

UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"

FACULDADE DE CIÊNCIAS - CAMPUS BAURU

DEPARTAMENTO DE COMPUTAÇÃO

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JOÃO PEDRO MARIN COMINI

**DETECÇÃO DE ANOMALIAS UTILIZANDO AUTOENCODER
VARIACIONAL**

BAURU

Novembro/2019

JOÃO PEDRO MARIN COMINI

DETECÇÃO DE ANOMALIAS UTILIZANDO AUTOENCODER VARIACIONAL

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

Orientador: Prof. Dr. Kelton Augusto Pontara Costa

BAURU

Novembro/2019

João Pedro Marin Comini Detecção de Anomalias utilizando Autoencoder Variacional/ João Pedro Marin Comini. – Bauru, Novembro/2019- 35 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Kelton Augusto Pontara Costa

Trabalho de Conclusão de Curso – Universidade Estadual Paulista “Júlio de Mesquita Filho”

Faculdade de Ciências

Bacharelado em Ciência da Computação, Novembro/2019.

1. Aprendizado de Máquina 2. Detecção de anomalias 3. Redes de computadores
4. Classificação

João Pedro Marin Comini

Detecção de Anomalias utilizando Autoencoder Variacional

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

Banca Examinadora

Prof. Dr. Kelton Augusto Pontara Costa

Orientador

Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Prof. Assoc. Dr. João Pedro Albino

Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Profa. Dra. Simone das Graças Domingues Prado

Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Bauru, _____ de _____ de _____.

À todos que acreditam que progresso e conhecimento consolidam-se através da ciência.

Agradecimentos

Agradeço em primeiro lugar aos meus pais, que estiveram comigo durante esses 4 longos anos de graduação, pelo imensurável apoio e por me proporcionar as oportunidades necessárias para me desenvolver e alcançar os meus sonhos. Meu caminho seria muito mais difícil sem seu suporte.

Agradeço a minha companheira, Gabriela, que acompanhou de perto os últimos dois anos dessa jornada, pela paciência, compreensão e pelas incontáveis conversas, desabafos e apoio. Sua presença é fundamental na minha vida.

Agradeço veemente aos meus companheiros de curso, principalmente ao Circo. Vocês, com certeza, tornaram todo esse caminho muito mais leve e prazeroso.

Agradeço aos meus colegas de trabalho, por me acolherem e me aguentarem durante as incontáveis horas de TCC durante o expediente, esse lugar foi definitivamente um berço para mim. Acabou o Covil (ou não).

Agradeço, finalmente, aos professores deste maravilhoso curso, por me guiar e fornecer o conhecimento necessário para a minha formação.

Be water, my friend.

Bruce Lee

Resumo

A Internet se desenvolveu de forma exponencial nos últimos anos, junto a isso, os riscos de agentes maliciosos atuarem se tornou ainda maior. Neste trabalho, realizou-se um estudo profundo sobre um modelo de aprendizagem de máquina conhecido como *autoencoder* variacional, que foi treinado e desenvolvido junto a diversos modelos de aprendizado de máquina, implementados através das *frameworks* TensorFlow e Keras para a linguagem de programação Python. Utilizou-se o conjunto de dados NSL-KDD, uma versão refinada do conjunto de dados KDDcup99, cuja preparação e tratamento também foram abordados em um capítulo deste trabalho. Os modelos foram desenvolvidos com o objetivo de avaliar sua efetividade no campo de detecção de anomalias, mais especificamente ao se tratar de anomalias em redes de computadores. Seus resultados foram comparados com diversos classificadores já estabelecidos na área e resultados satisfatórios foram obtidos. Espera-se, então, que este trabalho sirva de apoio para trabalhos futuros envolvendo *autoencoders* variacionais e/ou detecção de anomalias.

Palavras-chave: Detecção de anomalias; Autoencoder; Inferência Variacional; Aprendizado de Máquina.

Abstract

The Internet has developed exponentially in recent years, and the risk of malicious agents acting has become even greater. In this work, an in-depth study of a neural network model known as variational autoencoder, which was trained and developed along with several machine learning models, implemented through TensorFlow and Keras frameworks for the Python programming language. We used the NSL-KDD dataset, a refined version of the KDDcup99 dataset, whose preparation and treatment were also covered in a chapter of this paper. The models were developed with the objective of evaluating their effectiveness in the field of anomaly detection, more specifically when dealing with anomalies in computer networks. Their results were compared with several classifiers already established in the area and good results were obtained. This project is then expected to support future work involving variational autoencoders and/or anomaly detection.

Keywords: Anomaly detection; Autoencoder; Variational Inference; Machine Learning.

Lista de figuras

Figura 1 – Exemplo simples de anomalias em um espaço bidimensional.	15
Figura 2 – Estrutura básica de um <i>autoencoder</i> . Reconstrução de X para \tilde{X}	16
Figura 3 – Esquema simplificado de um <i>VAE</i>	17
Figura 4 – Crescimento das linguagens de programação nos países de alta renda	22
Figura 5 – <i>One-hot encoding</i> aplicado à variável <i>protocol_type</i>	24
Figura 6 – Esquema simplificado do método de codificação.	26
Figura 7 – Espaço latente do autoencoder variacional	27
Figura 8 – acurácia durante o treinamento da rede neural.	28
Figura 9 – Erro durante o treinamento da rede neural.	29
Figura 10 – Divisão do espaço utilizando um hiperplano	29
Figura 11 – Espaço latente com modelo treinado apenas com dados normais	31
Figura 12 – Espaço latente em 3D com modelo treinado apenas com dados normais	31

Lista de tabelas

Tabela 1 – Características básicas de cada conexão	20
Tabela 2 – Comparação: KDDcup99 e NSL-KDD	20
Tabela 3 – Resultado: Classificador rede neural	28
Tabela 4 – Resultado: Classificador bayesiano	29
Tabela 5 – Resultado: Máquina de Vetores de Suporte	30
Tabela 6 – Resultado: Floresta Aleatória	30
Tabela 7 – Resultado geral de todos os classificadores	30

Sumário

1	INTRODUÇÃO	13
1.1	Objetivos	13
1.1.1	Objetivo Geral	13
1.1.2	Objetivos Específicos	13
2	FUNDAMENTAÇÃO TEÓRICA	15
2.1	Deteção de anomalias	15
2.2	<i>Autoencoders</i>	16
2.2.1	<i>Autoencoder Variacional</i>	17
2.2.2	Otimização e o Truque de Reparametrização	18
2.3	Conjunto de Dados NSL-KDD	18
2.3.1	Características do conjunto	19
3	METODOLOGIA	21
3.1	Etapas	21
3.2	Ferramentas	21
3.2.1	Python	21
3.2.2	Keras API e TensorFlow	22
3.2.3	Scikit-learn	23
3.2.4	Matplotlib	23
3.2.5	Git	24
3.3	Tratamento do conjunto de dados	24
3.4	Desenvolvimento do modelo	25
3.5	Abordagens utilizando o modelo	25
4	RESULTADOS	27
4.1	Classificando através da representação codificada	27
4.1.1	Rede neural Artificial com aprendizado profundo	27
4.1.2	Classificador Bayesiano	28
4.1.3	Máquina de Vetores de Suporte	29
4.1.4	Floresta aleatória de decisão	30
4.1.5	Resultados Gerais	30
4.2	Classificando através do erro de reconstrução	30
5	CONCLUSÃO	33

REFERÊNCIAS 34

1 Introdução

De acordo com um estudo realizado no ano de 2007 na Universidade de Maryland nos Estado Unidos, acontece um ataque *hacker* a cada 29 segundos (CUKIER, 2007), junto a isso, devido ao grande aumento de aplicações tecnológicas e ao grande fluxo de informações, a segurança dos dados tem se tornado um trabalho cada vez mais complexo. Inúmeras técnicas são utilizadas para garantir a preservação de usuários e servidores, normalmente compostas de várias tecnologias e camadas, como detecção de intrusões, *firewalls* e antivírus (YOUSEFI-AZAR et al., 2017).

Ao se tratar dos sistemas de detecção de intrusão baseados em anomalias, a tentativa é de detectar padrões extraordinários no conjunto de dados que não estão de acordo com o comportamento comum esperado. Esses padrões incomuns são também chamados de anomalias, exceções, *outliers*, entre outros termos. A detecção de anomalias tem um vasto campo de uso em aplicações como na detecção de fraudes de cartão de crédito, ataques à servidores e até na vigilância militar (BHUYAN; BHATTACHARYYA; KALITA, 2017).

O Aprendizado de Máquina é um campo da Inteligência Artificial que tem sido vastamente utilizado em aplicações de detecção de anomalias, os métodos desenvolvidos são capazes de aprender padrões de ataques conhecidos e desconhecidos utilizando métodos supervisionados, não-supervisionados e semi-supervisionados, os quais possuem suas vantagens e desvantagens (YOUSEFI-AZAR et al., 2017).

1.1 Objetivos

O objetivo geral e os objetivos específicos deste trabalho são listados a seguir.

1.1.1 Objetivo Geral

Este trabalho teve por objetivo realizar um estudo aprofundado sobre *autoencoders* e inferência variacional aplicados a detecção de anomalias, mais especificamente, ao conjunto de dados NSL-KDD, que contém dados de intrusões simuladas em uma rede de computadores militar. Após isto, colher os resultados e comparar com outros métodos de detecção de anomalias.

1.1.2 Objetivos Específicos

- Realizar um estudo aprofundado sobre o modelo proposto por Kingma (KINGMA; WELLING, 2013);

- Desenvolver um modelo de Autoencoder Variacional que possua aplicabilidade no campo de detecção de anomalias;
- Tratar o conjunto de dados escolhido e utiliza-lo para o treinamento do modelo acima;
- Comparar os resultados obtidos com outros modelos já estabelecidos no campo de detecção de anomalias.

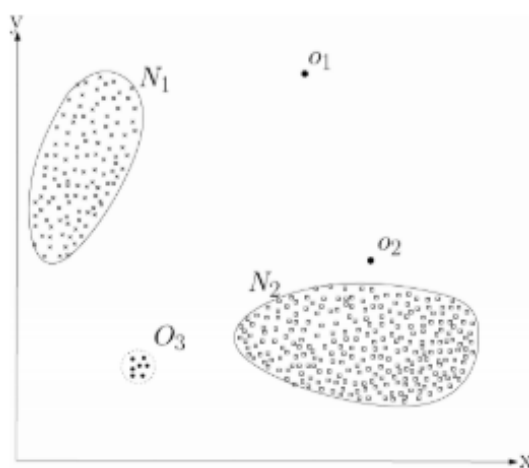
No próximo capítulo, são abordados os estudos e a coleta de material sobre detecção de anomalias e a rede neural artificial utilizados durante a realização de trabalho.

2 Fundamentação Teórica

2.1 Detecção de anomalias

A Internet como um todo avançou de forma exponencial nos últimos anos, devido a isso, houve um grande aumento do risco de ataques em redes de computadores. Apesar de grandes progressos nos estudos e sistemas sobre detecção de intrusão, ainda há muitas oportunidades para avanços no estado da arte de detecção de anomalias (BHUYAN; BHATTACHARYYA; KALITA, 2017).

Figura 1 – Exemplo simples de anomalias em um espaço bidimensional.



Fonte – Chandola, Banerjee e Kumar (2009)

Vemos um exemplo claro de anomalias em um espaço bidimensional na figura 1. Os pontos que possuem um valor anormal acabam ficando distantes dos aglomerados maiores (valores normais).

Falando em um nível mais abstrato, anomalias são definidas como padrões que possuem um comportamento diferente do padrão definido como "normal". Porém, esta simples definição esconde os grandes desafios que os sistemas de detecção de anomalias têm que superar. Para Chandola, Banerjee e Kumar (2009), os principais fatores que dificultam este processo são os seguintes:

- Para se definir o que é uma anomalia, antes, é necessário definir o conceito de normalidade no contexto que está sendo trabalhado. Frequentemente a fronteira que divide estes dois comportamentos não é precisa e é difícil de encontrar;
- Anomalias que são resultados de ações maliciosas normalmente tentam se camuflar em meio ao comportamento normal o que torna ainda mais difícil essa separação;

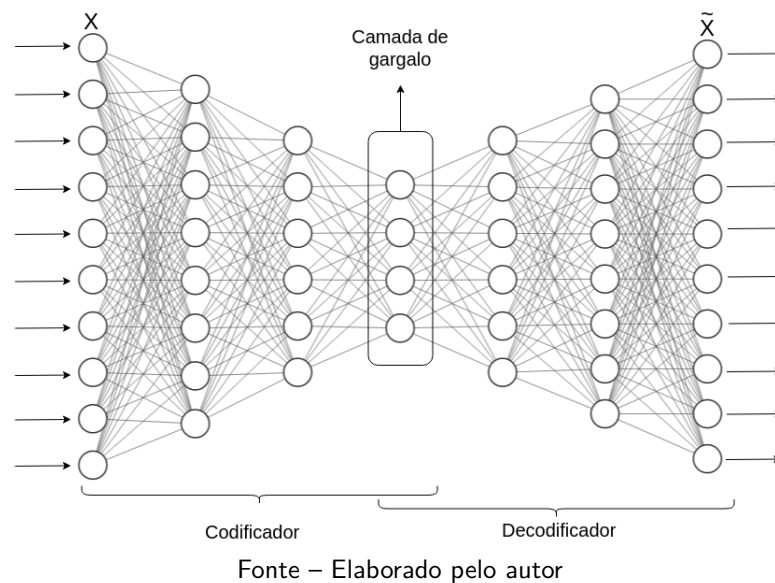
- A noção de anomalia é diferente para cada domínio de aplicação. Por exemplo, pequenas variações em um exame médico pode determinar uma anomalia, assim como grandes variações no mercado de ações pode ser um comportamento normal. Então, desenvolver uma solução para vários domínios não é uma tarefa simples.

Neste trabalho, são discutidas e avaliadas diversas abordagens desenvolvidas junto a um modelo de *autoencoder* variacional com a finalidade de detectar anomalias no conjunto de dados NSL-KDD.

2.2 Autoencoders

Em aprendizado de máquina, existem muitas abordagens possíveis para o desenvolvimento de softwares com capacidade de aprender de forma autônoma. Neste trabalho é discutido um modelo específico conhecido como *Autoencoder Variacional*.

Figura 2 – Estrutura básica de um *autoencoder*. Reconstrução de X para \tilde{X}



Os autoencoders são modelos de rede neural artificial que possuem como objetivo aprender representações codificadas dos dados de forma eficiente e não-supervisionada (KRAMER, 1991). Esses modelos são compostos de duas partes: um codificador $g_{\phi}(x) = z$ e um decodificador $f_{\theta}(z) = \tilde{x}$. O codificador possui a função de reduzir sua entrada a uma dimensão menor que a dimensão original, dessa forma, extraindo uma representação latente z da entrada x . Essa representação é passada ao decodificador que, por sua vez, possui a função de reconstruir a entrada original a partir dela. É possível notar essa estrutura na figura 2, uma rede neural composta de um codificador e um decodificador, no qual a menor camada da rede

também é chamada de camada de gargalo, camada responsável por manter a representação codificada latente do conjunto de dados.

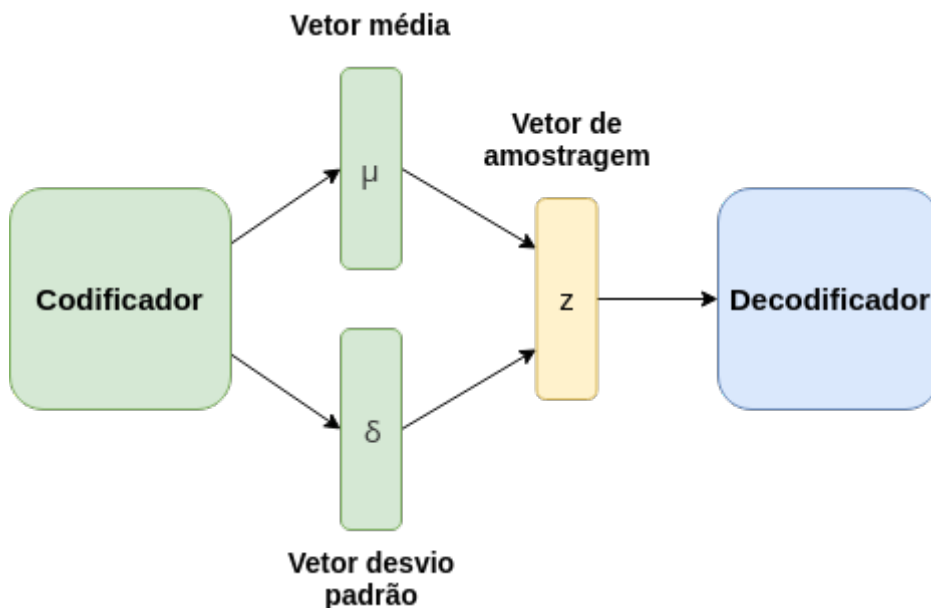
2.2.1 Autoencoder Variacional

Um modelo de *autoencoder* utiliza métodos de otimização que objetivam reduzir ao máximo o erro de reconstrução os entre dados de entrada e os dados reconstruídos, *i.e.* minimizar a função custo. Uma função de erro muito utilizada em *autoencoders* comuns para o cálculo do erro de reconstrução é a função Erro Quadrático Médio (EQM) que calcula a média da diferença quadrática entre os dados de saída da rede e os dados esperados:

$$EQM = \frac{1}{n} \sum_{i=1}^n (X_i - \tilde{X}_i)^2 \quad (1)$$

Em *autoencoders* variacionais (VAEs), modelo proposto por Kingma e Welling (2013), o processo de otimização e a arquitetura do possuem modificações. A rede neural artificial do codificador e do decodificador passam a ser os modelos probabilísticos $q_\phi(z|x)$ e $p_\theta(x|z)$ respectivamente, utilizando um método estatístico conhecido como Inferência Variacional. O problema que o VAE tenta resolver está relacionado ao comportamento do *autoencoder* comum, cujo decodificador assume que as codificações pertencem a uma distribuição desconhecida e não considera as distribuições fora dela.

Figura 3 – Esquema simplificado de um VAE



Fonte – Elaborado pelo autor

A figura 3 contém um simples esquema de um autoencoder variacional. A camada que contém a representação codificada dos dados, *i.e.*, camada de gargalo, é dividida em outras

três camadas: um vetor média μ , um vetor desvio padrão σ e um vetor de amostragem z .

O vetor de amostragem é o responsável por realizar as operações de amostragem na distribuição $q_\phi(z|x)$ através dos vetores μ e σ .

2.2.2 Otimização e o Truque de Reparametrização

O algoritmo de retropropagação é utilizado para ajustar os parâmetros do codificador q e do decodificador p . Porém, junto ao erro de reconstrução, é adicionada à função custo uma medida conhecida como Divergência de Kullback-Leibler ou Entropia Relativa. Explicando de forma simples, esta medida calcula o erro de aproximação entre as distribuições q e p , assim é possível manter a codificação dos dados em uma distribuição conhecida, a equação da medida é definida da seguinte forma:

$$D_{KL}(p||q) = \mathbb{E}_{x \sim p}[\log p(x) - \log q(x)] \quad (2)$$

O processo de otimização em um *autoencoder* comum é simples, executa-se apenas o algoritmo de retropropagação e propaga-se o erro de reconstrução para as camadas anteriores, corrigindo seus pesos. *Autoencoder* Variacionais são otimizados de uma forma mais complexa devido a operação de amostragem, que é descontínua e não possui gradiente (DOERSCH, 2016), portanto seus parâmetros não podem ser otimizados diretamente utilizando o algoritmo de retropropagação.

$$z \sim q(z|x) = \mathcal{N}(\mu, \sigma^2) \quad (3)$$

A operação de amostragem na equação 3, pode ser reparametrizada, de forma que o algoritmo consiga otimizar os parâmetros ϕ e θ das distribuições q e p respectivamente (codificador e decodificador), aproximando da seguinte forma:

$$z = \mu + \sigma \odot \epsilon \mid \epsilon \sim \mathcal{N}(0, I) \quad (4)$$

2.3 Conjunto de Dados NSL-KDD

A efetividade de um modelo de aprendizado de máquina frequentemente está relacionado ao conjunto de dados utilizado para treinamento, portanto, para este trabalho foi escolhido o conjunto de dados NSL-KDD, desenvolvido pelo Instituto Canadense de Cibersegurança, que é uma versão refinada do conjunto de dados KDD'99. KDD'99 foi uma competição de extração de dados no ano de 1999 conhecida como *Knowledge Discovery in Databases Cup*.

O conjunto é uma coletânea de dados de conexões normais e anormais em um ambiente de rede militar simulado, conjunto vastamente utilizado no desenvolvimento de sistemas de detecção de intrusão e anomalias (DHANABAL; SHANTHARAJAH, 2015).

A proposta do conjunto de dados escolhido é corrigir algumas falhas encontradas no conjunto original, KDD'99. Para isto as seguintes ações foram tomadas:

- Todos os registros redundantes foram removidos para que os classificadores consigam produzir um resultado imparcial;
- Possui-se uma quantidade suficiente de registros no conjunto de treino e teste, o que permite realizar experimentos com o conjunto inteiro.

2.3.1 Características do conjunto

Cada registro do conjunto NSL-KDD possui 41 características relacionadas a uma tentativa de conexão ao ambiente simulado. Algumas dessas características estão citadas na tabela 1.

O conjunto classifica os registros em 40 diferentes classes, a classe normal, que representa as conexões normais e inofensivas e outras 39 classes, que representam as conexões maliciosas. As conexões maliciosas podem ser divididas em 4 categorias de ataque:

- *DoS*: *DoS* é uma sigla que significa *Denial of Service*, que pode ser traduzida como "Negação de Serviço". É um tipo de ataque que visa esgotar os recursos da vítima, tornando-a incapaz de aceitar requisições legítimas.
- *Probing*: É um tipo de ataque que tem como objetivo roubar informações da vítima, remotamente.
- *U2R (User to Remote)*: O objetivo deste tipo de ataque é conseguir acessar o sistema local da vítima com os privilégios de administrador, explorando alguma vulnerabilidade da vítima.
- *R2L (Remote to Local)*: Este ataque visa atingir o sistema da vítima de forma remota, ganhando acesso local da máquina da vítima.

Por este trabalho tratar de detecção de anomalias, a classificação dos registros será dividida em apenas duas classes, normal e anormal, sendo as conexões maliciosas as conexões de classe "anormal".

Como citado anteriormente, no conjunto original KDDcup99 há muitos registros redundantes que foram removidos no conjunto NSL-KDD através de uma análise estatística,

Tabela 1 – Características básicas de cada conexão

#	Nome	Descrição	Exemplo
1	duration	Tempo de duração da conexão	0
2	protocol_type	Protocolo utilizado na conexão	tcp
3	service	Serviço de rede utilizado	ftp_data
4	flag	Status da conexão - Normal ou Erro	SF
5	src_bytes	Quantidade de bytes transferido da origem para o destino durante a conexão	492
6	dst_bytes	Quantidade de bytes transferido do destino para a origem durante a conexão	0
7	land	Se o IP e as Portas de destino e origem são iguais, esta variável recebe valor 1, senão, 0	0
8	wrong_fragment	Quantidade total de fragmentos errados durante a conexão	0
9	urgent	Quantidade de pacotes com o bit de urgência ativado	0

Fonte – Elaborado pelo autor

permitindo que os classificadores tenham um resultado imparcial. Na tabela 2 está uma comparação do conjunto original ao modificado.

Tabela 2 – Comparação: KDDcup99 e NSL-KDD

	Registros originais	Registros distintos	Taxa de Redução
Ataques	3925650	262178	93.32%
Normal	972781	812814	16.44%
Total	4898431	1074992	78.05%

Fonte – [Canadian Institute for Cybersecurity \(2009\)](#)

3 Metodologia

3.1 Etapas

O desenvolvimento deste trabalho consistiu-se em três principais etapas. Após o levantamento do material sobre estudos anteriores relacionados à detecção de anomalias e *autoencoders*, foram selecionadas possíveis implementações de classificação de anomalias utilizando um *autoencoder* variacional. O primeiro passo consistiu em encontrar e tratar um conjunto de dados relacionado a intrusões em redes de computadores, nesta etapa foi necessário utilizar algumas técnicas como *one-hot encoding* e normalização dos dados.

O segundo passo foi implementar um modelo de *autoencoder* variacional que produzisse resultados satisfatórios. Durante esta etapa o ambiente de desenvolvimento foi definido, optou-se por utilizar a linguagem de programação Python e as bibliotecas disponíveis para tal.

A execução do terceiro passo consistiu em colher resultados e comparar com outros métodos de detecção de anomalias.

As três etapas serão explicadas em detalhes neste capítulo.

3.2 Ferramentas

Nesta seção encontram-se as ferramentas utilizadas no trabalho como também os argumentos e as justificativas de tais escolhas.

3.2.1 Python

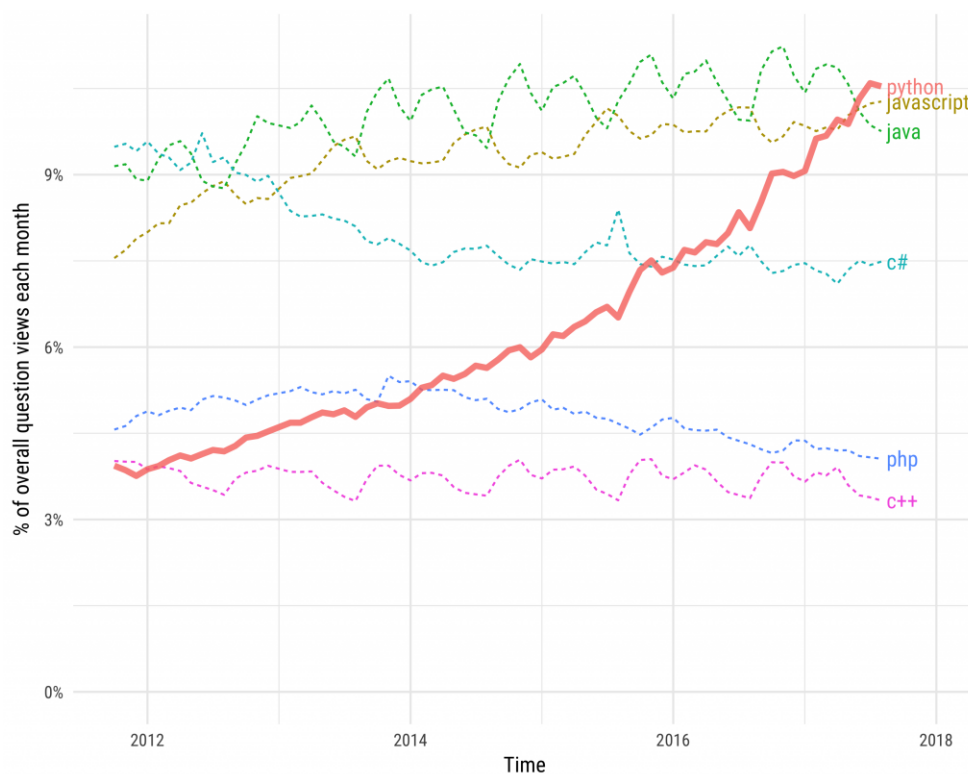
Na comunidade de aprendizado de máquina, Python é uma das linguagem de programação em maior crescimento nos últimos anos. De acordo com um estudo de um site de dúvidas conhecido como StackOverflow, em torno de 10% das perguntas visitadas em sua plataforma estão relacionados a linguagem Python, além disso, é a linguagem que mais cresce nos países de alta renda como é possível notar na figura 4.

A adoção desta linguagem pela comunidade deve-se à muitas de suas vantagens, mas as principais são:

- Menos esforço para mais resultado;
- Alta legibilidade do código;

- Comunidade ativa;
- Vasto leque de ferramentas de código aberto disponíveis.

Figura 4 – Crescimento das linguagens de programação nos países de alta renda



Fonte – Robinson (2017)

Neste trabalho, a escolha da linguagem deve-se principalmente pelo suporte das implementações das bibliotecas Keras API e Tensorflow, que serão destacadas a seguir.

3.2.2 Keras API e TensorFlow

Para o desenvolvimento do modelo de *autoencoder* variacional, optou-se pelas bibliotecas TensorFlow e Keras.

A biblioteca TensorFlow foi escolhida, principalmente, pela sua popularidade e grande quantidade de material disponível. Junto a ela, foi utilizada a Keras API que é uma camada de abstração por cima da biblioteca TensorFlow e ao mesmo tempo mantém a possibilidade de customização necessária para o trabalho (ABADI et al., 2015).

Com estas ferramentas, o desenvolvimento do modelo se torna rápido e direto. A abstração fornecida pela Keras API permite que não haja preocupações que não sejam relacionadas a arquitetura e treinamento do modelo (CHOLLET, 2015). Um trecho de código exemplo está descrito no código 3.1.

Código 3.1 – Exemplo de implementação de um autoencoder comum.

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense

model = Sequential()

model.add(Input(shape=(input_dim,)))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu')) # ENCODER

model.add(Dense(8, activation='relu')) # BOTTLENECK LAYER

model.add(Dense(16, activation='relu')) # DECODER
model.add(Dense(32, activation='sigmoid'))

```

3.2.3 Scikit-learn

Antes do treinamento do modelo, foi necessário realizar um pré-processamento do conjunto de dados. Para esta finalidade, a biblioteca scikit-learn possui inúmeras ferramentas disponíveis. As principais ferramentas utilizadas nesta etapa foram o *OneHotEncoder* e *MinMaxScaler* (ver seção 3.3), que têm a função de transformar as variáveis qualitativas em quantitativas e realizar a normalização dos dados, respectivamente.

Além das ferramentas de pré-processamento, a biblioteca também conta com uma vasta variedade de implementações de modelos de aprendizado de máquina, como por exemplo, o classificador bayesiano, máquina de vetores de suporte e floresta aleatória. Modelos estes, que foram treinados com o mesmo conjunto de dados e então utilizados para comparar aos resultados obtidos do modelo de *autoencoder* variacional implementado neste trabalho (PEDREGOSA et al., 2011).

3.2.4 Matplotlib

Na etapa de análise de resultado é muito importante ter uma representação visual do andamento do treino do modelo. Para isso, utilizou-se da biblioteca Matplotlib para Python que é uma ferramenta muito utilizada para plotagem na comunidade de ciência de dados (HUNTER, 2007).

O principal objetivo da ferramenta é facilitar o máximo as tarefas de visualização de dados, o que pode se tornar algo muito difícil. Com poucas linhas de código foi possível visualizar dados muito importantes durante o desenvolvimento do modelo, como o espaço latente do *autoencoder* variacional proposto no trabalho e métricas de performance.

3.2.5 Git

O git é uma ferramenta distribuída de código aberto de controle de versões, é amplamente utilizada no campo de desenvolvimento de software, porém tem a capacidade de registrar um histórico de modificações de qualquer tipo de arquivo (TORVALD, 2005).

Durante o desenvolvimento deste trabalho, a ferramenta foi utilizada junto a um repositório remoto na plataforma GitHub, mantendo o histórico do andamento do projeto, com a vantagem de ainda funcionar como um *backup*.


3.3 Tratamento do conjunto de dados

Para a utilização do conjunto de dados NSL-KDD no treinamento do *autoencoder*, alguns ajustes foram necessários. Como citado na seção 2.3, cada registro possui 41 variáveis, sendo que três delas são categóricas: *protocol_type*, *service* e *flag*. Variáveis categóricas são variáveis que não possuem um valor quantitativo, é uma variável que possui um valor de um conjunto finito de categorias.

Por exemplo, a característica *protocol_type* dos registros só pode assumir três valores: "tcp", "udp" e "icmp". Devido a este fato, torna-se necessário aplicar uma transformação nessas variáveis e transforma-las em variáveis quantitativas, para isto foi utilizada uma técnica chamada *one-hot encoding*. O que esta técnica faz é transformar o dado em uma representação "binária", sendo que há apenas um bit ativo por registro, que indica à qual categoria este pertence.

Na figura 5 há um exemplo de transformação em uma das variáveis do conjunto. A variável pode assumir apenas três valores qualitativos: tcp, udp e icmp, portanto, ela é substituída por três colunas com zeros e uns, nas quais somente uma das colunas pode ser "um" por linha do conjunto de dados.

Figura 5 – *One-hot encoding* aplicado à variável *protocol_type*

#	protocol_type		#	udp_protocol	tcp_protocol	icmp_protocol
1	tcp		1	0	1	0
2	udp		2	1	0	0
3	icmp		3	0	0	1
4	tcp		4	0	1	0

Fonte – Elaborado pelo autor

Para realizar essa transformação, utilizou-se da classe *OneHotEncoder*, disponibilizada através da biblioteca scikit-learn.

Após a transformação das variáveis categóricas foi necessário realizar a normalização dos dados quantitativos. A normalização é uma abordagem de uso muito comum que permite ao modelo aprender de forma rápida (JAYALAKSHMI; SANTHAKUMARAN, 2011), o que ela faz é reduzir a escala numérica dos dados para um intervalo conhecido. Para o conjunto de dados utilizado neste trabalho o intervalo escolhido de 0 a 1, para isso fez-se uso da classe *MinMaxScaler* da biblioteca scikit-learn, que executa a seguinte transformação no conjunto de dados:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \quad (5)$$

3.4 Desenvolvimento do modelo

O modelo de *autoencoder* variacional deste trabalho recebeu inspirações de trabalhos anteriores, propostos por An e Cho (2015) e Yousefi-Azar et al. (2017).

Após estudos e diversos testes durante o desenvolvimento, decidiu-se que o modelo teria uma arquitetura de rede neural artificial com 7 camadas: três camadas para o codificador, uma para a camada de gargalo e as outras três para o decodificador, lembrando que em um *autoencoder* variacional a camada de gargalo é dividida em outras três camadas como citado na seção 2.2.1. As camadas de neurônios têm as seguintes dimensões, em sequência: 96, 64, 32, 16, 32, 64, 96.

O treinamento do modelo foi executado utilizando os conceitos de retropropagação e reparametrização citados na seção 2.2.2. As bibliotecas Keras e TensorFlow permitiram que esta etapa permanecesse direta e simples, mesmo com a necessidade de algumas customizações para a implementação do modelo.

O conjunto foi dividido em duas partes: o conjunto de treino e o de teste. O conjunto de teste serve como um parâmetro para saber se o modelo está aprendendo de modo generalista, já que esse conjunto nunca é utilizado no treinamento.

3.5 Abordagens utilizando o modelo

O *autoencoder* variacional por si só não funciona como um classificador, a primeira abordagem utilizada após o treinamento foi usar a representação codificada extraída do codificador e utilizá-la como entrada para os classificadores. Acredita-se que a codificação extraída

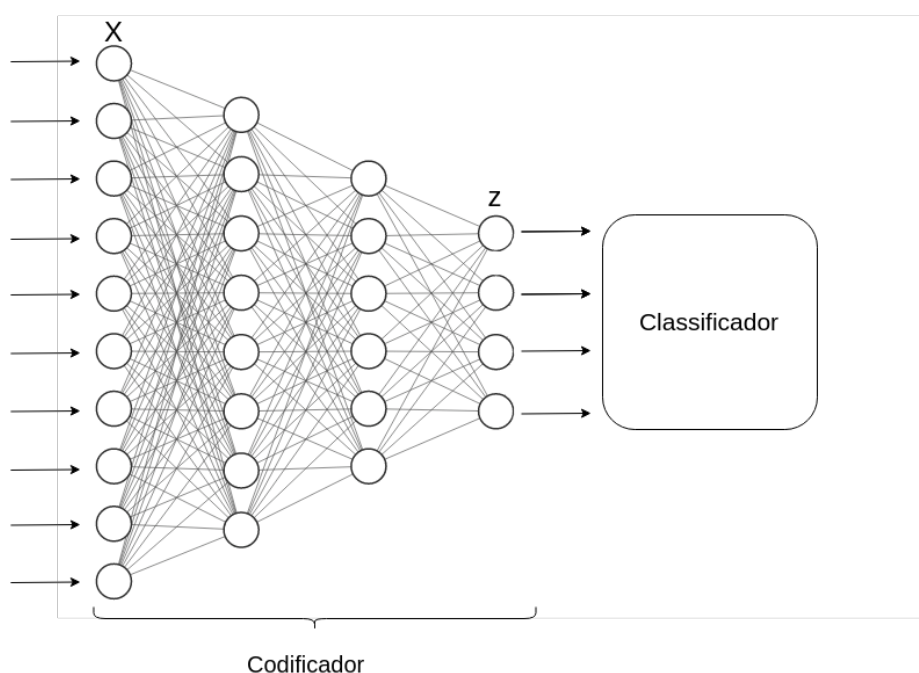
contenha uma representação mais selecionada dos dados que contenha apenas informações relevantes, neste caso, o modelo funciona como um seletor de variáveis.

Os classificadores utilizados com esta abordagem foram: Rede Neural Artificial com aprendizado profundo, Classificador Bayesiano, Máquina de Vetores de Suporte e Floresta Aleatória de Decisão. Cada tipo possui dois modelos treinados, um treinado a partir da representação normal dos dados e o outro treinado a partir da representação codificada. Essa abordagem está exemplificada na figura 6, a saída do codificador é passada diretamente ao classificador.

Outra possível abordagem implementada neste trabalho, é treinar o *autoencoder* apenas com os dados classificados como normais e então utilizar o erro de reconstrução como parâmetro para a classificação. Esta abordagem parte do princípio de que o *autoencoder* "aprendeu" a reconstruir apenas comportamentos normais e então o erro de reconstrução dos dados anormais seriam significativamente maiores (AN; CHO, 2015). Os passos para o desenvolvimento desta abordagem seriam os seguintes:

- Treinar o *autoencoder* variacional apenas com dados normais.
- Calcular o erro de reconstrução dos dados.
- Classificar como anômalos os dados que possuem um erro ϵ maior que o limite L estabelecido.

Figura 6 – Esquema simplificado do método de codificação.



Fonte – Elaborado pelo autor

4 Resultados

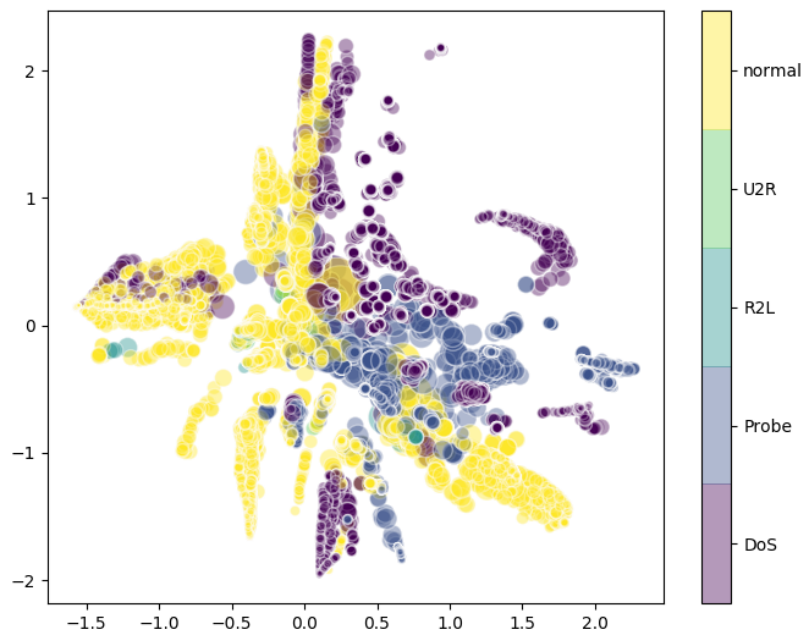
Neste capítulo, serão discutidos os resultados obtidos utilizando as abordagens definidas anteriormente. A métrica utilizada como parâmetro foi a acurácia, que apenas calcula a porcentagem de acertos.

$$acc = \frac{n_{acertos}}{n_{total}} * 100 \quad (6)$$

4.1 Classificando através da representação codificada

O primeiro método foi desenvolvido realizando o treinamento de vários classificadores a partir da representação codificada do conjunto de dados.

Figura 7 – Espaço latente do autoencoder variacional



Após o treinamento do modelo, é possível visualizar a representação do seu espaço latente, isto é, visualizar a disposição das codificações na distribuição inferida. Nota-se que os dados tendem a se aglomerar com seus semelhantes.

4.1.1 Rede neural Artificial com aprendizado profundo

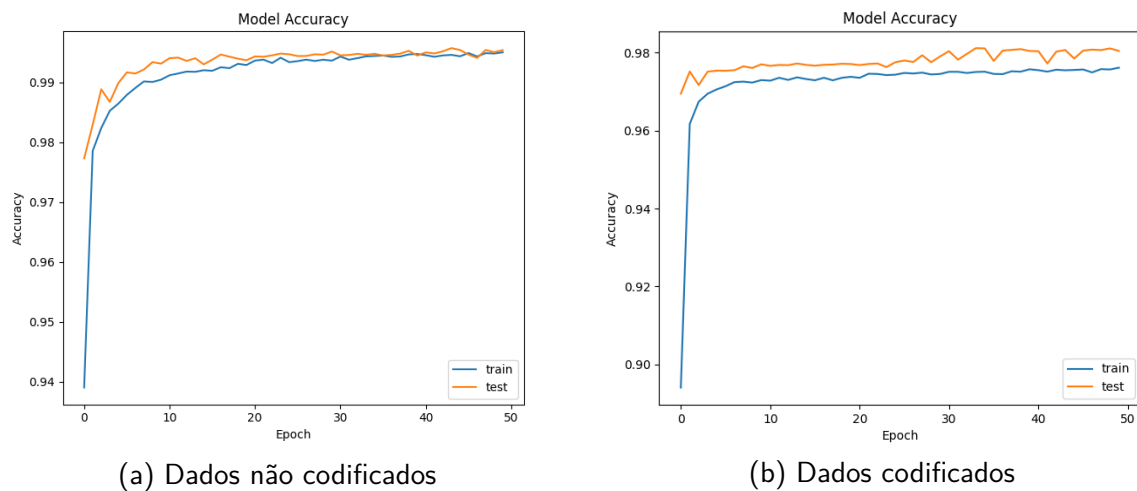
Duas redes neurais artificiais simples com apenas quatro camadas foram treinadas, uma com o conjunto codificado e a outra com o conjunto normal, o objetivo das redes era classificar a entrada como normal ou anormal. Após 50 épocas, obtiveram-se os resultados da tabela 3

Tabela 3 – Resultado: Classificador rede neural

	Dados normais	Dados codificados
acurácia	99.62%	97.63%

Fonte – Elaborado pelo autor

Figura 8 – acurácia durante o treinamento da rede neural.



Fonte – Elaborado pelo autor

Percebe-se que para este classificador o resultado não foi ótimo. A acurácia foi menor para os dados codificados, porém ainda se manteve em um nível satisfatório. A perda de acurácia deve-se principalmente à perda de informações importantes durante a codificação dos dados.

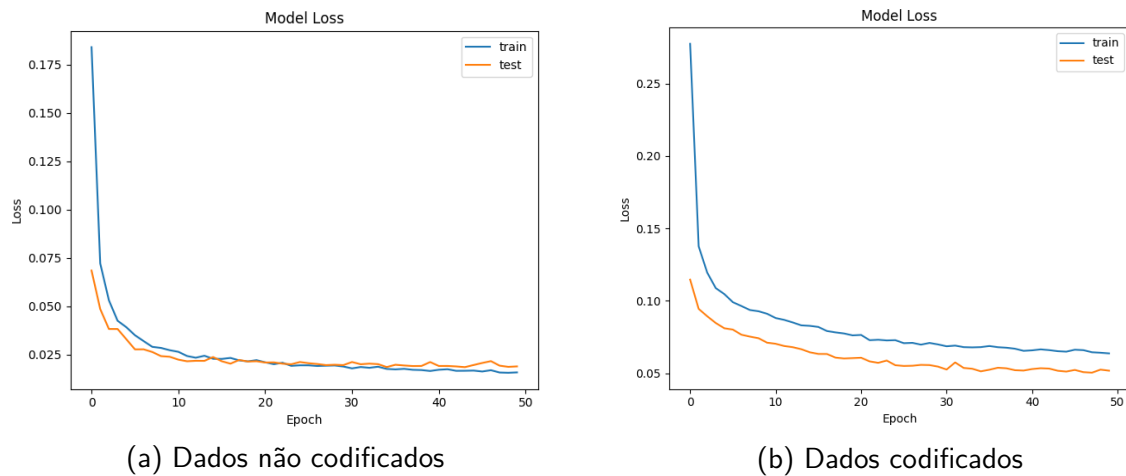
Entretanto, é possível notar na figura 8, que o modelo utilizando os dados codificados atingiu sua acurácia máxima mais rapidamente. Na figura 9 há a redução do erro das redes durante o treinamento, nota-se que com os dados codificados a rede demorou mais para alcançar o erro mínimo.

4.1.2 Classificador Bayesiano

O classificador bayesiano, conhecido mais popularmente como *Naive Bayes Classifier*, é uma família de classificadores probabilísticos muito eficientes e efetivos. Possui aplicações em diversas áreas, como diagnóstico médico e classificação de texto. É um classificador ainda muito utilizado devido a sua performance competitiva e simplicidade (ZHANG, 2004). Os resultados estão na tabela 4.

O classificador obteve uma melhoria significativa com os dados codificados. Neste caso o *autoencoder* funcionou bem como um seletor de variáveis e extraiu uma representação mais vantajosa para o classificador bayesiano.

Figura 9 – Erro durante o treinamento da rede neural.



Fonte – Elaborado pelo autor

Tabela 4 – Resultado: Classificador bayesiano

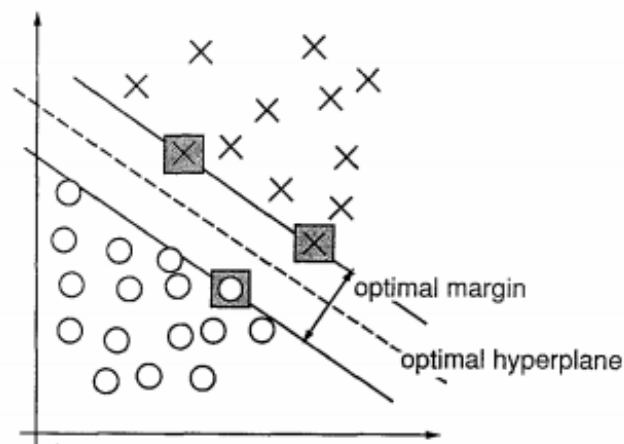
	Dados normais	Dados codificados
acurácia	85.90%	87.21%

Fonte – Elaborado pelo autor

4.1.3 Máquina de Vetores de Suporte

Modelo proposto por Cortes e Vapnik (1995), o *Support Vector Classifier* é um classificador muito utilizado hoje em dia. O objetivo deste classificador é dividir o espaço dos dados em partes, assim, cada parte é definida como o domínio de uma classe, um particionamento de um espaço bidimensional está exemplificado na figura 10.

Figura 10 – Divisão do espaço utilizando um hiperplano



Fonte – Cortes e Vapnik (1995)

O classificador manteve melhores resultados utilizando a representação não codificada dos dados. Porém, a diferença foi pequena, o que é um bom indicador de que o *autoencoder* está produzindo uma codificação fiel dos dados originais. Os resultados estão na tabela 5.

Tabela 5 – Resultado: Máquina de Vetores de Suporte

	Dados normais	Dados codificados
acurácia	98.49%	96.60%

Fonte – Elaborado pelo autor

4.1.4 Floresta aleatória de decisão

Este foi o classificador com melhores resultados, com os dados originais e os dados codificados. O modelo foi proposto por Ho (1995), suas grandes vantagens são a performance e a velocidade do treinamento. Os resultados estão citados na tabela 6.

Tabela 6 – Resultado: Floresta Aleatória

	Dados normais	Dados codificados
acurácia	99.89%	99.74%

Fonte – Elaborado pelo autor

O modelo foi capaz de classificar 99,74% utilizando os dados codificados. Percebe-se como o *autoencoder* variacional funciona muito bem como um seletor de variáveis.

4.1.5 Resultados Gerais

Na tabela 7 há todos os resultados obtidos com os classificadores nesse trabalho, com destaque para a manutenção dos resultados mesmo com os dados codificados.

Tabela 7 – Resultado geral de todos os classificadores

Classificador	Dados normais	Dados codificados
RNA	99.62%	97.63%
CB	85.90%	87.21%
MVS	98.49%	96.60%
FAD	99.89%	99.74%

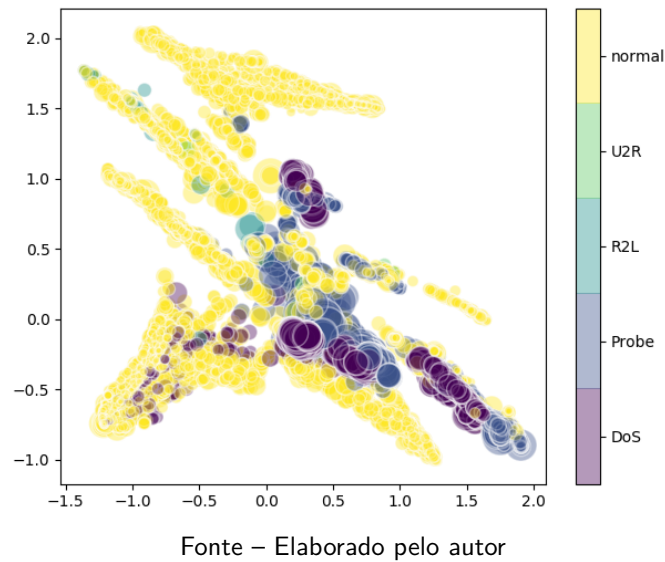
Fonte – Elaborado pelo autor

4.2 Classificando através do erro de reconstrução

Outro método de classificação desenvolvido neste trabalho envolve treinar o *autoencoder* variacional apenas com dados de conexões "normais". Após o treinamento, o espaço latente da

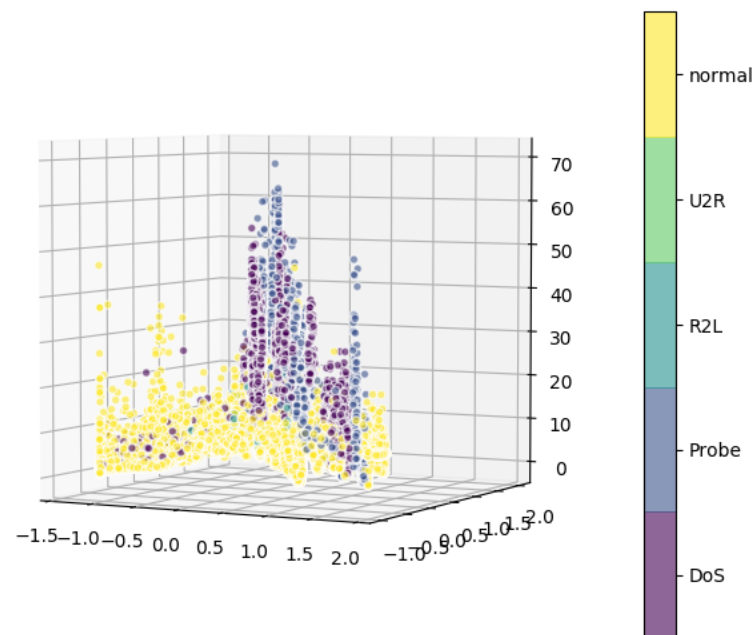
rede pode ser visualizado na figura 11. O espaço latente na figura 11 é a projeção dos dados codificados em um espaço bidimensional.

Figura 11 – Espaço latente com modelo treinado apenas com dados normais



Na figura 11, quanto maior for o ponto, maior é o erro de reconstrução do dado, desta forma, é possível notar que os pontos amarelos (dados normais) possuem um erro bem menor em comparação aos dados anômalos.

Figura 12 – Espaço latente em 3D com modelo treinado apenas com dados normais



Já na figura 12, o eixo Y representa o valor do erro para cada entrada. Nota-se que os pontos amarelos se mantêm na parte baixa do gráfico, o que vem a corroborar com o princípio citado na seção 3.5. O *autoencoder* é capaz de replicar com mais acurácia apenas os dados considerados normais. O objetivo desta abordagem era traçar um plano paralelo ao plano XY que conseguisse separar ao máximo os dados anômalos dos dados normais.

Com esta simples abordagem, foi possível classificar corretamente 93,51% dos dados, resultado melhor que o alcançado pelo classificador bayesiano.

5 Conclusão

O trabalho aqui desenvolvido apresenta métodos e ferramentas para a implementação de técnicas para o campo de detecção de anomalias, especificamente em redes de computadores, utilizando um *autoencoder* variacional. As técnicas desenvolvidas podem ser generalizadas para qualquer campo de atuação, além do apresentado neste trabalho.

Os objetivos propostos foram cumpridos, um estudo profundo sobre *autoencoders* variacionais foi realizado e as técnicas implementadas obtiveram resultados satisfatórios, porém a perda de informação causada pela redução de dimensionalidade foi o principal limitante da assertividade dos métodos aqui apresentados.

Ainda há espaço para melhorias dos métodos aqui apresentados, assim como a possibilidade da utilização de outros conjuntos de dados.

Todo o código do trabalho está disponível na plataforma GitHub ¹ sob a licença MIT, com pretensão de fomentar ainda mais a comunidade de aprendizado de máquina e servir como apoio para estudos futuros.

¹ <<https://github.com/JoaoComini/tcc-vae>>

Referências

- 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. Disponível em: <<https://www.tensorflow.org/>>. Acesso em: 16 Out. 2019.
- AN, J.; CHO, S. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, v. 2, n. 1, 2015.
- BHUYAN, M. H.; BHATTACHARYYA, D. K.; KALITA, J. K. Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys & Tutorials*, IEEE, 5 2017.
- CANADIAN INSTITUTE FOR CYBERSECURITY. *NSL-KDD dataset*. 2009. Disponível em: <<https://www.unb.ca/cic/datasets/nsl.html>>. Acesso em: 01 Out. 2019.
- CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, ACM, v. 41, n. 3, p. 15, 2009.
- CHOLLET, F. *Keras: The Python Deep Learning library*. 2015. Disponível em: <<https://www.keras.io/>>. Acesso em: 16 Out. 2019.
- CORTES, C.; VAPNIK, V. Support-vector networks. *Machine Learning*, v. 20, n. 3, p. 273–297, Sep 1995. ISSN 1573-0565.
- CUKIER, M. *Study: Hackers Attack Every 39 Seconds*. 2007. Disponível em: <<https://eng.umd.edu/news/story/study-hackers-attack-every-39-seconds>>. Acesso em: 29 Set. 2019.
- DHANABAL, L.; SHANTHARAJAH, S. A study on nsl-kdd dataset for intrusion detection system based on classification algorithms. *International Journal of Advanced Research in Computer and Communication Engineering*, v. 4, 6 2015.
- DOERSCH, C. *Tutorial on Variational Autoencoders*. 2016.
- HO, T. K. Random decision forests. In: IEEE. *Proceedings of 3rd international conference on document analysis and recognition*. [S.l.], 1995. v. 1, p. 278–282.
- HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007.
- JAYALAKSHMI, T.; SANTHAKUMARAN, A. Statistical normalization and back propagation for classification. *International Journal of Computer Theory and Engineering*, v. 3, n. 1, p. 1793–8201, 2011.
- KINGMA, D. P.; WELLING, M. Auto-encoding variational bayes. 2013.

KRAMER, M. A. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, v. 37, p. 233–243, 1991.

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.

ROBINSON, D. *The Incredible Growth of Python*. 2017. Acesso em 16 de Outubro de de 2019. Disponível em: <<https://stackoverflow.blog/2017/09/06/incredible-growth-python>>. Acesso em: 16 Out. 2019.

TORVALD, L. *Git*. 2005. Disponível em: <<https://git-scm.com/>>. Acesso em: 17 Out. 2019.

YOUSEFI-AZAR, M.; VARADHARAJAN, V.; HAMEY, L.; TUPAKULA, U. Autoencoder-based feature learning for cyber security applications. In: IEEE. *2017 International joint conference on neural networks (IJCNN)*. [S.l.], 2017. p. 3854–3861.

ZHANG, H. The optimality of naive bayes. *AA*, v. 1, n. 2, p. 3, 2004.