dataset KDD.



Fonte: A Autora

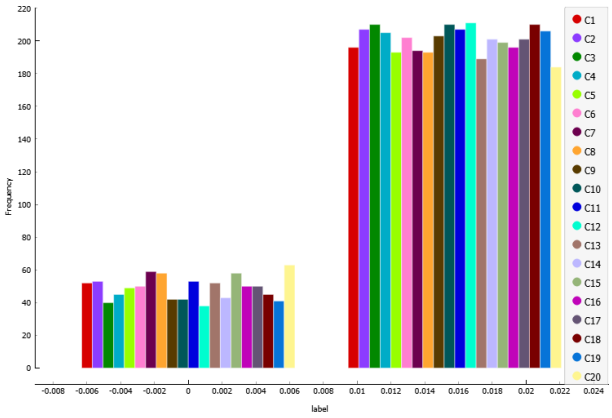
Figura 8 – Distribuição em agrupamentos dos dados do *subset* de 1300 *samples* do dataset KDD.



Fonte: A Autora

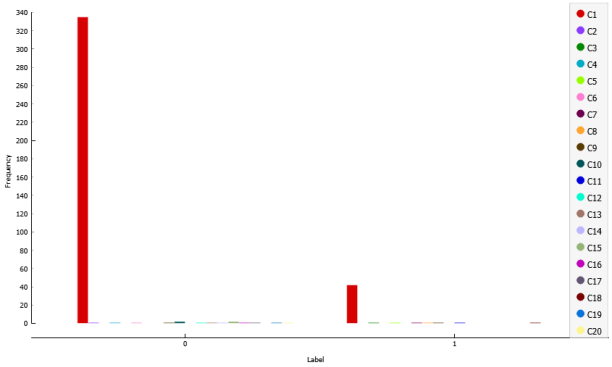
O mesmo não acontece com os dados do CSE-CIC (Figuras 10 a 13). No entanto, o mesmo acontece quando o número de agrupamentos delecionado é o sugerido pelo coeficiente de silhueta (Figuras 14 a 17). Acredita-se que isso se deva ao fato de os dados apresentarem características muito semelhantes, mesmo pertencendo a categorias distintas, de forma que não é possível para o algoritmo separá-los em grupos diferentes apenas por meio da análise de *features*.

Figura 9 – Distribuição em agrupamentos dos dados do *subset* de 5000 *samples* do dataset KDD.



Fonte: A Autora

Figura 10 – Distribuição em agrupamentos dos dados do *subset* de 400 *samples* do dataset CSE-CIC.



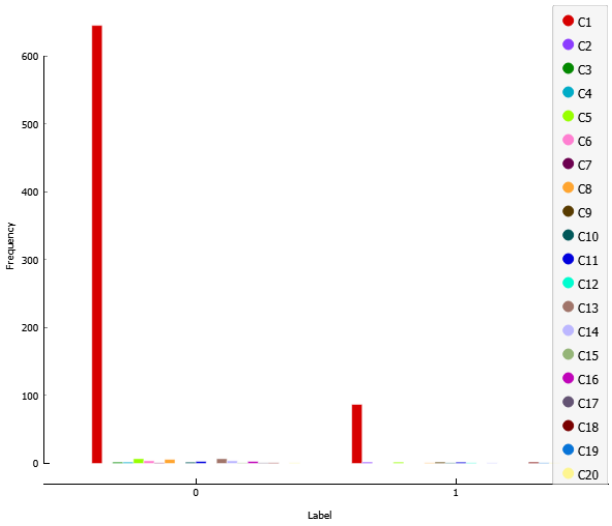
Fonte: A Autora

Como não foi possível encontrar um valor que funcionasse corretamente para o CSE-CIC por meio destes experimentos, considerou-se suficiente o funcionamento com o KDD99.

Após o processamento dos dados com o algoritmo K-means, os valores dos agrupamentos aos quais cada dado pertence foram adicionados aos *subsets* originais como uma nova *feature*, nomeada 'cluster'.

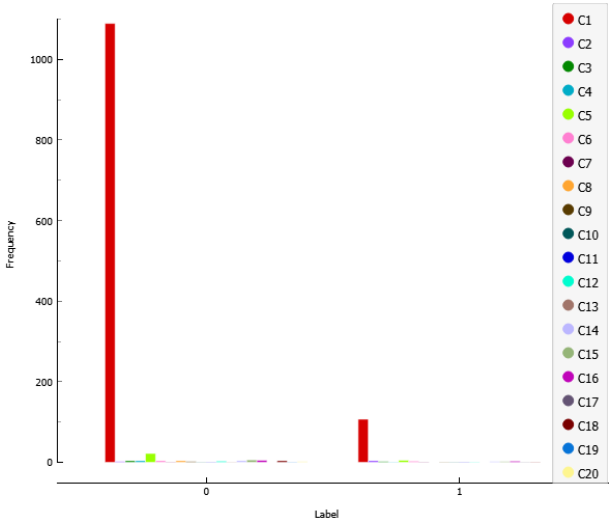
Estes novos conjuntos de dados foram também utilizados para treinar e testar os modelos de previsão. Os resultados e comparações entre os treinamentos com e sem a clusterização são explicados na Seção 5.6.

Figura 11 – Distribuição em agrupamentos dos dados do *subset* de 800 *samples* do dataset CSE-CIC.



Fonte: A Autora

Figura 12 – Distribuição em agrupamentos dos dados do *subset* de 1300 *samples* do dataset CSE-CIC.



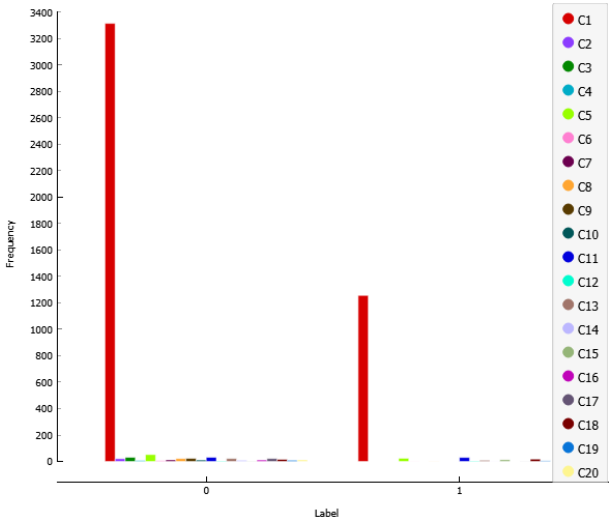
Fonte: A Autora

5.5 Ensemble Learning - Majority Voting

A estrutura de *Ensemble Learning* utilizada, *Majority Voting*, foi estruturada com três modelos de decisão de forma a evitar empates na votação.

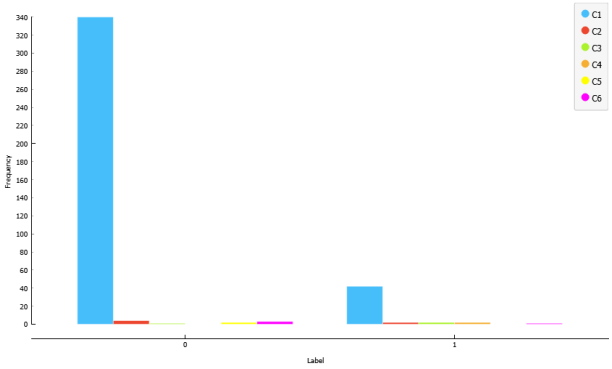
Cada modelo foi treinado e testado separadamente nos mesmos subconjuntos de dados e as previsões obtidas foram analisadas de forma que aquela emitida pela maioria fosse considerada correta.

Figura 13 – Distribuição em agrupamentos dos dados do *subset* de 5000 *samples* do dataset CSE-CIC.



Fonte: A Autora

Figura 14 – Melhor Silhueta: distribuição em agrupamentos dos dados do *subset* de 400 *samples* do dataset CSE-CIC.



Fonte: A Autora

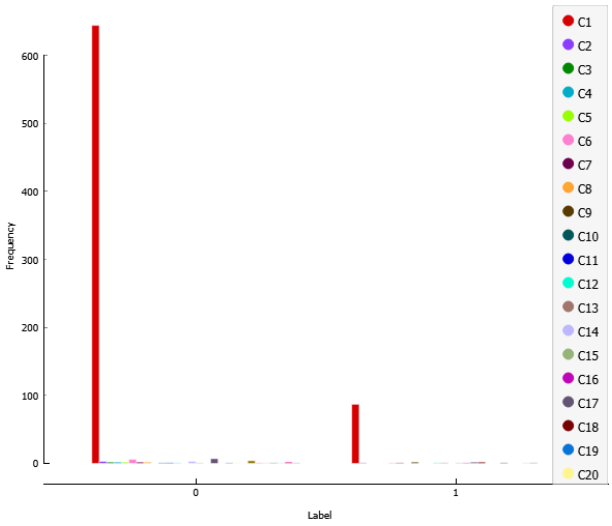
Os resultados e comparações entre os treinamentos com e sem *Majority Voting* são explicados na Seção 5.6.

5.6 Análise dos Resultados

Nesta Seção são analisados e comparados os resultados obtidos com os experimentos conduzidos nesta pesquisa.

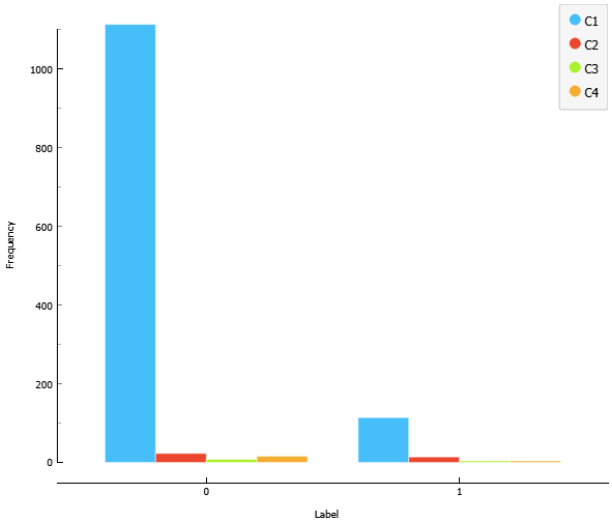
Os modelos que foram testados com muitas modificações de variáveis, apenas os 2 melhores resultados são explicados com mais profundidade, de forma a não estender este documento com muitas tabelas com resultados pouco relevantes para a proposta apresentada.

Figura 15 – Melhor Silhueta: distribuição em agrupamentos dos dados do *subset* de 800 *samples* do dataset CSE-CIC.



Fonte: A Autora

Figura 16 – Melhor Silhueta: distribuição em agrupamentos dos dados do *subset* de 1300 *samples* do dataset CSE-CIC.



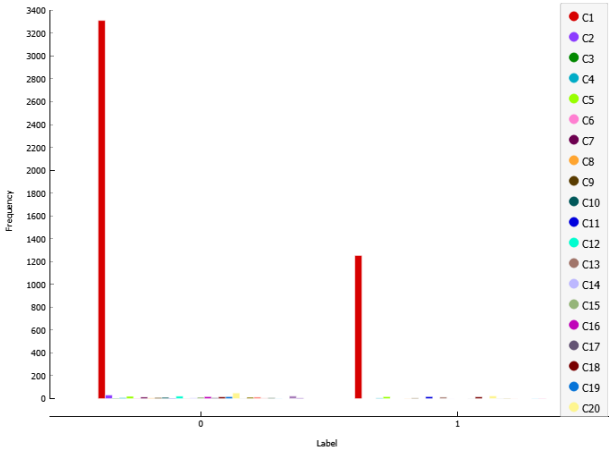
Fonte: A Autora

As subseções deste capítulo são divididas da seguinte maneira:

Seção 5.6.1: Modelos independentes apresenta análise dos modelos de autoencoder e RBM sem adição de técnicas de melhoramento, comparando resultados obtidos com as diferentes funções de normalização;

5.6.2: Clusterização - K-means apresenta os resultados obtidos após a aplicação da técnica de clusterização K-means sobre o modelo selecionado;

Figura 17 – Melhor Silhueta: distribuição em agrupamentos dos dados do *subset* de 5000 *samples* do dataset CSE-CIC.



Fonte: A Autora

5.6.3: Clusterização - MeanShift apresenta a análise da aplicação da clusterização MeanShift;

5.6.4: Majority Voting compara os resultados que resultam da aplicação de *majority voting* sobre o modelo simples e com as combinações de clusterização.

5.6.1 Modelos independentes

Mesmo apresentando estruturas diferentes, todos os modelos testados apresentaram um comportamento semelhante em relação aos dados: as melhores performances registradas foram resultados dos treinamentos com a utilização dos *subsets* do KDD99.

Inicialmente, considerou-se que a distribuição diferente de *labels* nos conjuntos poderia ser uma das causas deste comportamento, mas testes com *subsets* balanceados de forma diferente mostraram que uma maior quantidade de *labels* 1 nos treinamentos com o CSE-CIC resultaram em métricas ainda piores.

Desta forma, a escolha do melhor modelo baseou-se nos resultados obtidos com o KDD99 pois, com as combinações utilizadas na estruturação dos algoritmos, as performances do CSE-CIC não se destacaram.

A RBM mostrou-se um método inadequado para os propósitos desta pesquisa.

Todas as combinações testadas do modelo apresentaram bons resultados para os *subsets* do KDD99 (Tabela 2), e resultados ruins para os *subsets* do CSE-CIC (Tabela 3).

Tabela 2 – Melhor modelo RBM KDD99 - 0.004 LR.

Subset	Acurácia	Precisão	Recall	F1-score
KDD99 - 400	0,826	1	0,826	0,905
KDD99 - 800	0,783	1	0,783	0,878
KDD99 - 1300	0,835	1	0,835	0,910
KDD99 - 5000	0,801	1	0,801	0,89

Fonte: A Autora

Tabela 3 – Melhor modelo RBM CSE-CIC - 0.004 LR.

Subset	Acurácia	Precisão	Recall	F1-score
CSE-CIC - 400	0,175	1	0,175	0,298
CSE-CIC - 800	0,182	1	0,182	0,308
CSE-CIC - 1300	0,127	1	0,127	0,225
CSE-CIC - 5000	0,277	1	0,277	0,434

Fonte: A Autora

Buscando compreender melhor o motivo destas discrepâncias, foram criadas as matrizes de confusão dos modelos, e observou-se todos os dados estavam sendo previstos como anomalias.

Figura 18 – Matrizes de confusão: Modelo RBM KDD99, 0.004 LR.

		Previsto	
		Anormal	Normal
Real	Anormal	67	0
	Normal	14	0

(a) KDD99 - 400

		Previsto	
		Anormal	Normal
Real	Anormal	126	0
	Normal	35	0

(b) KDD99 - 800

		Previsto	
		Anormal	Normal
Real	Anormal	218	0
	Normal	43	0

(c) KDD99 - 1300

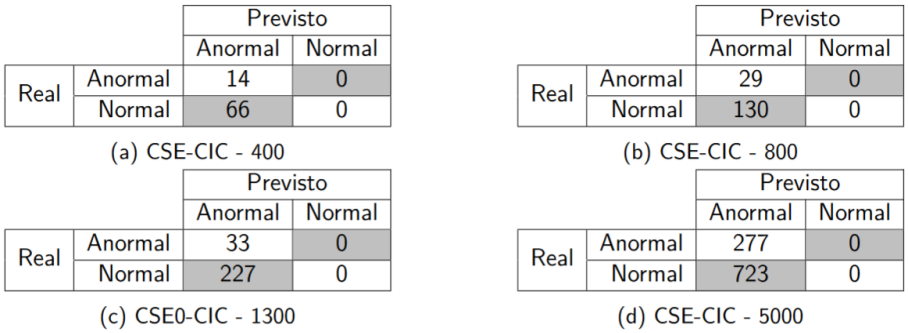
		Previsto	
		Anormal	Normal
Real	Anormal	801	0
	Normal	199	0

(d) KDD99 - 5000

Fonte: A Autora

Pode-se observar nas Figuras 18 e 19, que mostram as matrizes de confusão de uma das rodadas de treinamento da RBM que, independente da quantidade de dados utilizados nos treinamentos e testes, os modelos aparentam não aprender informações suficientes sobre os dados para fazer previsões bem informadas.

Figura 19 – Matrizes de confusão: Modelo RBM CSE-CIC, 0.004 LR.



		Previsto	
		Anormal	Normal
Real	Anormal	14	0
	Normal	66	0

(a) CSE-CIC - 400

		Previsto	
		Anormal	Normal
Real	Anormal	29	0
	Normal	130	0

(b) CSE-CIC - 800

		Previsto	
		Anormal	Normal
Real	Anormal	33	0
	Normal	227	0

(c) CSE0-CIC - 1300

		Previsto	
		Anormal	Normal
Real	Anormal	277	0
	Normal	723	0

(d) CSE-CIC - 5000

Fonte: A Autora

Desta forma, descartou-se a possibilidade de utilização de RBMs nesta pesquisa.

Os modelos de autoencoder desenvolvidos apresentaram boas métricas desde o início, embora não tenham alcançado a qualidade dos experimentos de Dawoud et al. (2020).

Tabela 4 – Melhor modelo AE KDD99.

Subset	Acurácia	Precisão	Recall	F1-score
KDD99 - 400	0,80617284	0,984994591	0,815528824	0,897959184
KDD99 - 800	0,78136646	0,975599338	0,794031402	0,875291817
KDD99 - 1300	0,808429119	0,981985043	0,818402739	0,892680404
KDD00 - 5000	0,8057	0,997653632	0,806964983	0,892182028

Fonte: A Autora

O AE que apresentou melhores resultados em média, dentre todos os testes executados, foi estruturado com 20 camadas *ReLU* (10 no codificador e 10 no decodificador), otimizador Adam, função de perda MSE, 100 epochs e batches de tamanho 64.

As métricas atingidas por ele encontram-se nas Tabelas 4 e 5, podendo-se observar o contraste de desempenho com cada conjunto de dados utilizado.

Tabela 5 – Melhor modelo AE CSE-CIC.

Subset	Acurácia	Precisão	Recall	F1-score
CSE-CIC - 400	0,18375	0,810818	0,125937	0,21586
CSE-CIC - 800	0,186792	0,816118	0,125942	0,217308
CSE-CIC - 1300	0,149615	0,895585	0,092722	0,167755
CSE-CIC - 5000	0,2858	0,985228	0,276373	0,431541

Fonte: A Autora

Para uma análise mais aprofundada das previsões realizadas pelo modelo foram estruturadas as matrizes de confusão de uma rodada de treinamento com cada *subset* (Figuras 20 e 21).

O treinamento com KDD99 resulta em um modelo com melhor capacidade de prever valores de tráfego anormal como tal, mas também é perceptível uma certa dificuldade na classificação correta de dados de tráfego normal.

Figura 20 – Matrizes de confusão: Modelo AE KDD99.

		Previsto	
		Anormal	Normal
Real	Anormal	69	9
	Normal	2	1

(a) KDD99 - 400

		Previsto	
		Anormal	Normal
Real	Anormal	122	32
	Normal	5	2

(b) KDD99 - 800

		Previsto	
		Anormal	Normal
Real	Anormal	212	41
	Normal	2	6

(c) KDD99 - 1300

		Previsto	
		Anormal	Normal
Real	Anormal	809	182
	Normal	6	3

(d) KDD99 - 5000

Fonte: A Autora

Figura 21 – Matrizes de confusão: Modelo AE CSE-CIC.

		Previsto	
		Anormal	Normal
Real	Anormal	11	56
	Normal	2	11

(a) CSE-CIC - 400

		Previsto	
		Anormal	Normal
Real	Anormal	28	116
	Normal	2	13

(b) CSE-CIC - 800

		Previsto	
		Anormal	Normal
Real	Anormal	28	211
	Normal	3	17

(c) CSE0-CIC - 1300

		Previsto	
		Anormal	Normal
Real	Anormal	294	696
	Normal	2	8

(d) CSE-CIC - 5000

Fonte: A Autora

O mesmo não acontece quando o treinamento é realizado nos dados do CSE-CIC. Neste caso, a taxa de falsos negativos é muito grande, com muitas instâncias de dados anormais sendo classificados como normais.

Em um modelo que se propõe a detectar intrusões em rede, não pode-se aceitar uma taxa tão alta de anomalias que são ignoradas pelo sistema.

5.6.2 Clusterização: K-means

Nas Tabelas 6 e 7 são apresentados os resultados obtidos no treinamento do modelo selecionado após a aplicação da clusterização.

Tabela 6 – Aplicação de K-means: AE KDD99.

Subset	Acurácia	Precisão	Recall	F1-score
CSE-CIC - 400	0,437037	0,496362	0,452975	0,585174
CSE-CIC - 800	0,710559	0,87967	0,765251	0,811185
CSE-CIC - 1300	0,639464	0,765109	0,736772	0,737407
CSE-CIC - 5000	0,6907	0,838157	0,72959	0,77694

Fonte: A Autora

A Tabela 6 contém os resultados da execução do modelo com os *subsets* do KDD99 e a Tabela 7 contém os resultados da execução do modelo com os *subsets* do CSE-CIC.

Tabela 7 – Aplicação de K-means: AE CSE-CIC.

Subset	Acurácia	Precisão	Recall	F1-score
CSE-CIC - 400	0,1725	0,875505	0,109546	0,193608
CSE-CIC - 800	0,189937	0,819137	0,125942	0,220115
CSE-CIC - 1300	0,151538	0,848069	0,092257	0,1662
CSE-CIC - 5000	0,2849	0,991611	0,274592	0,429923

Fonte: A Autora

É possível observar que a clusterização teve efeitos majoritariamente negativos nos resultados, se comparados com os obtidos nos treinamentos sem a clusterização.

A única métrica que apresentou melhoria mínima com a clusterização foram a acurácia dos teste com CSE-CIC 800 e CSE-CIC 1300. Mas mesmo nestas intâncias, as outras métricas apresentaram piora.

Além disso, alguns resultados do KDD99 400 resultaram em 0 TP em certas instâncias, resultando em Precisão e Recall 0.

Analisando as matrizes de confusão nas Figuras 22 e 23, geradas para este experimento, nota-se uma melhoria na classificação de intâncias normais como tal, menos no treinamento com 5000 dados.

Figura 22 – Matrizes de confusão: Modelo AE KDD99 K-means.

		Previsto	
		Anormal	Normal
Real	Anormal	65	12
	Normal	0	4

(a) KDD99 - 400

		Previsto	
		Anormal	Normal
Real	Anormal	130	27
	Normal	2	2

(b) KDD99 - 800

		Previsto	
		Anormal	Normal
Real	Anormal	207	48
	Normal	0	6

(c) KDD99 - 1300

		Previsto	
		Anormal	Normal
Real	Anormal	634	203
	Normal	162	1

(d) KDD99 - 5000

Fonte: A Autora

Figura 23 – Matrizes de confusão: Modelo AE CSE-CIC K-means.

		Previsto	
		Anormal	Normal
Real	Anormal	11	63
	Normal	1	5

(a) CSE-CIC - 400

		Previsto	
		Anormal	Normal
Real	Anormal	21	118
	Normal	3	17

(b) CSE-CIC - 800

		Previsto	
		Anormal	Normal
Real	Anormal	25	219
	Normal	2	14

(c) CSE0-CIC - 1300

		Previsto	
		Anormal	Normal
Real	Anormal	287	702
	Normal	4	7

(d) CSE-CIC - 5000

Fonte: A Autora

É importante considerar, neste caso, a característica do K-means não performar bem com dados de grandes dimensões, que pode ter sido a causa dos resultados observados, já que o KDD99 possui 42 *features* e o CSE-CIC possui 79.

Outro ponto que pode ter causado dificuldades para este método é a possível existência de *outliers* nos conjuntos de dados, já que a análise da existência destes não foi realizada durante esta pesquisa.

Observa-se que, neste estado dos dados, a aplicação da clusterização K-means é prejudicial para o desempenho do modelo.

As comparações com a clusterização MeanShift é feita na Seção 5.6.3. A análise comparativa com a aplicação de MV encontra-se na Seção 5.6.4.

5.6.3 Clusterização - MeanShift

A clusterização MeanShift, assim como a K-means, piorou o desempenho do modelo orginal quando aplicada individualmente.

Tabela 8 – Aplicação de MeanShift: AE KDD99.

Subset	Acurácia	Precisão	Recall	F1-score
KDD99 - 400	0,646914	0,801484	0,676488	0,725497
KDD99 - 800	0,557143	0,697432	0,672584	0,663341
KDD99 - 1300	0,2123	0,222358	0,815559	0,349386
KDD99 - 5000	0,7902	0,97478	0,804181	0,8808

Fonte: A Autora

Analisando as Tabelas 8 e 9 de resultados e comparando-as com 6 e 7 respectivamente, não há diferenças significativas.

Tabela 9 – Aplicação de MeanShift: AE CSE-CIC.

Subset	Acurácia	Precisão	Recall	F1-score
CSE-CIC - 400	0,19125	0,895829	0,12878	0,223762
CSE-CIC - 800	0,166038	0,78996	0,114782	0,199339
CSE-CIC - 1300	0,142308	0,790546	0,091175	0,162943
CSE-CIC - 5000	0,305	0,952759	0,271184	0,421477

Fonte: A Autora

Destaca-se apenas que K-means produz os melhores resultados com os *subsets* de 800 e 1300 dados e MeanShift com os de 400 e 5000 dados.

Figura 24 – Matrizes de confusão: Modelo AE KDD99 MeanShift.

		Previsto	
		Anormal	Normal
Real	Anormal	63	15
	Normal	2	1

(a) KDD99 - 400

		Previsto	
		Anormal	Normal
Real	Anormal	129	30
	Normal	2	0

(b) KDD99 - 800

		Previsto	
		Anormal	Normal
Real	Anormal	222	45
	Normal	732	1

(c) KDD99 - 1300

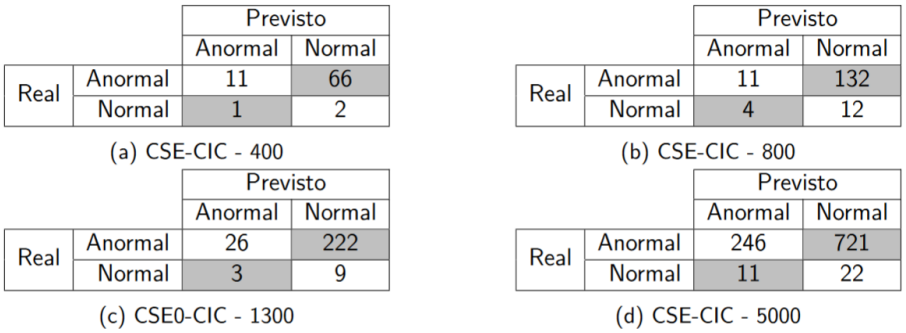
		Previsto	
		Anormal	Normal
Real	Anormal	796	195
	Normal	3	6

(d) KDD99 - 5000

Fonte: A Autora

Ambos os *datasets* utilizados no treinamento resultam em altos índices de alarmes falsos (previsões incorretas), para os teste com MeanShift (Figuras 24 e 25).

Figura 25 – Matrizes de confusão: Modelo AE CSE-CIC MeanShift.



		Previsto	
		Anormal	Normal
Real	Anormal	11	66
	Normal	1	2

(a) CSE-CIC - 400

		Previsto	
		Anormal	Normal
Real	Anormal	11	132
	Normal	4	12

(b) CSE-CIC - 800

		Previsto	
		Anormal	Normal
Real	Anormal	26	222
	Normal	3	9

(c) CSE0-CIC - 1300

		Previsto	
		Anormal	Normal
Real	Anormal	246	721
	Normal	11	22

(d) CSE-CIC - 5000

Fonte: A Autora

Por sofrer dos mesmos problemas apresentados para o algoritmo K-means, acredita-se que a performance ruim do MeanShift deva-se a *outliers* e à grande dimensionalidade dos dados aplicados.

5.6.4 Majority Voting

As métricas da aplicação do algoritmo do AE combinado com MV encontram-se nas Tabelas 10 e 11. A execução do AE, juntamente com MV e clusterização, apresentou os resultados apresentados nas Tabelas 12 e 13.

Tabela 10 – Aplicação de Majority Voting: AE KDD99.

Subset	Acurácia	Precisão	Recall	F1-score
KDD99 - 400	0,82716	0,992279	0,829718	0,903477
KDD99 - 800	0,791304	0,980471	0,801797	0,881889
KDD99 - 1300	0,803448	0,98723	0,810778	0,890111
KDD99- 5000	0,8049	0,998878	0,805288	0,89163

Fonte: A Autora

Tabela 11 – Aplicação de Majority Voting: AE CSE-CIC.

Subset	Acurácia	Precisão	Recall	F1-score
CSE-CIC - 400	0,195	0,826732	0,121928	0,211696
CSE-CIC - 800	0,2	0,844353	0,129508	0,224294
CSE-CIC - 1300	0,145769	0,838777	0,092052	0,165253
CSE-CIC - 5000	0,2778	0,994007	0,2705	0,425214

Fonte: A Autora

Contrariamente ao ocorrido anteriormente, com a aplicação da clusterização sobre o modelo de AE, a combinação da clusterização com *majority voting* apresentou resultados consistentemente melhores do que a aplicação exclusiva de MV.

Figura 26 – Matrizes de confusão: Modelo AE KDD99 *Majority Voting*.

		Previsto	
		Anormal	Normal
Real	Anormal	63	15
	Normal	0	3

(a) KDD99 - 400

		Previsto	
		Anormal	Normal
Real	Anormal	128	28
	Normal	2	3

(b) KDD99 - 800

		Previsto	
		Anormal	Normal
Real	Anormal	207	47
	Normal	5	2

(c) KDD99 - 1300

		Previsto	
		Anormal	Normal
Real	Anormal	791	202
	Normal	1	6

(d) KDD99 - 5000

Fonte: A Autora

A mudança mais perceptível pode ser observada na comparação entre as matrizes de confusão das Figuras 26 e 28 e das Figuras 27 e 29.

O modelo treinado com KDD99, clusterização e votação é mais capaz de identificar dados de tráfego normal, com instâncias de FN zeradas na maior parte dos testes.

Figura 27 – Matrizes de confusão: MModelo AE CSE-CIC *Majority Voting*.

		Previsto	
		Anormal	Normal
Real	Anormal	11	61
	Normal	1	7

(a) CSE-CIC - 400

		Previsto	
		Anormal	Normal
Real	Anormal	15	124
	Normal	4	16

(b) CSE-CIC - 800

		Previsto	
		Anormal	Normal
Real	Anormal	19	216
	Normal	2	23

(c) CSE0-CIC - 1300

		Previsto	
		Anormal	Normal
Real	Anormal	268	720
	Normal	1	11

(d) CSE-CIC - 5000

Fonte: A Autora

Por outro lado, a junção de MV e clusterização no treinamento com o CSE-CIC gerou aumento na proporção de dados classificados erroneamente como normais.

Comparando os resultados desta Seção com os obtidos no modelo AE simples, não é possível notar melhoria significante com a aplicação de MV. Supõe-se que isso aconteça devido ao treinamento dos modelos.

Como todas as instâncias do AE são treinadas com o mesmo *subset* pequeno de dados, é possível que os aprendizados sejam muito semelhantes e as previsões não apresentam grande variação, diminuindo assim a efetividade da técnica aplicada.

Esta interpretação dos resultados baseia-se nas medidas pequenas de desvio padrão das métricas obtidas.

Tabela 12 – Aplicação de K-means & Majority Voting: AE KDD99.

Subset	Acurácia	Precisão	Recall	F1-score
KDD99 - 400	0,824691	0,998438	0,816744	0,89804
KDD99 - 800	0,822981	0,989923	0,82394	0,898853
KDD99 - 1300	0,847126	0,998553	0,841246	0,913108
KDD99- 5000	0,8424	1	0,836405	0,910874

Fonte: A Autora

Tabela 13 – Aplicação de K-means & Majority Voting: AE CSE-CIC.

Subset	Acurácia	Precisão	Recall	F1-score
CSE-CIC - 400	0,2425	0,814621	0,128991	0,221795
CSE-CIC - 800	0,238994	0,832406	0,136837	0,233917
CSE-CIC - 1300	0,195769	0,876723	0,099953	0,179117
CSE-CIC - 5000	0,324	0,837366	0,264859	0,402338

Fonte: A Autora

Figura 28 – Matrizes de confusão: Modelo AE KDD99 K-means & *Majority Voting*.

		Previsto	
		Anormal	Normal
Real	Anormal	70	8
	Normal	0	3

(a) KDD99 - 400

		Previsto	
		Anormal	Normal
Real	Anormal	129	23
	Normal	3	6

(b) KDD99 - 800

		Previsto	
		Anormal	Normal
Real	Anormal	218	36
	Normal	0	7

(c) KDD99 - 1300

		Previsto	
		Anormal	Normal
Real	Anormal	792	172
	Normal	0	36

(d) KDD99 - 5000

Fonte: A Autora

Figura 29 – Matrizes de confusão: Modelo AE CSE-CIC K-means & *Majority Voting*.

		Previsto	
		Anormal	Normal
Real	Anormal	10	59
	Normal	4	7

(a) CSE-CIC - 400

		Previsto	
		Anormal	Normal
Real	Anormal	22	117
	Normal	5	15

(b) CSE-CIC - 800

		Previsto	
		Anormal	Normal
Real	Anormal	22	205
	Normal	4	29

(c) CSE0-CIC - 1300

		Previsto	
		Anormal	Normal
Real	Anormal	228	620
	Normal	45	107

(d) CSE-CIC - 5000

Fonte: A Autora

Diferente dos resultados obtidos com MV e K-means, as métricas de MV e MeanShift (Tabelas 14 e 15) não são consistentes entre os *subsets* de treinamento e também não apresentam melhoras se comparados ao modelo original.

Tabela 14 – Aplicação de MeanShift & Majority Voting: AE KDD99.

Subset	Acurácia	Precisão	Recall	F1-score
KDD99 - 400	0,8	0,976669	0,812421	0,886727
KDD99 - 800	0,792547	0,995314	0,792361	0,882024
KDD99 - 1300	0,2085	0,218646	0,80662	0,343875
KDD99- 5000	0,8118	0,995297	0,813441	0,895205

Fonte: A Autora

Tabela 15 – Aplicação de MeanShift & Majority Voting: AE CSE-CIC.

Subset	Acurácia	Precisão	Recall	F1-score
CSE-CIC - 400	0,1375	0,832521	0,092504	0,164492
CSE-CIC - 800	0,177987	0,917552	0,137707	0,238652
CSE-CIC - 1300	0,133462	0,920238	0,104769	0,187826
CSE-CIC- 5000	0,2874	0,967038	0,27503	0,428221

Fonte: A Autora

Analisando as matrizes de confusao (Figuras 30 e 31), de forma geral, observa-se que a clusterização MeanShift não auxiliou na melhora do modelo original, mas ambas as técnicas de clusterização, quando pareadas com votação, apresentaram melhores resultados do que quando aplicadas individualmente.

Figura 30 – Matrizes de confusão: Modelo AE KDD99 MeanShift & *Majority Voting*.

		Previsto	
		Anormal	Normal
Real	Anormal	69	11
	Normal	1	0

(a) KDD99 - 400

		Previsto	
		Anormal	Normal
Real	Anormal	125	34
	Normal	0	2

(b) KDD99 - 800

		Previsto	
		Anormal	Normal
Real	Anormal	214	47
	Normal	738	1

(c) KDD99 - 1300

		Previsto	
		Anormal	Normal
Real	Anormal	818	172
	Normal	2	8

(d) KDD99 - 5000

Fonte: A Autora

A combinação de clusterização K-means com MV apresentou melhoras reais no modelo treinado com ambos os *datasets*, destacando-se como a melhor implementação dentre todos os testes realizados.

Figura 31 – Matrizes de confusão: Modelo AE CSE-CIC MeanShift & *Majority Voting*.

		Previsto	
		Anormal	Normal
Real	Anormal	7	68
	Normal	1	4

(a) CSE-CIC - 400

		Previsto	
		Anormal	Normal
Real	Anormal	18	127
	Normal	1	13

(b) CSE-CIC - 800

		Previsto	
		Anormal	Normal
Real	Anormal	23	228
	Normal	3	6

(c) CSE0-CIC - 1300

		Previsto	
		Anormal	Normal
Real	Anormal	276	699
	Normal	10	15

(d) CSE-CIC - 5000

Fonte: A Autora

Uma possível explicação para a clusterização melhorar os resultados do modelo com *majority voting* é que a clusterização aumenta o desvio padrão das métricas, ou seja, gera maior variação nas previsões dos modelos. Esta diferença entre as previsões parece ser aproveitada pelo algoritmo de MV.

6 Considerações Finais

O estudo e reprodução de *frameworks* desenvolvidos por outros pesquisadores é uma maneira bastante completa de se compreender o funcionamento de técnicas diversas e explorar ferramentas em busca de resultados positivos.

Existe uma base comparativa de resultados gerados pelos algoritmos que permitem compreender melhor as diferenças que são observadas no desenvolvimento, como a qualidade das combinações de funções para modelos de aprendizado de máquina e os efeitos da dimensionalidade de dados sobre o processamento.

Foi possível compreender também que a criação de um único modelo que possa realizar detecção de intrusão com diversos conjuntos de dados distintos é uma tarefa complexa, que demanda bastante pesquisa e ainda pode ser estudada mais a fundo em projetos futuros.

Devido a restrições de tempo, não foi possível estudar com profundidade quais problemas impediram o bom funcionamento do modelo de RBM, assim como quais questões impediram um melhor desempenho do modelo de autoencoder.

A partir de todos os testes realizados, o melhor modelo observado foi estruturado com autoencoder e melhorado com clusterização K-means e *ensemble learning Majority Voting*.

A combinação de técnicas de melhoria de aprendizado podem ser bastante benéficas ao desenvolvimento de boas estruturas de previsão, já que individualmente podem não ter tanto impacto sobre o treinamento.

A área de pesquisa que trata de segurança de redes sempre se beneficia de descobertas e avanços, e a exploração de novas técnicas permite visualizar novos caminhos a serem explorados.

6.1 Trabalhos Futuros

A partir do trabalho apresentado é possível buscar diversas melhorias e aprofundamentos, buscando por melhores desempenhos dos modelos apresentados.

Algumas abordagens podem ser destacadas:

- Estudar o efeito das dimensões dos dados nos resultados obtidos, trabalhando com técnicas de seleção de características para destacar as mais importantes na identificação dos dados;
- Estudar os tempos de execução de cada modelo apresentado no projeto e fazer uma análise utilizando este parâmetro para definir aplicabilidade e praticidade;

- Estudar os efeitos de outras técnicas de clusterização, como Clustream, para o pré-processamento dos dados;
- Estudar a estrutura do autoencoder e desenvolver um modelo mais robusto e capaz de apresentar um melhor aprendizado;
- Estudar a estrutura da RBM e desenvolver um modelo funcional e capaz de apresentar um melhor aprendizado;
- Estudar outras possibilidades de aplicação para a RBM, como seletora de característica ou pareada com um algoritmo de aprendizado profundo.

Referências

AL-QATF, M.; LASHENG, Y.; AL-HABIB, M.; AL-SABAHI, K. Deep learning approach combining sparse autoencoder with svm for network intrusion detection. *IEEE Access*, v. 6, p. 52843–52856, 2018.

ALASHHAB, Z. R.; ANBAR, M.; SINGH, M. M.; LEAU, Y.-B.; AL-SAI, Z. A.; ALHAYJA'A, S. A. Impact of coronavirus pandemic crisis on technologies and cloud computing applications. *Journal of Electronic Science and Technology*, v. 19, n. 1, p. 100059, 2021.

ALDWAIRI, T.; PERERA, D.; NOVOTNY, M. A. An evaluation of the performance of restricted boltzmann machines as a model for anomaly network intrusion detection. *Computer Networks*, v. 144, p. 111–119, 2018. ISSN 1389-1286.

ALOM, M. Z.; TAHA, T. M. Network intrusion detection for cyber security using unsupervised deep learning approaches. In: *2017 IEEE National Aerospace and Electronics Conference (NAECON)*. [S.l.: s.n.], 2017. p. 63–69.

ALRAWASHDEH, K.; PURDY, C. Toward an online anomaly intrusion detection system based on deep learning. In: *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*. [S.l.: s.n.], 2016. p. 195–200.

BAIR, E. Semi-supervised clustering methods. *WIREs Computational Statistics*, v. 5, n. 5, p. 349–361, 2013.

BAJAJ, A. Performance Metrics in Machine Learning [Complete Guide]. *Neptune*, dez. 2022. Disponível em: https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide. Acesso em: 9 ago. 2022.

BANERJI, A. K-Mean Clustering: Methods To Find The Best Value Of K. *Analytics Vidhya*, nov. 2022. Disponível em: https://www.analyticsvidhya.com/blog/2021/05/k-mean-getting-the-optimal-number-of-clusters. Acesso em: 12 set. 2022.

BEYELER, M. *Machine Learning for OpenCV*. [S.l.]: Packt Publishing, 2019. ISBN 978-1-78398028-4.

CHU, Y.; ZHAO, X.; ZOU, Y.; XU, P. *A Decoding Scheme for Incomplete Motor Imagery EEG With Deep Belief Network*. 2018. Disponível em: https://www.researchgate.net/publication/327936128_A_Decoding_Scheme_for_Incomplete_Motor_Imagery_EEG_With_Deep_Belief_Network. Acesso em: 28 ago. 2021.

DANSBECKER. *Rectified Linear Units (ReLU) in Deep Learning*. 2018. Disponível em: https://www.kaggle.com/code/dansbecker/rectified-linear-units-relu-in-deep-learning/notebook. Acesso em: 7 abr. 2022.

DAWOUD, A.; SIANAKI, O. A.; SHAHRISTANI, S.; RAUN, C. Internet of things intrusion detection: A deep learning approach. *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, n. 1, 2020.

DEVELOPERS, O. C. *OpenCV: Meanshift and Camshift*. 2020. Disponível em: https://docs.opencv.org/3.4/d7/d00/tutorial_meanshift.html. Acesso em: 9. Jan. 2023.

DEVELOPERS, S. learn. *Sklearn Preprocessing: Normalize*. 2022a. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.normalize.html. Acesso em: 8 ago. 2022.

DEVELOPERS, S. learn. *SciKit-Learn Preprocessing: MinMaxScaler*. 2022b. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html. Acesso em: 8 ago. 2022.

DEVELOPERS, T. *TensorFlow*. 2021. Disponível em: https://www.tensorflow.org/?hl=pt-br. Acesso em: 27 ago. 2021.

DIETTERICH, T. G. *Ensemble Learning*. 2002. Disponível em: https://courses.cs.washington.edu/courses/cse446/12wi/tgd-ensembles.pdf. Acesso em: 20 jul. 2021.

FARAHNAKIAN, F.; HEIKKONEN, J. A deep auto-encoder based approach for intrusion detection system. In: *2018 20th International Conference on Advanced Communication Technology (ICACT)*. [S.l.: s.n.], 2018. p. 178–183.

FELDMANN, A.; GASSER, O.; LICHTBLAU, F.; PUJOL, E.; POESE, I.; DIETZEL, C.; WAGNER, D.; WICHTLHUBER, M.; TAPIADOR, J.; VALLINA-RODRIGUEZ, N.; HOHLFELD, O.; SMARAGDAKIS, G. A year in lockdown: How the waves of covid-19 impact internet traffic. *Communications of the ACM*, v. 64, n. 7, p. 101–108, 2021.

GOOGLE. *Classification: Precision and Recall*. 2020. Disponível em: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall. Acesso em: 7 abr. 2022.

GOOGLE. *Normalization*. 2021. Disponível em: https://developers.google.com/machine-learning/data-prep/transform/normalization. Acesso em: 7 abr. 2022.

HENRIQUEZ, M. *92% of data breaches in Q1 2022 due to cyberattacks.* 2022. Disponível em: https://www.securitymagazine.com/articles/97431-92-of-data-breaches-in-q1-2022-due-to-cyberattacks. Acesso em: 25 maio. 2022.

IBM. *What is Unsupervised Learning?* 2020. Disponível em: https://www.ibm.com/cloud/learn/unsupervised-learning. Acesso em: 6 abr. 2022.

IERACITANO, C.; ADEEL, A.; MORABITO, F. C.; HUSSAIN, A. A novel statistical analysis and autoencoder driven intelligent intrusion detection approach. *Neurocomputing*, v. 387, p. 51–62, 2020. ISSN 0925-2312.

LAROCHELLE, H.; MANDEL, M.; PASCANU, R.; BENGIO, Y. Learning algorithms for the classification restricted boltzmann machine. *Journal of Machine Learning Research*, v. 13, p. 643–669, 2012.

LEEVY, J. L.; KHOSHGOFTAAR, T. M. A survey and analysis of intrusion detection models based on cse-cic-ids2018 big data. *Journa of Big Data*, 2020.

PELTARION. *Mean squared error loss function*. 2022. Disponível em: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-squared-error. Acesso em: 6 abr. 2022.

PRAK, H. *Intuition of Adam Optimizer - GeeksforGeeks*. 2020. Disponível em: https://www.geeksforgeeks.org/intuition-of-adam-optimizer/. Acesso em: 7 abr. 2022.

PU, G.; WANG, L.; SHEN, J.; DONG, F. A hybrid unsupervised clustering-based anomaly detection method. *Tsinghua Science and Technology*, v. 26, n. 2, p. 146–153, 2021.

RASCHKA, S.; PATTERSON, J.; NOLET, C. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information*, MDPI, v. 11, n. 4, p. 193, 2020.

RAVIKUMAR, D. Towards enhancement of machine learning techniques using cse-cic-ids2018 cybersecurity dataset. *Thesis*, Rochester Institute of Technology, 2021.

RAWAT, A. S. *A Cost Function in Machine Learning: Analytics Steps*. 2021. Disponível em: https://www.analyticssteps.com/blogs/cost-function-machine-learning. Acesso em: 7 abr. 2022.

REBELLO, G. A. F.; REIS, M. L. dos; DAMASCENO, R. G.; SOUZA, R. O. S. d.; JESUS, R. C. R. d. *Sistemas de Detecção de Intrusão*. 2016. Disponível em: https://www.gta.ufrj.br/grad/16_2/2016IDS/tipos.html. Acesso em: 25 jul. 2021.

RODER, M.; ROSA, G. H. de; PAPA, J. P. *Learnergy: Energy-based Machine Learners*. 2020.

SALO, F.; INJADAT, M.; MOUBAYED, A.; NASSIF, A. B.; ESSEX, A. Clustering enabled classification using ensemble feature selection for intrusion detection. In: IEEE. *2019 International Conference on Computing, Networking and Communications (ICNC)*. [S.l.], 2019. p. 276–281.

SEO, S.; PARK, S.; KIM, J. Improvement of network intrusion detection accuracy by using restricted boltzmann machine. In: *2016 8th International Conference on Computational Intelligence and Communication Networks (CICN)*. [S.l.: s.n.], 2016. p. 413–417.

SHARMA, S. *Activation Functions in Neural Networks - Towards Data Science*. 2017. Disponível em: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. Acesso em: 6 abr. 2022.

SHAUKAT, K.; LOU, S.; VARADHARAJAN, V.; HAMEED, I.; XU, M. A survey on machine learning techniques for cyber security in the last decade. *IEEE Access*, v. 8, p. 222310–222354, 2020.

SONG, Y.; HYUN, S.; CHEONG, Y.-G. Analysis of autoencoders for network intrusion detection. *Sensors*, v. 21, n. 13, 2021. ISSN 1424-8220.

STANFORD. *Unsupervised Feature Learning and Deep Learning Tutorial*. 2022. Disponível em: http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/. Acesso em: 6 abr. 2022.

TEAM, K. *Keras documentation: Layer activation functions*. 2017. Disponível em: https://keras.io/api/layers/activations/. Acesso em: 7 abr. 2022.

TOMMASO, Z.; CECCARELLI, A.; BONDAVALLI, A. Into the unknown: Unsupervised machine learning algorithms for anomaly-based intrusion detection. In: *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*. [S.l.]: IEEE, 2020. p. 81–81.

UNION, I. T. *Statistics*. 2019a. Disponível em: https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx. Acesso em: 25 jul. 2021.

UNION, I. T. *Digital Development Dashboard*. 2019b. Disponível em: https://www.itu.int/en/ITU-D/Statistics/Dashboards/Pages/Digital-Development.aspx. Acesso em: 25 jul. 2021.

WOOD, T. *F-Score*. 2019. Disponível em: https://deepai.org/machine-learning-glossary-and-terms/f-score. Acesso em: 7 abr. 2022.

YAMASHITA, T.; TANAKA, M.; YOSHIDA, E.; YAMAUCHI, Y.; FUJIYOSHII, H. To be bernoulli or to be gaussian, for a restricted boltzmann machine. In: *2014 22nd International Conference on Pattern Recognition*. [S.l.: s.n.], 2014. p. 1520–1525.

ZAVRAK, S.; İSKEFIYELI, M. Anomaly-based intrusion detection from network flow features using variational autoencoder. *IEEE Access*, v. 8, p. 108346–108358, 2020.

ZHANG, X.; CHEN, J. Deep learning based intelligent intrusion detection. In: *2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN)*. [S.l.: s.n.], 2017. p. 1133–1137.

ZHAO, R.; YIN, J.; XUE, Z.; GUI, G.; ADEBISI, B.; OHTSUKI, T.; GACANIN, H.; SARI, H. An efficient intrusion detection method based on dynamic autoencoder. *IEEE Wireless Communications Letters*, v. 10, n. 8, p. 1707–1711, 2021.

ZHOU, Z.-H. *Ensemble Learning*. [S.l.]: Springer, 2009. 270–273 p.