Universidade Federal de Ouro Preto
Escola de Minas
CECAU - Colegiado do Curso de
Engenharia de Controle e Automação

Gabriela Ottoni Santa Bárbara Bartolozzi Chaves

Predição de Falhas em Válvulas de um Sistema de Refrigeração via Aprendizado de Máquina

Monografia de Graduação

Ouro Preto, 2022

Gabriela Ottoni Santa Bárbara Bartolozzi Chaves

Predição de Falhas em Válvulas de um Sistema de Refrigeração via Aprendizado de Máquina

Trabalho apresentado ao Colegiado do Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para a obtenção do Grau de Engenheira(o) de Controle e Automação.

Universidade Federal de Ouro Preto

Orientador: Prof. Eduardo José da Silva Luz, Dr.

Coorientador: Prof. Agnaldo José da Rocha Reis, Dr.

Ouro Preto

2022

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

C512p Chaves, Gabriela Ottoni Santa Barbara Bartolozzi.
Predição de falhas em válvulas de um sistema de refrigeração via aprendizado de máquina. [manuscrito] / Gabriela Ottoni Santa Barbara Bartolozzi Chaves. - 2022.
80 f.: il.: color., gráf., tab..

Orientador: Prof. Dr. Eduardo José da Silva Luz.
Coorientador: Prof. Dr. Agnaldo José da Rocha Reis.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Escola de Minas. Graduação em Engenharia de Controle e Automação .

1. Inteligência Artificial. 2. Aprendizado do computador. 3. Redes neurais (Computação). 4. Válvulas. 5. Heating, Ventilation and Air Conditioning (HVAC). 6. Tecnologia - Acurácia. I. Luz, Eduardo José da Silva. II. Reis, Agnaldo José da Rocha. III. Universidade Federal de Ouro Preto. IV. Título.

CDU 681.5

Bibliotecário(a) Responsável: Maristela Sanches Lima Mesquita - CRB-1716

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
ESCOLA DE MINAS
DEPARTAMENTO DE ENGENHARIA CONTROLE E AUTOMACAO

FOLHA DE APROVAÇÃO

Gabriela Otoni Santa Bárbara Bartolozzi Chaves

Predição de Falhas em Válvulas de um Sistema de Refrigeração via Aprendizado de Máquina

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Engenheira de Controle e Automação

Aprovada em 01 de novembro de 2022

Membros da banca

Doutor -Eduardo José da Silva Luz - Orientador (Universidade Federal de Ouro Preto)
Doutor - Agnaldo José da Rocha Reis - (Universidade Federal de Ouro Preto)
Doutora - Luciana Gomes Castanheira - (Universidade Federal de Ouro Preto)
Doutora - Adrielle de Carvalho Santana - (Universidade Federal de Ouro Preto)

Eduardo José da Silva Luz, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 01/11/2022

Documento assinado eletronicamente por **Eduardo Jose da Silva Luz**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 03/11/2022, às 11:09, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0421312** e o código CRC **FF40D601**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591533 - www.ufop.br

Agradecimentos

Agradeço primeiro a Deus por permitir que eu trilhasse esse caminho com muita dedicação e força para superar os desafios.

Agradeço à minha mãe Rejane por sempre me apoiar nas minhas decisões e torcer pelo meu sucesso. Ao meu pai Nando por sempre acreditar e me fazer acreditar na minha própria capacidade. A eles devo tudo que sou e são para eles todas as minhas conquistas.

Às minhas amigas-irmãs da melhor e maior República de Ouro Preto, a República Joselitas, por estarem comigo em todos os momentos da minha graduação vibrando com minhas vitórias.

Aos meus colegas de curso e profissão, em especial ao Anderson, Douglas e Wesley por terem compartilhado comigo tantos aprendizados fazendo os meus dias de faculdade mais alegres e leves.

Por fim, agradeço a todos meus professores que dedicaram tempo para compartilhar seus conhecimentos e me fizeram ser a profissional que eu sou hoje, em especial ao meu orientador Eduardo e ao meu coorientador Agnaldo por acreditarem em mim e no meu projeto sempre me orientando com muita sabedoria e prontidão.

"Por vezes sentimos que aquilo que fazemos não é, senão, uma gota de água no mar.
Mas o mar seria menor se lhe faltasse uma gota."
(Madre Teresa de Calcutá)

Resumo

Com o grande aumento das indústrias e o avanço das tecnologias, tornou-se necessário obter um bom sistema de refrigeração capaz de atender aos requisitos do processo e manter um custo razoável. Um sistema de HVAC (*heating, ventilation and air conditioning*) é extremamente necessário no setor farmacêutico visto que este precisa garantir os requisitos exigidos pelos órgãos regulamentadores. Sabendo disso, é necessário haver bombas, válvulas, atuadores e sensores que serão responsáveis por monitorar as variáveis e parâmetros do processo. Em uma indústria farmacêutica do norte de Minas Gerais, muitas vezes percebe-se uma queda no desempenho das válvulas ao final de seu ciclo de vida e, para garantir a robustez do processo e evitar atrasos e prejuízos para a indústria é preciso perceber essas falhas o mais rápido possível para agir sobre o problema. A utilização de inteligência artificial pode ser uma grande aliada nesse contexto, pois são capazes de analisar variáveis e encontrar correlações que poderão auxiliar na identificação de falhas. O presente trabalho tem como objetivo utilizar técnicas de aprendizado de máquina, como regressão logística, árvores de decisão e alguns métodos como TabNet, XGBoost e redes neurais recorrentes para correlacionar as variáveis de monitoramento e de controle em busca de identificar e/ou predizer possíveis defeitos em válvulas, assim contribuindo para que as falhas sejam detectadas mais rapidamente e o tempo de resposta da manutenção ou substituição diminuído. Acredita-se que o objetivo foi cumprido com sucesso, visto que todos os métodos alcançaram excelentes desempenhos, sendo possível obter acurácias altíssimas variando entre 95% e 99%.

Palavras-chaves: Inteligência Artificial, Aprendizado de máquina, Rede Neural, Válvulas, Predição, HVAC, Acurácia.

Abstract

With the great increase in industries and the advancement of technologies, it became necessary to obtain a good refrigeration system capable of meeting the process requirements and maintaining a reasonable cost. An HVAC (heating, ventilation and air conditioning) system is extremely necessary in the pharmaceutical sector as it needs to guarantee the requirements demanded by regulatory bodies. Knowing this, it is necessary to have pumps, valves, actuators and sensors that will be responsible for monitoring the variables and parameters of the process. In a pharmaceutical industry in the north of Minas Gerais, a lot of times it's noticed a drop in the performance of valves at the end of the valve's life cycle and, to ensure the robustness of the process and avoid delays and losses for the industry, it is necessary to notice these failures as soon as possible to be able to act on the problem. The use of artificial intelligence can be a great ally in this context, as they are able to analyze variables and find correlations that can help to identify failures. The present work aims to use machine learning techniques such as logistic regression, decision trees and some methods such as TabNet, XGBoost and recurrent neural networks to correlate monitoring and control variables in order to identify and/or predict possible defects in valves, and so contributing to detect failures more quickly allowing the decreasement of response time of maintenance or replacement. It is believed that the objective was successfully achieved, since all methods accomplished excellent performance, being possible to obtain very high accuracies ranging between 95% and 99%.

Key-words: Artificial Intelligence, Machine Learning, Neural Network, Valves, Prediction, HVAC, Accuracy.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

HVAC	Sistema de aquecimento, ventilação e ar condicionado (*heating, ventilation and air conditioning*)
ANVISA	Agência Nacional de Vigilância Sanitária
PID	Proporcional, Integral e Derivativo
IA	Inteligência Artificial
GPU	Unidades de Processamento Gráficos (*Graphics Processing Unit*)
MPC	Modelo de controle preditivo (*Model Predictive Control*)
BNMI	Melhor Rede após Múltiplas Iterações (*Best Network after Multiple Iterations*)
MOGA	Algoritmos genéticos multi-objetivos (*Multi-objective genetic algorithm*)
FDD	Detecção e diagnóstico de falhas (*Fault detection and diagnosis*)
CNN	Redes Neurais Convolucionais (*Convolutional Neural Networks*)
RNN	Redes Neurais Recorrentes (*Recurrent Neural Network*)
ADAM	Estimativa de Momento Adaptativo (*Adaptive Moment Estimation*)
LSTM	Memória de longo e curto prazo (*Long Short-Term Memory*)
NAS	Busca automática de arquiteturas de redes neurais (*Neural Architecture Search*)
∞	Infinito

Sumário

Parte I

Preparação da pesquisa

1 Introdução

1.1 Introdução ao problema

Com o crescimento destacado das grandes organizações e o aumento considerável do consumo de energia, obter um bom sistema de refrigeração tornou-se essencial para as grandes manufaturas. Este sistema, também conhecido como sistema de aquecimento, ventilação e ar condicionado (HVAC – do inglês *heating, ventilation and air conditioning*) é o responsável por deixar o ambiente interno em condições climáticas agradáveis e apropriadas para o processo. Ao se tratar de indústrias do setor farmacêutico, o sistema de HVAC é extremamente necessário para garantir que os requisitos exigidos pelos órgãos regulamentadores, no caso do Brasil a Agência Nacional de Vigilância Sanitária - ANVISA, serão cumpridos.

Segundo Handbook (1996) os sistemas de HVAC são categorizados pelo método utilizado para controlar o aquecimento, a ventilação e o ar condicionado. Sabendo disso, é necessário que o engenheiro de *design* encontre o método que atuará melhor em cima dos critérios requisitados pelo processo. Os parâmetros considerados podem ser temperatura, umidade, pureza ou qualidade do ar, clima local, requisitos de velocidade do ar, custo operacional, custo de manutenção, entre outros.

As válvulas utilizadas em um sistema de HVAC podem ser dos mais diversos tipos, no entanto, sabe-se que determinadas válvulas possuem um comportamento diferente do padrão quando estão se aproximando ao final do seu ciclo de vida, o que pode acarretar em alterações no funcionamento do sistema e por consequência alterações nos parâmetros requisitados.

Em uma indústria farmacêutica do Norte de Minas Gerais - Brasil, o sistema de refrigeração é composto por diversos equipamentos que são responsáveis pela climatização interna de cada sala. Para que seja possível realizar a climatização, o sistema possui um condicionador de ar do tipo *FanCoil* que funciona como uma caixa de alumínio que possui em seu interior serpentinas capazes de aquecer, resfriar e desumidificar o ar. Em conjunto com o condicionador de ar, se faz necessária a utilização de bombas, válvulas e atuadores que serão responsáveis por promover o fluxo de água necessário.

Além disso, para permitir o acompanhamento das variáveis que se deseja controlar, são utilizados sensores de umidade, de temperatura, de pressão e sensores contadores de partículas. Com o sistema de HVAC e os sensores distribuídos pela indústria, é possível garantir que os requisitos necessários para o produto final e para a segurança dos colaboradores estão sendo cumpridos.

No entanto, por ser uma indústria com processo bem extenso, o sistema de monitoramento das variáveis não é específico para cada sala, e sim para áreas maiores que foram subdivididas de acordo com a necessidade do processo. Sabe-se também que o processo sofre influência de vários fatores externos como a temperatura ambiente local ou fatores internos como a ação de abrir e fechar portas, que permite uma maior movimentação do ar.

Tendo em vista os fatos supracitados, podem-se haver situações em que as variáveis monitoradas e os controles proporcional, integral e derivativos (PID) aplicados à malha camuflarão o mau funcionamento da válvula e, por isso, o processo levará um tempo maior para perceber as alterações provenientes dela.

Contudo, ainda que não seja refletido nas variáveis monitoradas de uma área maior, uma válvula quebrada ou com seu funcionamento comprometido, pode alterar as condições climáticas de uma sala em específico, o que pode vir a acarretar o descarte de tudo o que está sendo produzido/armazenado neste ambiente, gerando prejuízos e atrasos para a indústria. Vale ressaltar que o funcionamento dessa indústria é contínuo durante todos os dias do ano e que todos as variáveis monitoradas podem ser facilmente acessadas em um banco de dados. As informações de manutenção e falhas passadas também podem ser obtidas sem maiores esforços.

De acordo com Ruschel, Santos e Loures (2017) a eficácia das manutenções está diretamente relacionada ao bom planejamento de intervalo de intervenções e muitas vezes essa tomada de decisão é realizada com base em critérios inadequados. O uso de inteligência artificial pode ser um grande aliado nesses momentos.

Inteligência artificial (IA), em Nilsson (2009) é considerado como a atividade dedicada a tornar as máquinas inteligentes. E ao dizer máquinas inteligentes, o autor refere-se a máquinas que são capazes de funcionar de maneira adequada e otimizada, dentro das condições e particularidades de seu ambiente.

Segundo Goodfellow, Bengio e Courville (2016), a IA é capaz de resolver rapidamente problemas que são difíceis para os seres humanos mas que podem facilmente ser descritos por passo a passos ou regras matemáticas. Em contrapartida, tarefas simples que podem ser realizadas intuitivamente pelos seres humanos, mas que dificilmente são descritas formalmente, podem ser executadas por máquinas de forma otimizada por meio das redes neurais profundas.

O presente trabalho tem como objetivo utilizar técnicas de inteligência artificial, mais especificamente redes neurais profundas, recorrentes para correlacionar as variáveis de monitoramento e de controle em busca de identificar e/ou predizer possíveis defeitos em válvulas, assim contribuindo para que as falhas sejam detectadas mais rapidamente e o tempo de resposta da manutenção ou substituição diminuído.

1.2 Objetivos

Nessa seção serão apresentados os objetivos do presente trabalho.

1.2.1 Objetivo Geral

Aplicar técnicas de aprendizado de máquina para correlacionar variáveis de monitoramento e de controle com a finalidade de predizer falhas em válvulas de um sistema de refrigeração e analisar o desempenho de cada método aplicado.

1.2.2 Objetivos específicos

1. Extrair e tratar os dados.
2. Classificar a base de dados.
3. Correlacionar as variáveis.
4. Testar e comparar métodos de redes neurais profundas recorrentes e para dados tabulares.
5. Analisar o desempenho dos métodos aplicados.
6. Definir o método mais eficiente para o problema, dentre os analisados.

1.3 Organização e estrutura

O objetivo desse trabalho é aplicar técnicas de aprendizado de máquina, mais especificamente métodos baseados em redes neurais para correlacionar variáveis. Visando esse objetivo, esse texto está organizado como se segue: no segundo Capítulo apresenta-se uma revisão da literatura pertinente ao tema do trabalho. O desenvolvimento do trabalho é descrito no Capítulo 3. Os resultados são apresentados e são assunto do Capítulo 4. Por fim, no Capítulo 5 é apresentado a conclusão, com as considerações finais e sugestões para trabalhos futuros.

Parte II

Referencial teórico

2 Revisão da literatura

Ao longo dessa seção serão explorados conceitos, algoritmos e métodos essenciais para a elaboração do presente trabalho visando uma maior compreensão do assunto.

2.1 Trabalhos relacionados

Sistemas de HVAC são sistemas complexos e não lineares devido à quantidade de interações entre os sub-sistemas que o compõe, como ar condicionados, serpentina, dutos, tubulação e etc. Por isso, segundo Afram et al. (2017), é essencial investigar métodos de reduzir o custo operacional desses sistemas.

Durante o trabalho, Afram et al. (2017), utilizam redes neurais artificias baseadas em um modelo de controle preditivo (MPC - do inglês *Model Predictive Control*) para otimizar o sistema de HVAC de uma casa localizada em Ontario no Canadá. A proposta dos autores é utilizar um novo algoritmo, chamado Melhor Rede após Múltiplas Iterações (BNMI - do inglês *Best Network after Multiple Iterations*), para predizer o total de energia consumida pelo sistema de HVAC. Esse algoritmo tem como objetivo determinar a rede neural apropriada para esse tipo de problema. Após o desenvolvimento da rede neural os autores se propuseram a implementar um sistema supervisório para monitoramento do sistema e da rede criada. Os resultados foram promissores, o desempenho da predição utilizando a BNMI variou entre 6% e 59% melhor do que vários outros modelos. Além disso, o modelo preditivo foi capaz de salvar entre 6% e 73% do custo operacional do sistema de HVAC, a depender da estação do ano.

Outro trabalho relacionado, intitulado *"Optimization of HVAC system energy consumption in a building using artificial neural network and multi-objective genetic algorithm"* e desenvolvido por Satrio et al. (2019), utilizou uma combinação entre redes neurais artificiais e algoritmos genéticos multi-objetivos (MOGA - do inglês *Multi-objective genetic algorithm*) para otimizar a operação de um sistema HVAC de um prédio. Os autores tinham como objetivo minimizar o consumo de energia anual e maximizar o conforto térmico, visto que o percentual das pessoas insatisfeitas com esses dois parâmetros estava alto. Para otimizar o sistema de HVAC foram feitos vários testes utilizando MOGA, no entanto, a otimização que considerava dois objetivos mostrou o melhor resultado na diminuição do custo e no aumento do conforto térmico do edifício.

2.2 Inteligência artificial na manutenção

Com a urgência da indústria 4.0, a utilização de inteligência artificial passa por uma constante crescente em diversos aspectos nas grandes empresas, incluindo a utilização em manutenções.

Segundo Lee et al. (2019), manutenções de equipamentos são frequentemente realizadas sem uma metodologia de planejamento prévio. Isso pode fazer com que a disponibilidade dos equipamentos caia, ou seja, os equipamentos podem ficar mais tempo fora de uso devido a falhas repentinas e não esperadas. Com isso, tem-se dado maior atenção às estratégias de manutenção preditiva alcançadas com o uso de inteligência artificial.

O estudo de Lee et al. (2019) é um caso de sucesso, onde se utilizou redes neurais para monitorar duas ferramentas críticas para o processo apresentado, sendo elas uma ferramenta de corte e um motor de fuso. Para estudar a capacidade dos algoritmos de IA na construção de um modelo de manutenção preditiva, foram acoplados sensores nos equipamentos monitorados para coletar dados em tempo real. Os experimentos realizados, foram executados até o momento de uma falha. Após a fase de treino, os dados coletados durante as falhas foram utilizados para testar o modelo de detecção de falhas. Os autores conseguiram uma acurácia entre 84% e 98%, concluindo que a utilização de manutenções preditivas baseadas em modelos de redes neurais artificiais é capaz de reduzir o tempo de inatividade das máquinas assim reduzindo significantemente o custo de manutenção dos equipamentos.

Em Zhao et al. (2019), foi realizada uma revisão da literatura sobre detecção e diagnóstico de falhas (FDD - do inglês *Fault detection and diagnosis*) baseadas em inteligência artificial para sistemas de energia, resumindo os pontos fortes e os pontos fracos dos métodos já existentes para construção de sistemas de energia dos últimos vinte anos (entre 1998 e 2018). Durante o trabalho, os autores ressaltam que nas últimas décadas uma grande quantidade de métodos FDD foi desenvolvida para sistemas de energia, visando otimizar sistemas de unidade de tratamento de ar, terminais de volume de ar chegando até em nível de sistemas completos de HVAC.

De acordo com os autores em Zhao et al. (2019), o método FDD é composto por duas etapas: a detecção de falhas e o diagnóstico da falha. A primeira permite detectar a ocorrência de falhas enquanto a segunda permite identificar o tipo e a localização da falha. Alguns métodos podem lidar com as duas etapas simultaneamente como é o caso de métodos onde se têm uma base de dados extensas. Durante o estudo, foram analisados vários trabalhos que contemplavam assuntos sobre métodos FDD baseados em inteligência artificial, e foi percebido que a maior parte utilizava métodos de redes neurais não supervisionadas e métodos baseados em classificação. Na Figura 1 pode ser visto um gráfico com a classificação dos estudos utilizados para embasar o trabalho de Zhao et al. (2019).

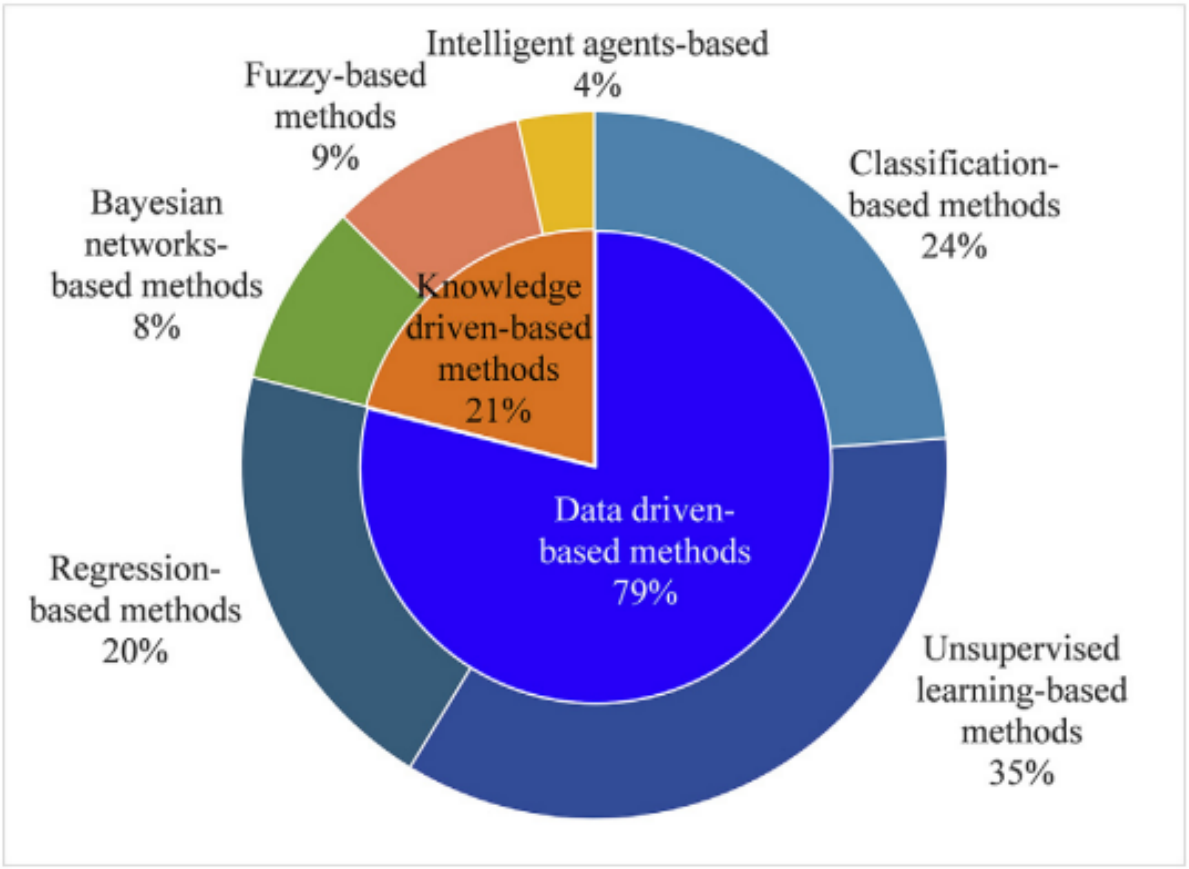


Figura 1 – Classificação dos trabalhos utilizados para realizar a revisão da literatura sobre FDD.

Fonte: Zhao et al. (2019).

2.3 Série Temporal

Segundo Esling e Agon (2012), em quase todos os campos científicos as medições são realizadas ao longo do tempo. Essas medições compõem uma coleção de dados organizados e estruturados que chamamos de séries temporais. Muitas vezes nós humanos temos facilidade em interpretar e extrair conhecimento de séries temporais, no entanto, essa tarefa continua sendo complexa para máquinas e computadores.

Ao se trabalhar com séries temporais, torna-se necessário realizar um processo de análise dessas séries, para isso, analisei modelos estacionários e não estacionários, modelos sazonais e não sazonais, verificamos tendências, resíduos e padrões (WEI, 2013).

2.4 Overfitting

Overfitting, que pode ser observado na Figura 2 de Madiraju, Nischal (2022), é o fenômeno que ocorre quando o modelo se ajusta perfeitamente ao conjunto de treino mas se mostra ineficaz para prever novos resultados.

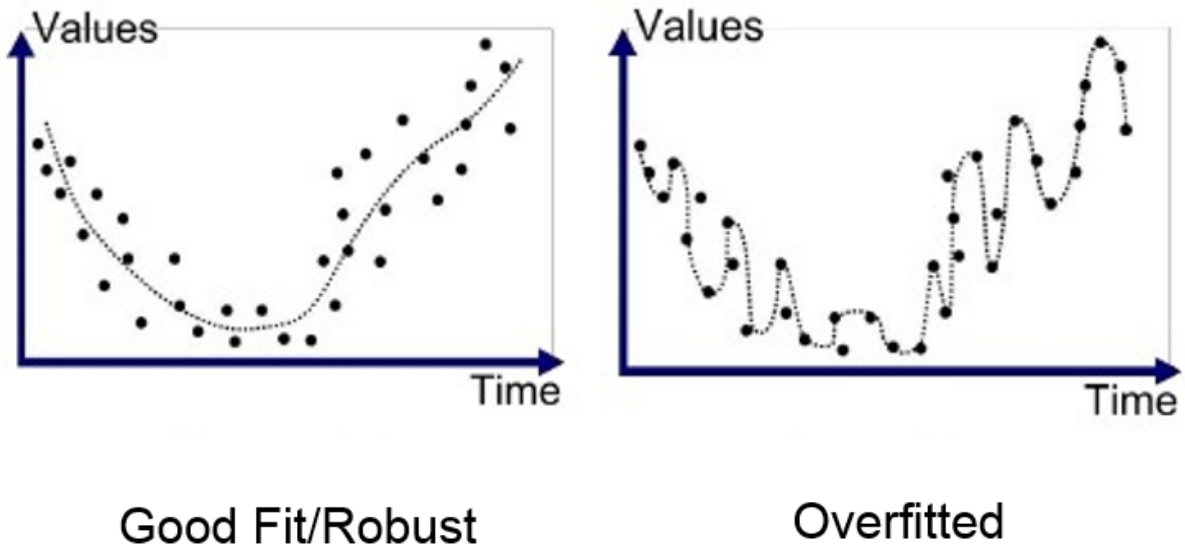


Figura 2 – Diferença entre um modelo robusto e um *overfitting*.
Fonte: Madiraju, Nischal (2022).

De acordo com Dietterich (1995) o objetivo de uma tarefa de aprendizado de máquina é maximizar a acurácia preditiva nos novos dados, no entanto, se se trabalhar focando-se em encontrar o melhor ajuste aos dados de treinos, existirá o risco de ajustar também os ruídos desses dados ao memorizar várias peculiaridades deles em vez de se encontrar uma regra geral de previsão que será capaz de ajustar bem aos novos dados e assim conseguir predizer acertadamente novos resultados.

2.5 Regressão Logística

Regressão logística é um dos modelos estatísticos mais utilizados em pesquisas no ramo de saúde, estatística social, ecologia e afins. No entanto, segundo Hilbe (2011), esta análise tem sido considerada por muitos cientistas de dados um procedimento importante para a análise preditiva.

Diferente da regressão linear, a regressão logística foi desenvolvida para modelar variáveis binárias. Esse é considerado por Kleinbaum et al. (2002) um dos motivos de sua popularidade. Segundo ele, em seu livro "*Logistic Regression - A Self-Learning Text*", esse modelo logístico foi criado para garantir que qualquer que seja a estimativa e probabilidade que se busca, o valor sempre será algum número entre 0 e 1.

Outro ponto positivo da regressão logística é a forma de sua função. No gráfico representado na Figura 3 pode-se ver que se z começar em z= -∞e caminhar à direta, à medida que z aumenta, o valor da f(z) se mantém perto de zero por um momento até que começa a aumentar drasticamente em direção de 1 até se estabelecer muito próximo

de 1 à medida que z caminha para z = $+\infty$. Ainda de acordo com Kleinbaum et al. (2002) o resultado desse movimento é uma curva no formato de um S alongado que é amplamente aplicável para considerar a natureza multivariável de um problema de riscos e probabilidades.

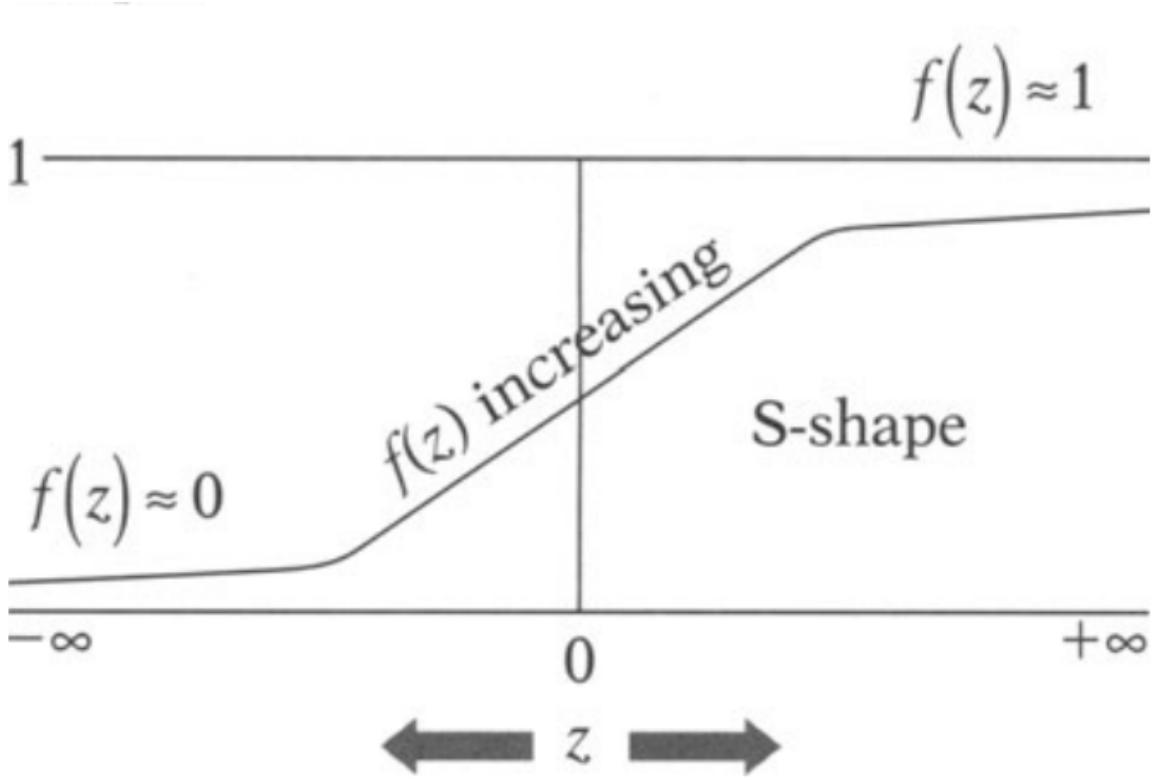


Figura 3 – Formato de uma função de regressão logística.

Fonte: Kleinbaum et al. (2002)

2.6 Rede neural

A brilhante capacidade do cérebro humano em resolver problemas por meio dos bilhões de neurônios interconectados foi a responsável por inspirar o estudos relacionados Às redes neurais artificiais.

Segundo Wang (2003), uma rede neural consiste em uma camada de entrada, camadas ”escondidas”de neurônios e uma camada final de saída. Uma arquitetura de rede neural pode ser vista na Figura 4, onde pode ser observado as camadas, os neurônios e suas conexões.

A saída da rede neural depende de alguns fatores, como o peso associado a cada conexão, a função de ativação escolhida, entre outros. O objetivo da função de ativação é introduzir uma não linearidade para a rede neural, além de evitar que ela seja paralisada por divergência nos neurônios (WANG, 2003).

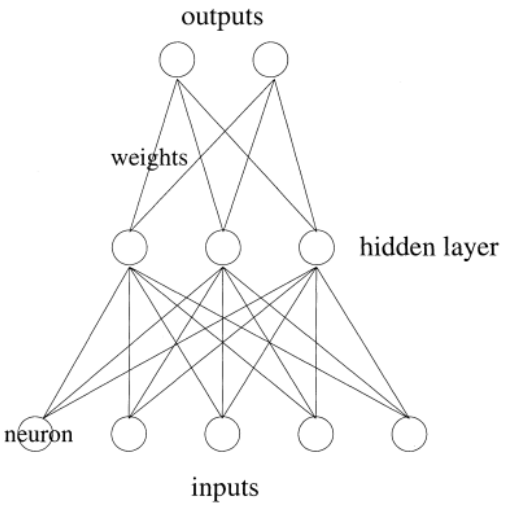


Figura 4 – Arquitetura básica de uma rede neural.

Fonte: Wang (2003).

2.6.1 Redes Neurais Recorrentes

Ainda que as redes neurais convolucionais (CNN - do inglês *Convolutional Neural Network* sejam capazes de processar uma enorme quantidade de dados de maneira eficiente, muitas vezes é preciso perceber que os dados não são distribuídos de forma independente e idêntica e que sua ordem de distribuição deve ser mantida. Visto isso, sabe-se que as redes neurais recorrentes (RNN - do inglês *Recurrent Neural Network*) são projetadas para lidar melhor com informações sequenciais. Zhang et al. (2021).

De acordo com Mikolov et al. (2010) as RNN, exemplificadas na Figura 5, são representadas por neurônios com conexões recorrentes e possuem, geralmente, uma camada de entrada, uma camada oculta, também chamada de estado, e uma camada de saída. Além disso, esse método é apto a adquirir uma memória de curto prazo capaz de melhorar o desempenho do modelo em momentos de invariância dos valores e, por isso, pretende-se utilizar esse método.

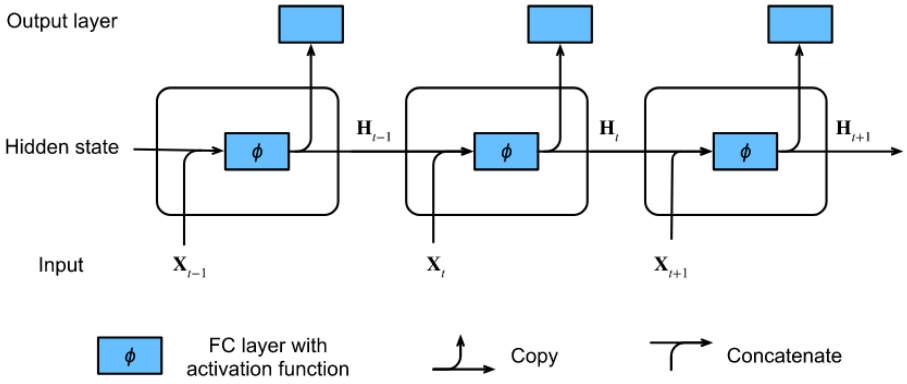


Figura 5 – Exemplo de uma rede neural recorrente.

Fonte: Zhang et al. (2021)

2.6.2 Redes Neurais para Dados Tabulares

A evolução dos recursos computacionais e a necessidade de se compreender e analisar base de dados cada vez maiores fazem com que estudos acerca de redes neurais profundas tenham cada vez mais sucesso. Redes neurais convolucionais e recorrentes são capazes de alcançar um desempenho extraordinário em tarefas de classificação com dados homogêneos e organizados, no entanto, segundo Borisov et al. (2021) dados tabulares ainda representam um grande desafio para modelos de aprendizado de máquina.

Em seu estudo, Xu et al. (2019) descreve dados tabulares como geralmente contendo uma mistura de colunas discretas e contínuas. As contínuas podem serm facilmente analisadas, enquanto as discretas às vezes são desequilibradas dificultando a modelagem. Com isso, os modelos de estatística e de redes neurais profundas podem falhar em analisar e modelar adequadamente esse tipo de dado.

Embora dado tabular seja uma das formas mais antigas de dado a ser analisado e que, antigamente, antes da era digital, praticamente todos os dados estivessem nesse formato, ainda não se sabe ao certo por que o aprendizado profundo de máquinas pode não atingir o mesmo nível de qualidade preditiva em dados tabulares se comparado a tarefas de classificação de imagem ou processamento de linguagem. Borisov et al. (2021), em seu trabalho "*Deep Neural Nerworks and Tabular Data: A Survey*", aponta e discute quatro possíveis razões para isso: Baixa qualidade nos dados de treinamento, dependências espaciais complexas ou irregulares, pré-processamento ineficaz e a falta de análise de algumas características únicas.

2.7 Árvores de decisão

Árvores de decisão são métodos de aprendizado supervisionado de máquina para construir modelos que prevejam o valor de uma variável desejada aprendendo regras inferidas a

partir de uma base de dados (SCIKIT-LEARN, 2022). Segundo Loh (2011) os modelos são obtidos particionando recursivamente o espaço de dados e aplicando um modelo de previsão simples dentro de cada uma das partições.

A utilização de árvores de decisões apresenta várias vantagens além de ser simples de entender e de interpretar. Uma das vantagens é necessitar de pouca preparação de dados. De acordo com Scikit-learn (2022), outras técnicas precisam de uma etapa de análise de dados extensa, incluindo normalização de dados, tratamento dos valores em branco e outros. Além disso, é capaz de lidar com dados numéricos e categóricos e é capaz de performar bem mesmo que algumas informações ou suposições estejam um pouco confusas.

Por outro lado, é preciso ter cuidado ao criar uma árvore de decisão, para que ela não seja supercomplexa, de forma a não ser capaz de generalizar bem os dados e assim trazer uma resposta fidedigna ao modelo.

Árvores de decisões podem ser dividas entre árvores de classificação e árvores de regressão, neste trabalho utilizaremos a primeira.

2.7.1 Classificador Baseado em Árvore de Decisão

Em um problema de classificação, a principal função do algoritmo é aprender um modelo que cria rótulos para os dados oferecidos e consegue classificar bem, a partir desses rótulos, os dados não vistos ou não obtidos. Assim, o algoritmo de classificação tem como objetivo descobrir a relação entre os atributos de entrada e os atributos de saída para construir um modelo de treinamento. Sabendo disso, segundo Charbuty e Abdulazeez (2021), se a quantidade de dados é suficiente, o conjunto foi devidamente tratado e classificado e contém o número mínimo de nós, pode-se considerar uma ótima opção a utilização de uma árvore de decisão para problemas de classificação.

Nos classificadores baseados em árvores de decisões cada árvore é composta por ramos e nós, onde cada nó representa característica em uma categoria a ser classificada e cada subconjunto define um valor que pode ser extraído de cada nó. Um exemplo de árvore de decisão pode ser observado na Figura 6.

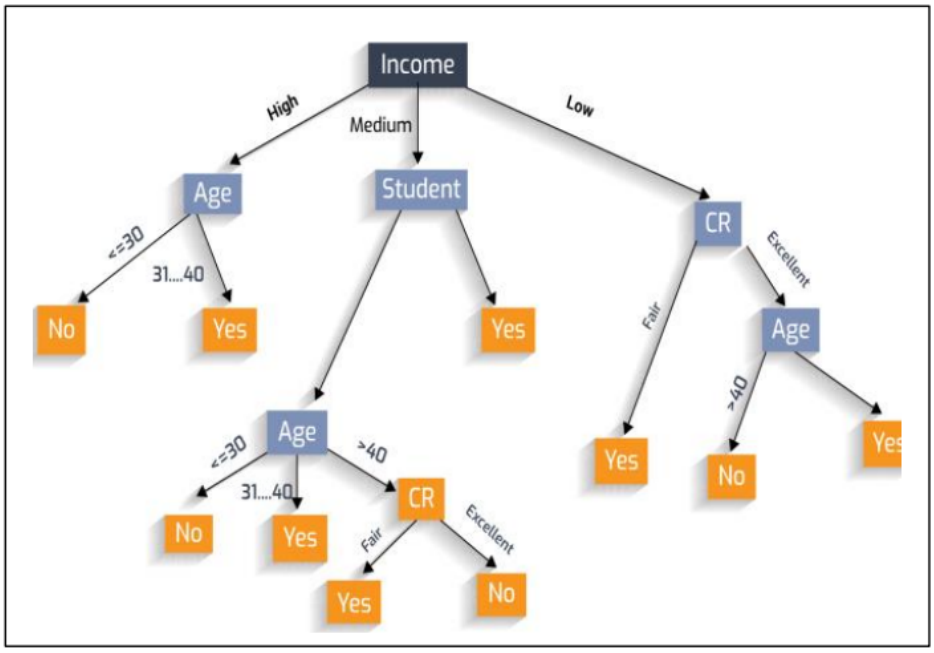


Figura 6 – Exemplo de árvore de decisão.

Fonte: Charbuty e Abdulazeez (2021)

Segundo Loh (2011) as vantagens de se usar árvores de decisão para classificação, quando comparado, por exemplo, a um gráfico, é que estruturas em árvore têm aplicabilidade a qualquer número de variáveis, enquanto outros métodos podem estar limitados a pequenas quantidades de variáveis. Na Figura 7 pode-se observar um exemplo de modelo de classificação com três classes rotuladas (como 1, 2 e 3), e a cada nó intermediário, parte-se para o próximo nó à esquerda se e somente se a condição for completamente satisfeita.

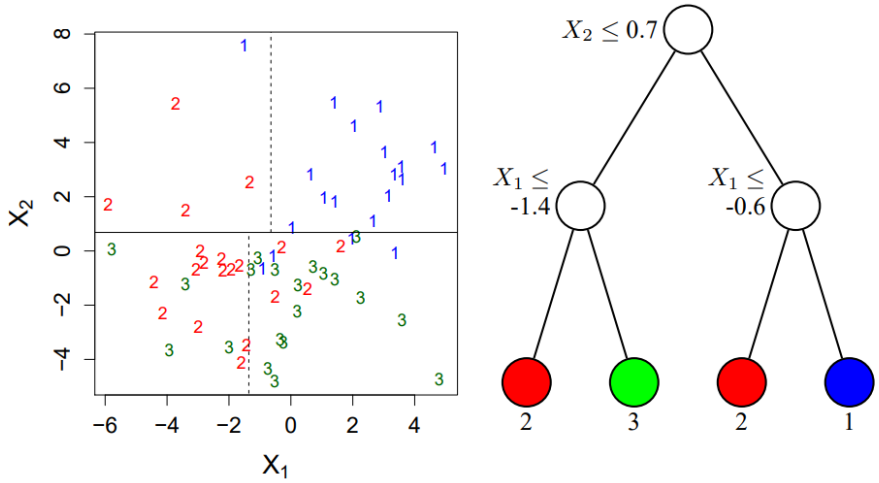


Figura 7 – Exemplo de árvore de decisão. À esquerda gráfico de partições e à direita estrutura de árvore de decisão.

Fonte: Loh (2011)

2.8 TabNet

TabNet é uma arquitetura de aprendizado de dados tabulares profundos de alto desempenho que usa atenção sequencial para escolher quais características serão utilizadas para tomada de decisão em cada etapa de decisão. Essa arquitetura foi desenvolvida por colaboradores da *Google Cloud AI*.

Segundo os desenvolvedores, TabNet insere dados tabulares brutos sem qualquer tipo de processamento prévio e os treina utilizando otimização baseada em gradiente descendente (ARIK; PFISTER, 2021).

Essa arquitetura possui a chamada seleção de recursos por instância, ou seja, a seleção pode ser diferente para cada entrada. Isso faz com que a TabNet supere outros modelos de aprendizado tabular com vários conjuntos de dados para problemas tanto de classificação quanto de regressão.

Outro fator determinante para o sucesso desse método, segundo Arik e Pfister (2021), é que este permite dois tipos de interpretabilidade. A local e a global, onde a primeira visualiza a importância dos recursos e como eles são combinados e a segunda quantifica a contribuição que cada um desses recursos emprega para o modelo treinado.

O *encoder* TabNet é composto por um *Transformer* de características, uma arquitetura *Transformer* com auto-atenção e um mascaramento de recursos. A Tabnet é responsável por, pela primeira vez para dados tabulares, mostrar melhorias significativas no desempenho usando pré-treinamento para prever recursos mascarados. Para cada etapa do modelo, a máscara de seleção de características fornece informações sobre a funcionalidade do modelo, e essas máscaras podem ser agregadas para obter informações importantes das características globais.

Falando um pouco sobre a arquitetura do modelo, o bloco transformador de recursos é constituído por uma rede de 4 camadas, onde duas são compartilhadas em todas as etapas de decisão e duas são dependentes de cada etapa.

Ainda segundo Arik e Pfister (2021), a capacidade do modelo é toda utilizada para as características mais salientes e é capaz de desempenhar uma tomada de decisão mais interpretável por meio da visualização das máscaras de seleção. No trabalho eles demonstram que TabNet supera trabalhos anteriores conhecidos, como as arquiteturas XGBoost, LightGBM e outras, demonstrando benefícios significativos para uma rápida adaptação e melhor desempenho em dados tabulares.

2.9 XGBoost

XGBoost, do inglês *Extreme Gradient Boosting* é uma biblioteca de aprendizado de máquina de árvores de decisão escaláveis e altamente precisas de aumento de gradiente. De acordo com NVIDIA Developers (2022) é a principal biblioteca de aprendizado de máquina para problemas de regressão, classificação e ranqueamento.

A XGBoost, diferente de outras arquiteturas, constrói árvores em paralelo e segue uma estratégia de nível, onde os valores do gradiente são varridos e os resultados parciais são utilizados para avaliar a qualidade das divisões ou classificações em cada etapa do conjunto de treinamento.

Essa biblioteca foi desenvolvida na Universidade de Washington nos Estados Unidos da América. Os autores demonstram que o sistema fornece resultados de última geração em uma ampla gama de problemas. Acredita-se que o fator mais importante por trás do sucesso do XGBoost seja devido a sua alta escalabilidade em todos os cenários, além de que, ainda segundo os desenvolvedores, o sistema é capaz de rodas mais de dez vezes mais rápido do que outras soluções populares existentes (CHEN; GUESTRIN, 2016).

Vale ressaltar que já existem alguns trabalhos de reforço de árvores paralelas, no entanto, a XGBoost foi a primeira a explorar temas como computação fora do núcleo, reconhecimento de cache e reconhecimento de esparsidade. Combinar todos esses aspectos em um sistema de ponta a ponta é oferecer uma solução inovadora para casos reais, onde cientistas de dados e pesquisadores são capazes de construir poderosos algoritmos que impulsionem árvores de decisão.

Outro ponto importante é que como XGBoost é um código aberto, uma enorme lista de cientistas de dados de todo o mundo estão ativamente contribuindo para a otimização e aprimoramento dessa biblioteca.

No trabalho, Chen e Guestrin (2016) comparam o algoritmo ganancioso de XGBoost com duas outras implementações de impulso de árvore gananciosa. Uma das duas implementações utiliza uma abordagem de método guloso que expande apenas um galho da árvore, o que torna o método mais rápido mas pode apresentar uma menor precisão. A outra utiliza uma abordagem de aprendizado que utiliza a árvore inteira assim como a XGBoost. Como resultado percebeu-se que a XGBoost presenta um desempenho mais alto que o método que expande um galho só, e desempenho similar a outra implementação, no entanto a XGBoost sai na frente quando se tratando de velocidade de processamento.

Parte III

Desenvolvimento

3 Desenvolvimento

3.1 Linguagem de programação e ambiente de execução

O trabalho foi desenvolvido e codificado utilizando a linguagem de programação *Python* na versão 3.7.13. *Python* é uma poderosa linguagem de programação, fácil de ser utilizada e que possui eficientes estruturas de dados de alto-nível (VAN ROSSUM; DRAKE JR, 1995). Assim como outros tipos de linguagem de programação emergentes, *Python* está em constante ascensão e com isso o número de usuários está cada vez maior. A disseminação do conhecimento de linguagens de programação faz com que os tópicos de aprendizagem de máquinas, ciência de dados e inteligência artificial esteja cada vez mais presente na vida das pessoas, auxiliando na solução de problemas complexos.

O ambiente de execução utilizado para o desenvolvimento desse projeto foi o *Google Colaboratory*, exemplificado na Figura 8 que permite aos usuários codificarem em *Python* por meio do próprio navegador, utilizando unidades de processamento gráficos (GPU – do inglês *Graphics Processing Unit*) gratuitas da *Google* e com alta facilidade de compartilhamento. O *Google Colaboratory*, também conhecido como *Colab*, é uma estrutura online onde podem ser escritos e executados códigos de aprendizagem de máquinas e redes neurais complexas (KANANI; PADOLE, 2019).

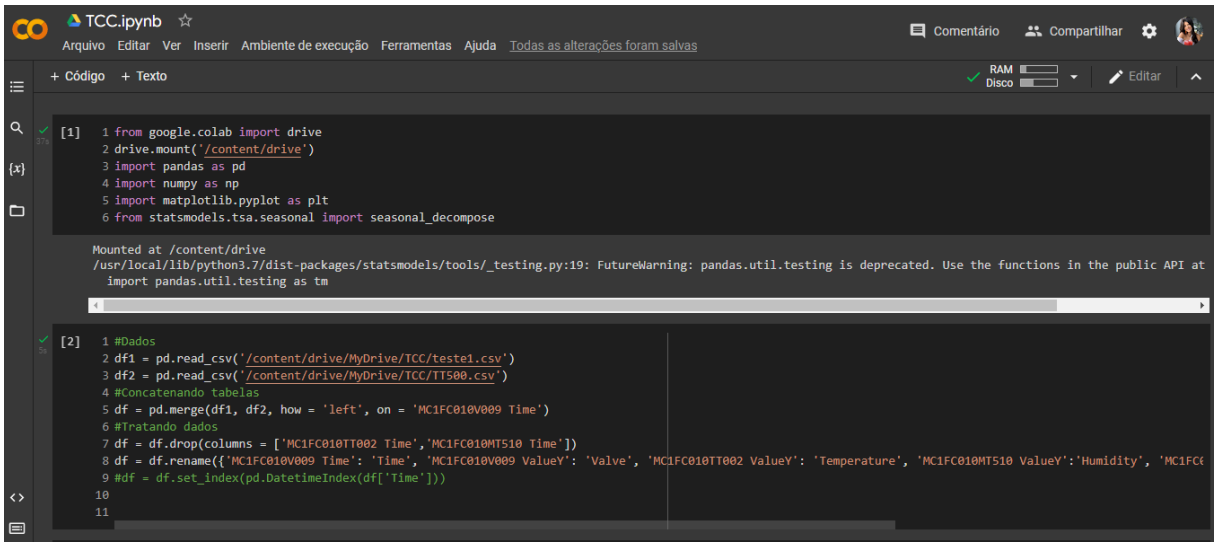


Figura 8 – Ambiente de execução *Google Colaboratory*.

Fonte: Elaborado pela autora.

3.2 Base de dados

A base de dados é o principal material utilizado para a construção de modelos de aprendizagem de máquinas. A base utilizada no presente trabalho foi extraída de uma empresa do norte de Minas Gerais do setor farmacêutico e para a obtenção dos dados foi necessário a extração e o armazenamento dos valores de cada uma das variáveis desejadas.

O sistema de HVAC dessa empresa possui armazenamento de dados de medição das variáveis de processo de todos os equipamentos e subsistemas. Para o presente trabalho foi utilizado apenas os dados de um dos subsistemas, aqui chamado de FT010, no período de janeiro de 2020 a maio de 2021.

A base de dados escolhida para esse projeto é composta de uma coluna com os tempos de coleta, aqui chamada de "*Time*"e armazenada a cada um minuto, uma coluna correspondente ao sinal de controle da válvula, aqui chamado de "*Valve*", que diz respeito ao sinal que deve ser aplicado ao processo em determinado momento visando corrigir o desvio do sinal de saída em relação ao *setpoint* do processo. As colunas "*Temperature*"e "*Back_Temperature*"são referentes às temperatura de insuflamento e de retorno, respectivamente. As temperaturas são medidas por meio de sensores de temperatura localizados na tubulação do sistema. O sensor de temperatura de insuflamento está posicionado na tubulação de saída do sistema de trocador de calor enquanto o sensor de temperatura de retorno está localizado na tubulação de realimentação do sistema. Por fim, tem-se uma coluna referente à umidade do sistema que é medida por meio de um transmissor de umidade disposto próximo à tubulação de realimentação do sistema. Uma imagem da representação da base de dados e uma imagem para ilustrar o processo podem ser observadas nas Figuras 9 e 10 respectivamente.

É importante conhecer um pouco da arquitetura do sistema, para se entender como ocorreu a obtenção dos dados utilizados nesse projeto. O sistema é composto por um Controlador Lógico Programável (CLP) que recebe como entradas analógicas os sinais vindos dos sensores de temperatura e umidade. Já as saídas analógicas do CLP vão para as válvulas que compõem o sistema de HVAC. O CLP envia as variáveis para o Sistema de Supervisão e Aquisição de Dados (SCADA) que, por sua vez, armazena essa informação em um banco de dados. Esses dados são armazenados a cada um segundo, durante todos os dias do ano.

	Time	Valve	Temperature	Humidity	Back_Temperature
0	2020-01-28 15:22:44	35.026661	17.203775	58.807144	21.732132
1	2020-01-28 15:23:44	34.454697	17.187500	58.684170	21.701389
2	2020-01-28 15:24:44	34.304714	17.171225	58.640770	21.717665
3	2020-01-28 15:25:44	34.392952	17.203775	58.579281	21.750217
4	2020-01-28 15:26:44	33.891094	17.234520	58.557579	21.733940
...	...	...	...	...	...
699748	2021-05-28 15:18:55	27.454987	19.641565	52.003761	21.685112
699749	2021-05-28 15:19:55	26.857983	19.769964	51.978443	21.668837
699750	2021-05-28 15:20:55	27.056767	19.911024	51.953125	21.717665
699751	2021-05-28 15:21:55	26.688835	20.043041	51.916954	21.715857
699752	2021-05-28 15:22:55	26.352100	20.185907	51.945889	21.717665

Figura 9 – Base de Dados.

Fonte: Elaborado pela autora.

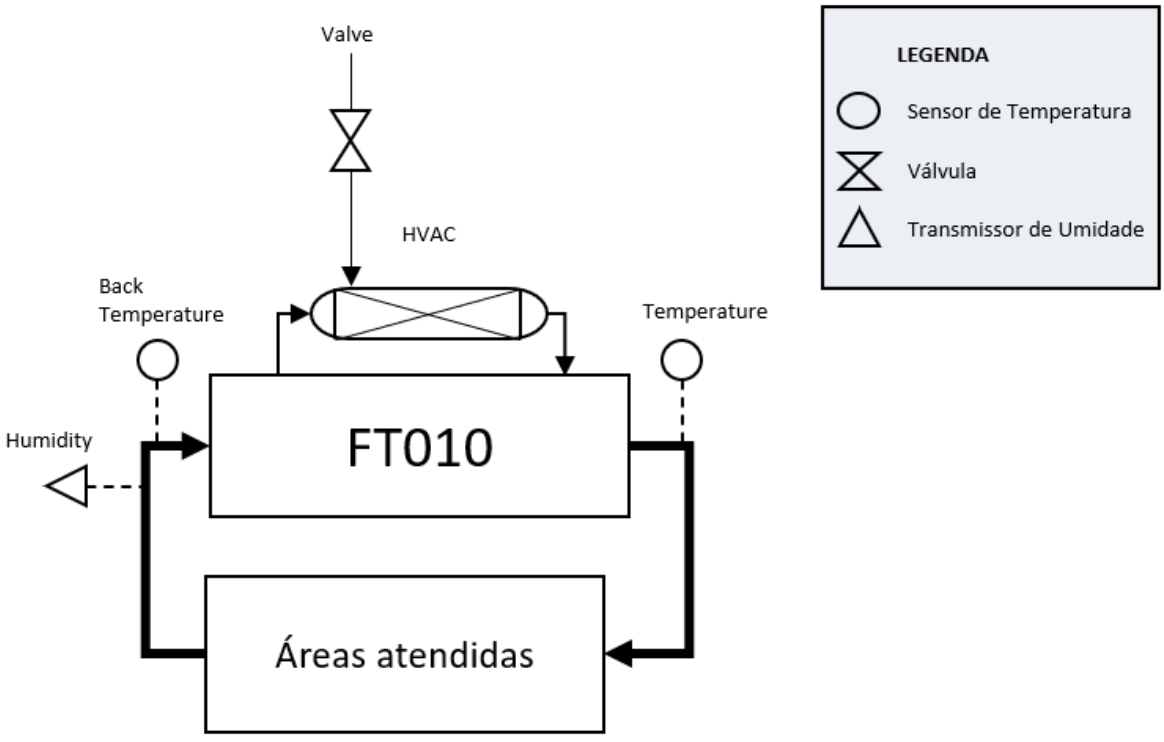


Figura 10 – Diagrama do processo.

Fonte: Elaborado pela autora.

3.3 Tratamento dos dados

Uma vez que a base de dados foi extraída do sistema, foi necessário realizar o tratamento dos dados. No primeiro momento a base de dados possuía duas colunas duplicadas referentes ao tempo de coleta, essas duas colunas foram retiradas da base de dados utilizando a função *drop* da biblioteca *Pandas*.

Em seguida, todas as colunas foram renomeadas, de forma a deixar o trabalho mais intuitivo e de fácil compreensão. Com isso, foi obtido a base de dados final, a qual foi utilizada ao longo deste trabalho.

3.4 Dados nulos

Para que seja possível obter bons resultados ao treinar um algoritmo de aprendizado de máquina é preciso tratar com atenção e cautela os dados nulos presentes na base de dados uma vez que esses valores serão prejudiciais ao processo de aprendizado da rede.

Utilizando as bibliotecas de *Python Pandas* e *Numpy* foi possível obter o percentual de valores nulos em todo o *dataset*. O resultado obtido foi de 0,02572% de valores nulos. Embora esse seja um valor praticamente inexpressivo, é preciso entender bem a base de dados e a origem desses valores.

Sabendo que a base é composta por dados referentes a temperaturas e umidades, que os valores nulos estavam presentes na coluna de temperatura de retorno, como pode ser visto na Figura 11, e que os sistemas observados não são capazes de alcançar temperaturas nulas, foi possível concluir que os valores nulos tratam-se de dados que não foram coletados corretamente, correspondendo a falhas de leituras nos sensores. Para evitar que informações falsas na base de dados dificulte o aprendizado da rede, todos esses dados nulos foram tratados.

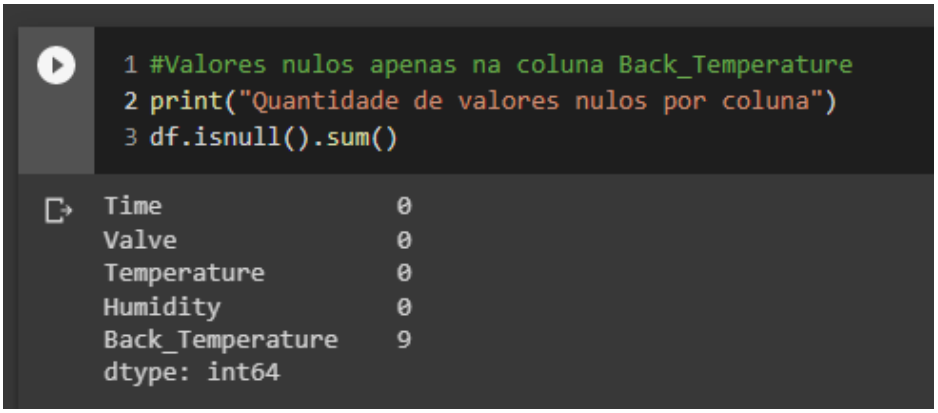


Figura 11 – Quantidade de valores nulos por coluna.

Fonte: Elaborado pela autora.

Utilizando a função *interpolate* da biblioteca *Pandas*, foi possível eliminar todos os valores nulos. A função *interpolate* preenche valores nulos por meio de métodos de interpolação. O método utilizado nesse trabalho foi a interpolação linear, que ignora o *index* da base e trata todos os valores como valores igualmente espaçados. Pandas (2022)

3.5 Classificando os dados

Para ser possível aplicar métodos de aprendizado supervisionado fez-se necessário transformar a base de dados em uma base classificada, ou seja, uma base que possui rótulos para que seja possível comparar a resposta e medir a precisão e acurácia da do método.

Para classificar os dados foram utilizados dados registrados no sistema SAP com informação de todas as manutenções e intervenções realizadas no sistema onde é possível pesquisar todas as trocas em válvulas no período desejado. Assim, a base de dados foi divida em duas classes baseadas nas informações captadas no sistema e seguindo orientações de um técnico especialista no sistema de HVAC da empresa. A primeira classe representa o bom cenário, ou seja, todos os dados presentes nessa classificação representam momentos onde a válvula estava em seu perfeito funcionamento, tanto antes da falha como após intervenção de manutenção. Já a segunda classificação trata-se do cenário ruim, onde a válvula apresentava defeitos ou não estava em seu perfeito funcionamento.

Para classificar os dados, foi utilizado um laço de repetição que adicionava a coluna ”classe” a cada linha da base de dados, classificando-os em 0, 1, respectivamente ”bom cenário” e ”cenário ruim” de acordo com o período e tempo de coleta da base de dados.

Uma representação da base de dados com a coluna classe adicionada pode ser vista na Figura 12.

Time	Valve	Temperature	Humidity	Back_Temperature	classe
2020-01-28 15:23:44	34.454697	17.187500	58.684170	21.701389	0.0
2020-01-28 15:24:44	34.304714	17.171225	58.640770	21.717665	0.0
2020-01-28 15:25:44	34.392952	17.203775	58.579281	21.750217	0.0
2020-01-28 15:26:44	33.891094	17.234520	58.557579	21.733940	0.0
2020-01-28 15:27:44	33.156586	17.250795	58.604599	21.701389	0.0
...	...	...	...	...	...
2021-05-28 15:18:55	27.454987	19.641565	52.003761	21.685112	0.0
2021-05-28 15:19:55	26.857983	19.769964	51.978443	21.668837	0.0
2021-05-28 15:20:55	27.056767	19.911024	51.953125	21.717665	0.0
2021-05-28 15:21:55	26.688835	20.043041	51.916954	21.715857	0.0
2021-05-28 15:22:55	26.352100	20.185907	51.945889	21.717665	0.0

699750 rows × 5 columns

Figura 12 – Base de dados com a coluna classse.

Fonte: Elaborado pela autora.

3.6 Separação dos dados em treino e teste.

A etapa de separação dos dados em um conjunto de treino e um conjunto de teste é imprescindível para a execução do modelo. Segundo Goot (2021), o conjunto de treino é utilizado para a avaliação de diferentes versões do modelo, enquanto o conjunto de teste é utilizado para confirmar as saídas em relação às entradas. No presente trabalho a base de dados foi dividida em 70% para um conjunto de treino e os demais 30% compuseram o conjunto de teste. Vale ressaltar que por se tratar de uma série temporal é de suma importância manter a estrutura temporal, visto que a validação cruzada pode ser uma problemática para esses problemas. Sendo assim, os 30% de dados mais recentes foram utilizados no conjunto de teste.

3.7 Métodos e experimento

3.7.1 Regressão Logística

No primeiro momento foi utilizado o método de regressão logística, chamado *LogisticRegression* disponível na biblioteca de aprendizado de máquinas *Scikit Learn*.

Quase todos os parâmetros necessários para o funcionamento desse método foram utilizados no modo *default* ou seja, nenhuma alteração foi realizada. O único parâmetro alterado foi o *random_state*. Esse parâmetro, de acordo com Scikit-learn Developers (2022), sempre está presente quando a randomização faz parte de um algoritmo da *Scikitlearn* para controlar o gerador de números aleatórios usado. Nesse trabalho esse parâmetro recebeu o valor de zero, para garantir que a cada vez que o código fosse rodado os mesmos resultados seriam obtidos.

Após a inicialização do modelo, foi utilizado o método *fit* que tem como objetivo treinar o algoritmo na base de dados separada para treino. Como parâmetro esse método recebeu o vetor que contém as variáveis de entrada e o vetor que contém a variável de saída. Após treinado, chegou o momento de validar o modelo, aplicando o que foi aprendido na base de dados de teste.

Para a validação foi utilizada a função *score* que calcula a precisão do conjunto, ou seja, o conjunto de rótulos previsto para uma amostra deve corresponder ao conjunto de rótulos verdadeiros. Sendo assim, essa função recebe como parâmetros o X e o Y da base de dados de teste, ou seja, as variáveis dependentes e o rótulo de cada uma delas.

Para esse método, obteve-se uma acurácia de 99%. Uma representação do código em python da utilização desse método pode ser vista na Figura 13.

```
[ ]  1 from sklearn.model_selection import train_test_split
     2 X_train, X_test, y_train, y_test = train_test_split( X, y, test_size = 0.3, random_state = 42)

[ ]  1 from sklearn.linear_model import LogisticRegression
     2 classifier = LogisticRegression(random_state = 0)
     3 classifier.fit(X_train,y_train)

     LogisticRegression(random_state=0)

[ ]  1 classifier.score(X_test,y_test)

     0.9903441705370966
```

Figura 13 – Método de regressão logística.

Fonte: Elaborado pela autora.

3.7.2 Classificador Baseado em Árvore de Decisão

O segundo método utilizado foi o *DecisionTreeClassifier* da biblioteca de aprendizado de máquinas *Scikit Learn*. Num primeiro momento o método foi definido sem nenhuma alteração de parâmetros. No entanto, a árvore de decisão gerada para esse problema era muito grande e sua representação gráfica se mostrou inadequada.

Devido ao tamanho da árvore e a alta acurácia obtida, para evitar qualquer influência de *overfitting*, foi decidido definir o parâmetro de máxima profundidade para 5. Com isso, obteve-se uma nova árvore de decisão, o seu formato pode ser observado na Figura 14, com distância máxima de 5 ramos entre a raiz e qualquer folha e 29 folhas ao total.

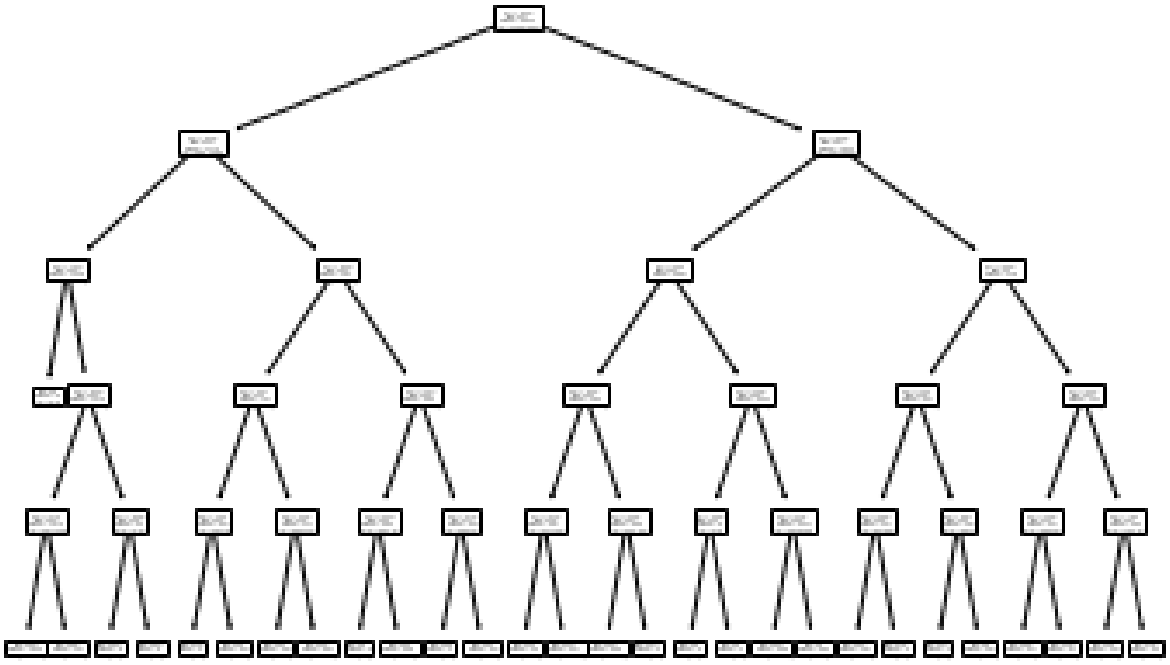


Figura 14 – Árvore de decisão final, contendo 29 folhas.

Fonte: Elaborado pela autora.

Mesmo com a diminuição drástica do tamanho da árvore, após treinar o modelo e validá-lo nos dados de teste foi possível obter uma alta acurácia. O trecho do código referente à aplicação do método de árvore de decisão pode ser analisado na Figura 15.

```
[56]  1 from sklearn import tree
      2 dt = tree.DecisionTreeClassifier(max_depth=5)
      3
      4 dt.fit(X_train,y_train)

      DecisionTreeClassifier(max_depth=5)

[57]  1 dt.score(X_test,y_test)

      0.9910015481719662

[61]  1 dt.get_n_leaves()

      29

[62]  1 dt.get_depth()

      5
```

Figura 15 – Método de árvore de classificação.

Fonte: Elaborado pela autora.

3.7.3 TabNet

Para a implementação da arquitetura TabNet foi utilizado o *framework* de aprendizado de máquinas *Pytorch*.

A TabNet possui três métodos de aprendizado de máquina. Para esse trabalho foi utilizado o método *TabNetClassifier* visto que se está enfrentando um problema de classificação.

TabNet é uma arquitetura compatível com a biblioteca *ScikitLearn*, sendo assim, treinar seu método classificador se torna uma tarefa simples.

No primeiro momento é preciso definir alguns parâmetros para a inicialização do modelo. Para esse trabalho foram definidos os seguintes parâmetros:

- *optimizer_fn* - como função de otimização foi utilizado o otimizador padrão *Adam*, Estimativa de Momento Adaptativo - do inglês *Adaptive Moment Estimation*;
- *optimizer_params* - como taxa de aprendizado inicial foi utilizado 0,02 com decaimento, como sugerido pelos autores em Arik e Pfister (2021);
- *scheduler_params* - esse parâmetro tem como objetivo passar parâmetros para o planejador alterar a taxa de aprendizado durante o treinamento. Nesse trabalho foram passados os parâmetros gamma = 0,95 e *step_size* = 10;
- *scheduler_fn* - como planejador foi mantido o padrão do modelo;
- *mask_type* - como máscara usada para selecionar os recursos do modelo foi utilizada a entmax.

Após a definição dos parâmetros treinou-se o modelo utilizando a função *fit*. Para o treinamento foi considerado o número máximos de épocas a ser treinado igual a 1000, com uma paciência de 50, ou seja, se o modelo permanecer 50 épocas sem apresentar melhorias ele será interrompido. O *Batch_size*, ou tamanho de lote, escolhido foi 1000, visto que os autores recomendam, em Arik e Pfister (2021), algo próximo a ¨10% do total de dados.

O modelo definido com os parâmetros citados anteriormente foi interrompido após 256 épocas e apresentou uma acurácia máxima de 99%. O código em *python* com a implementação da arquitetura pode ser observado na Figura 16

```python
import pytorch_tabnet
from pytorch_tabnet.tab_model import TabNetClassifier

import torch
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import roc_auc_score, accuracy_score
from sklearn.model_selection import KFold

np.random.seed(8)
```

```python
clf1_nopreproc = TabNetClassifier(optimizer_fn=torch.optim.Adam,
                      optimizer_params=dict(lr=2e-2),
                      scheduler_params={"step_size":50, # how to use learning rate scheduler
                                        "gamma":0.9},
                      scheduler_fn=torch.optim.lr_scheduler.StepLR,
                      mask_type='entmax' # "sparsemax"
                     )
```

```
/usr/local/lib/python3.7/dist-packages/pytorch_tabnet/abstract_model.py:75: UserWarning: Device used : cpu
  warnings.warn(f"Device used : {self.device}")
```

```python
clf1_nopreproc.fit(
    X_train,y_train,
    eval_set=[(X_train, y_train), (X_test, y_test)],
    eval_name=['train', 'valid'],
    eval_metric=['accuracy'],
    max_epochs=1000 , patience=50,
    batch_size=1000, virtual_batch_size=500,
    num_workers=0,
    weights=1,
    drop_last=False
)
```

Figura 16 – Implementação da arquitetura TabNet.

Fonte: Elaborado pela autora.

3.8 XGBoost

Após a implementação da arquitetura TabNet, chegou a vez de implementar o algoritimo *XGBoost*. Sua implementação pode ser observada na Figura 17.

```
1 import xgboost as xgb
2 from sklearn.metrics import accuracy_score

1 data_dmatrix = xgb.DMatrix(data=X,label=y)

1 xg_reg = xgb.XGBClassifier(objective ='binary:logistic',
2                            colsample_bytree = 0.3,
3                            learning_rate = 0.1,
4                 max_depth = 5, alpha = 10, n_estimators = 10)

1  eval_set = [(X_train, y_train), (X_test, y_test)]

1 xg_reg.fit(X_train,y_train,eval_metric=["error"], eval_set=eval_set, verbose=False)
2
3 preds = xg_reg.predict(X_test)

1 xg_reg.score(X_test,y_test)

0.9558604263427415
```

Figura 17 – Implementação da arquitetura XGBoost.

Fonte: Elaborado pela autora.

O primeiro passo para se utilizar esse modelo é converter o conjunto de dados em uma estrutura de dados chamada Dmatrix. De acordo com XGBoost Developers (2021) essa estrutura é otimizada e por isso promove ganhos de eficiência e desempenho.

Após a conversão dos dados, é preciso inicializar o modelo. Para isso, foram utilizados os seguintes parâmetros:

- *objective* - esse parâmetro determina a função de perda a ser usada, nesse trabalho utilizou-se a função *binary.logistic*;
- *colsample_bytree* - esse parâmetro foi definido como 0.3 e é o responsável por definir a porcentagem de recursos que será utilizado pela árvore;
- *learning_rate* - a taxa de aprendizado, ou seja, o tamanho do passo em cada iteração foi definido como 0.1;
- *max_depth* - esse parâmetro determina qual a profunidade máxima que a árvore pode chegar, nesse caso foi escolhido a profundidade 5;
- *alpha* - esse parâmetro se refere ao tipo de regularização aplicada ao modelo.
- *n_estimators* - esse parâmetro se refere à quantidade de árvore que se deseja construir no modelo e foi definido como 10.

Após a inicialização do modelo com os parâmetros citados anteriormente, o modelo foi treinado levando em consideração a métrica de erro de classificação. A implementação da biblioteca de aprendizado de máquinas *XGBoost* apresentou um bom desempenho com uma acurácia de 95%.

3.9 Rede Neural

Finalmente, após implementar os métodos citados nas seções anteriores, foi desenvolvida uma rede neural para predizer possíveis falhas em válvulas de um sistema de HVAC.

Como a base de dados está configurada como uma série temporal, aqui utilizou-se uma rede neural recorrente, visto que possuem memória e por isso são vantajosas para lidar com dados sequenciais.

No primeiro momento, foi preciso redimensionar a base de dados. Como a base inicial possui uma enorme quantidade de dados, não foi possível rodar uma rede neural recorrente com todos os dados devido a uma limitação de memória. Para redimensionar, foi preciso garantir que a sequência da série temporal não seria perdida, por isso, foi utilizado a função *resample* do pacote *Pandas* para manipulação e análise de dados. Com essa função, que pode ser vista na Figura 18, foi possível diminuir a frequência dos dados de um minuto para uma hora. Ou seja, a nossa nova base de dados tem dados de cada hora do dia 28 de janeiro de 2020 ao dia 28 de maio de 2021.

```
1 df = df.resample('60T', on = 'Time').mean()
```

```
1 df
```

Time	Valve	Temperature	Humidity	Back_Temperature
2020-01-28 15:00:00	30.961320	17.767250	59.953227	21.815987
2020-01-28 16:00:00	32.229765	18.304669	61.715554	22.048731
2020-01-28 17:00:00	26.595899	18.711721	63.039339	21.859085
2020-01-28 18:00:00	25.024177	19.824671	66.926660	22.037850
2020-01-28 19:00:00	10.292481	21.572778	66.956620	22.414460
...	...	...	...	...
2021-05-28 11:00:00	32.874664	20.408347	55.304060	22.084448
2021-05-28 12:00:00	36.779631	19.662935	54.159794	21.931272
2021-05-28 13:00:00	31.371524	20.320939	53.454137	22.023443
2021-05-28 14:00:00	39.125112	19.690212	52.016480	22.035047
2021-05-28 15:00:00	30.537005	19.336959	51.888020	21.747858

11665 rows × 4 columns

Figura 18 – Redimensionamento da base de dados.

Fonte: Elaborado pela autora.

Em seguida, a base de dados foi rotulada entre ”bom cenário”e ”cenário ruim”novamente, utilizando os mesmos critérios e métodos que foram citados na seção 3.2 deste trabalho. Após classificar todas as amostras da base de dados, os dados foram divididos entre 70% para o conjunto de treino e 30% para conjunto de teste assim como foi descrito na seção 3.6.

Para efetuar a predição é preciso considerar uma janela de entrada da rede. Para esse trabalho considerou-se adequado uma janela de três dias. Isso quer dizer que a rede olhará para os dados dos últimos três dias e por meio dele tentará predizer o próximo valor, podendo classificá-lo como 0 = ”bom cenário”ou 1 = ”cenário ruim”.

Uma vez que os dados foram devidamente tratados e separados, e que os conjuntos de dados de treino e teste foram criados, conforme Figura 19, deu-se início à construção da rede. Para a construção da rede foi utilizado o *framework Keras* da biblioteca de aprendizado de máquina e inteligência artificial *TensorFlow*.

A rede construída para esse trabalho possui apenas duas camadas e o código de

```
dias = 24*3
```

```
#função para criar os conjuntos de dados de treino
def create_dataset(dataset, look_back=dias):
    dataX, dataY = [], []
    for i in range(len(dataset) - look_back):
        a = dataset.iloc[i:(i + look_back), 0:4]
        dataX.append(a)
        dataY.append(dataset.iloc[i + look_back, 4])
    print(len(dataY))
    return np.array(dataX), np.array(dataY)
```

```
look_back = dias
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
```

Figura 19 – Código da implementação da janela de entrada da rede e do conjunto de dados de treino.

Fonte: Elaborado pela autora.

implementação pode ser visto na Figura 20. A primeira camada é uma camada *LSTM* - do inglês *Long Short-Term Memory* ou memória de longo e curto prazo com 120 neurônios. A quantidade de neurônios foi escolhida empiricamente, para isso foram feitos diversos testes e o valor de 120 neurônios foi escolhido com base no desempenho nos dados de treino. Segundo Livieris, Pintelas e Pintelas (2020) a ideia de se usar modelos *LSTM* em séries temporais é que eles podem capturar as informações do padrão de sequência com eficiência, diferente de outros modelos que geralmente não são adaptados para gerenciar dados temporais complexos e longos.

```
input_size = (trainX.shape[1], trainX.shape[2])
model = tf.keras.Sequential()

model.add(tf.keras.layers.LSTM(120, input_shape = input_size,
                               return_sequences=False))
model.add(tf.keras.layers.Dense(1, activation="sigmoid"))

model.summary()
```

Figura 20 – Construção da rede neural recorrente.

Fonte: Elaborado pela autora.

A segunda e última camada é uma camada densa de um neurônio, ou seja, uma camada que está totalmente conectada com a camada anterior e que retornará a saída do modelo. A função de ativação utilizada para essa camada foi a função Sigmoid. A escolha da função de ativação se deu pelo fato de que a última camada é para calcular o custo e se está trabalhando com um problema de classificação que deseja retornar o valor 0 ou 1.

Em seguida foi utilizado a função *summary* para ser possível visualizar as camadas criadas como pode ser visto na Figura 21.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 120)               60000

 dense (Dense)               (None, 1)                 121

=================================================================
Total params: 60,121
Trainable params: 60,121
Non-trainable params: 0
_________________________________________________________________
```

Figura 21 – Sumário do modelo de redes neurais recorrentes.

Fonte: Elaborado pela autora.

Após a construção do modelo, foi necessário compilá-lo, conforme pode ser visto na Figura 22. Para isso, utilizou-se a função de custo *binary_crossentropy* que compara cada valor predito com a saída real que pode ser 0 ou 1 e o otimizador *Adam* que combina ideias de outros otimizadores utilizando a média exponencialmente decrescente de gradientes passados.

```
1 model.compile(loss='binary_crossentropy',
2               optimizer='Adam',
3               metrics="accuracy")
```

Figura 22 – Compilando o modelo.

Fonte: Elaborado pela autora.

O modelo foi treinado com 20 épocas, utilizando a métrica de acurácia e apresentou um bom desempenho alcançando a acurácia máxima de 99%.

Parte IV

Resultados

4 Resultados

Neste capítulo serão apresentados os resultados obtidos por meio da implementação dos métodos citados no Capítulo 3.

4.1 Regressão Logística

Como citado anteriormente, o método de regressão apresentou uma acurácia muito satisfatória para o problema proposto. Isso pode ter ocorrido por vários motivos. Aqui se destaca a facilidade de métodos de regressão logística para modelar variáveis binários.

Sabendo que o problema de classificação pretende rotular os dados em duas classes, sendo elas "bom cenário" igual a 0 e "cenário ruim" igual a 1, plotou-se a matriz confusão, que pode ser observada na Figura 23, utilizando a função *confusion_matrix* da biblioteca de aprendizado de máquinas *ScikitLearn*.

Figura 23 – Matriz confusão do método de regressão logística.

Fonte: Elaborado pela autora.

Por meio da matriz confusão é possível observar que o método desempenhou bem ao longo de toda base de dados, no entanto, teve maior dificuldade ao analisar o "cenário

ruim". Acredita-se que em alguns instantes durante o cenário ruim a válvula apresentava funcionamento aparentemente normal, o que pode ter sido lido como "bom cenário" e assim confundido o modelo. Ainda assim, vale ressaltar que esse método apresentou acurácia de 99% assim apresentando um excelente resultado.

4.2 Árvore de Decisão

A árvore de decisão é um método muito utilizado para classificar problemas com mais de duas variáveis, no entanto, isso não a impede de desempenhar bem ao enfrentar um problema com apenas dois rótulos.

Sabendo que o presente trabalho possuía dados o suficiente para abastecer a base da árvore e que o conjunto foi completamente e devidamente classificado, obteve-se, como esperado, um resultado satisfatório de 99% de acurácia ao classificar o comportamento da válvula em cenário bom e cenário ruim.

Para esse modelo, assim como o anterior, foi utilizado o método de matriz confusão para analisar o resultado. Na Figura 24, pode-se observar na matriz confusão que a árvore de decisão também encontrou maior dificuldade em classificar os dados do cenário ruim, no entanto, esse modelo cometeu menos erros que o anterior, classificando erroneamente apenas 1889 amostras de um total de 209925 amostras, enquanto a regressão logística classificou 2027 de maneira errada.

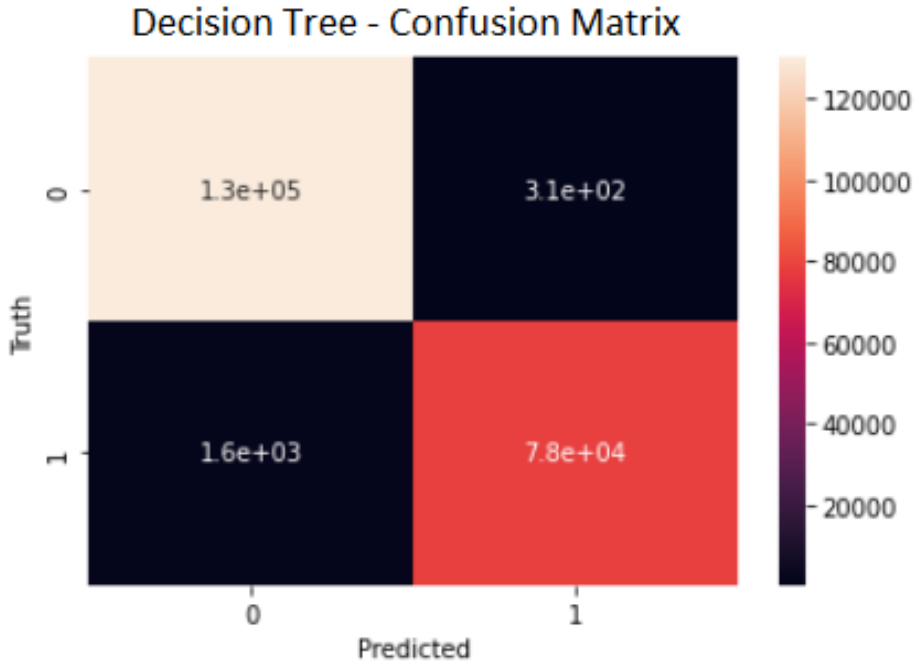


Figura 24 – Matriz confusão do método de árvore de decisão.

Fonte: Elaborado pela autora.

4.3 TabNet

Tabnet é uma arquitetura muito utilizada e que desempenha muito bem em problemas de dados tabulares. Para avaliar o desempenho desse método no presente trabalho foi utilizada a métrica de acurácia que demonstrou que a melhor acurácia no conjunto de teste foi 99%.

Para facilitar a visualização, foi traçado um gráfico, utilizando o módulo *Pyplot* da biblioteca *matplotlib* para *Python*. Essa biblioteca é capaz de plotar gráficos 2D de maneira simples.

Na Figura 25 fica fácil de se observar que o modelo apresentou uma excelente precisão. No gráfico temos, em azul, a acurácia para o conjunto de treinamento e, em laranja, a acurácia no conjunto de teste.

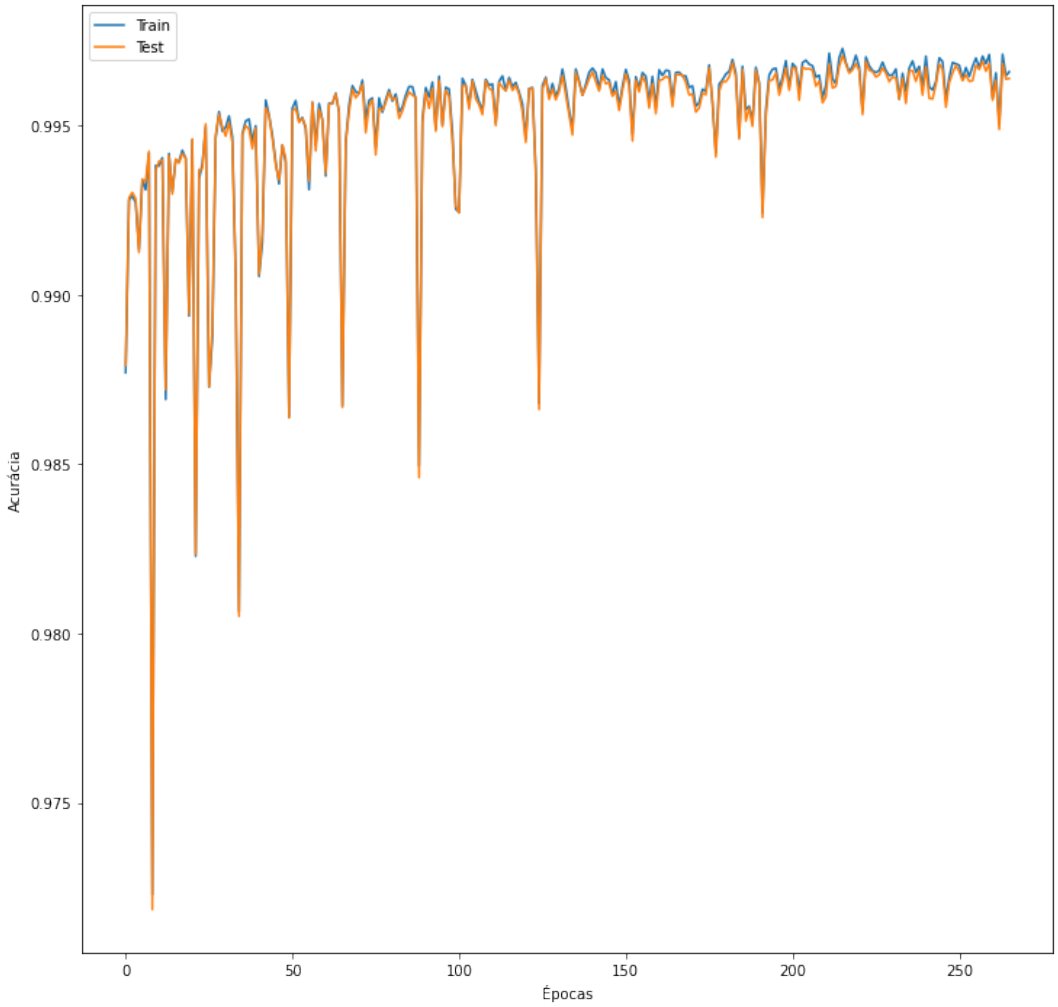


Figura 25 – Acurácia de TabNet.

Fonte: Elaborado pela autora.

4.4 XGBoost

A biblioteca XGBoost apresentou uma acurácia de 95,59% no conjunto de dados de validação.

Assim como dito pelos autores, em Chen e Guestrin (2016), o algoritmo possui uma alta velocidade de processamento. Esse fato pôde ser percebido no desenvolvimento deste trabalho, pois quando comparado, por exemplo, com o método citado anteriormente TabNet, pode-se dizer que XGBoost apresentou uma menor acurácia, no entanto, ainda assim o modelo foi capaz de trazer um bom resultado com uma maior velocidade de processamento, ou seja, levou-se muito menos tempo para ser processado.

Além da métrica de acurácia, foi gerado o gráfico de erro de classificação. Nesse gráfico, visto na Figura 26 pode-se observar, em azul, a curva do erro de classificação no conjunto de treino, e em laranja, a curva do erro de classificação no conjunto de teste. É possível observar que o erro de classificação se manteve baixo todo o tempo.

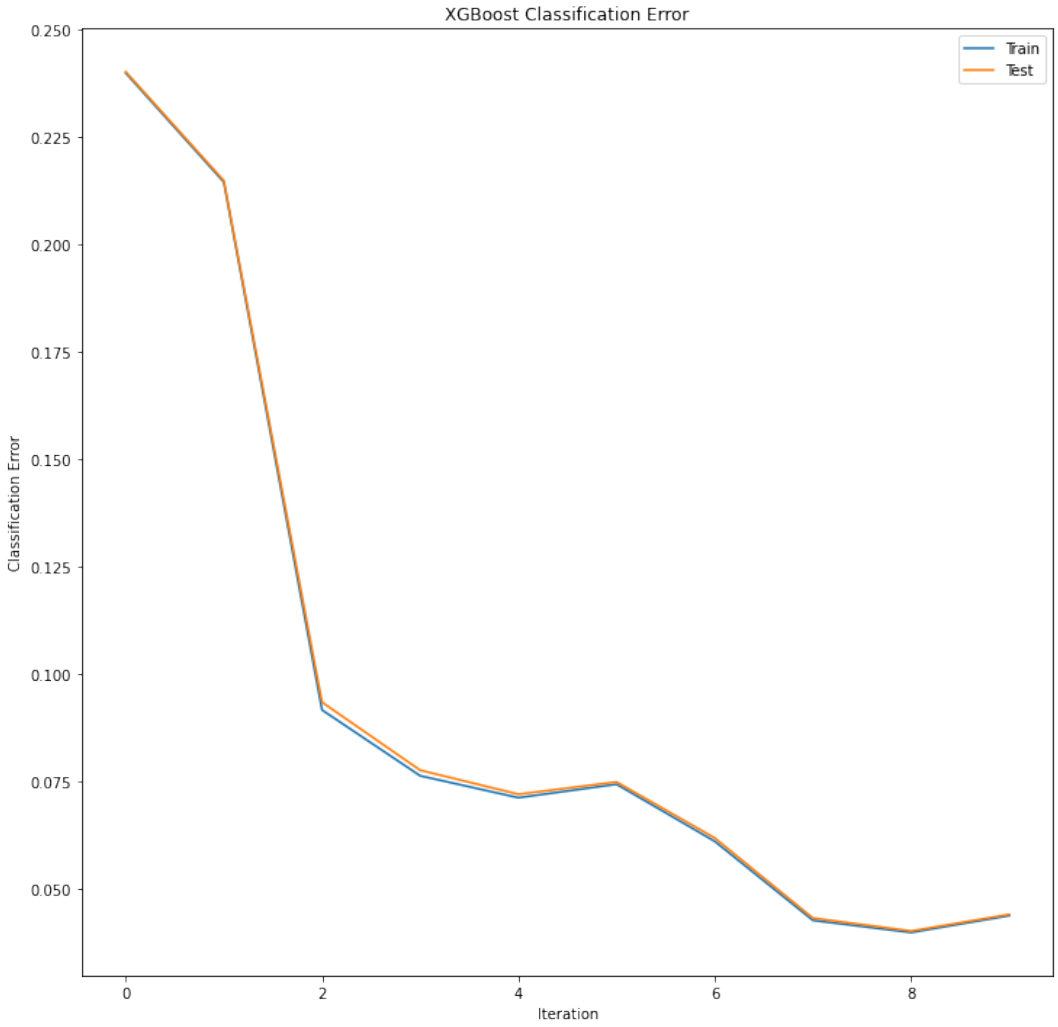


Figura 26 – Erro de Classificação de XGBoost.

Fonte: Elaborado pela autora.

4.5 Rede Neural Recorrente

O último método implementado foi a Rede Neural Recorrente. Como citado anteriormente redes neurais recorrentes possuem memória e por isso costumam apresentar um bom desempenho em séries temporais.

Logo, assim como esperado, a rede neural construída nesse projeto apresentou bons resultados ao tentar predizer o funcionamento da válvula do sistema de refrigeração.

Após treinar o modelo, foi obtida uma acurácia máxima de 99%.

Para facilitar a visualização, foi gerado um gráfico utilizando o módulo *Pyplot* da biblioteca *Matplotlib* com as curvas do valor real e do valor predito pelo modelo. Pelo gráfico, que pode ser visto na Figura 27, é possível observar que praticamente todos os valores foram preditos de maneira correta. No entanto, é preciso notar que a predição está alguns dias atrasadas, ou seja, quando o cenário altera, o modelo demora um pouco para se ajustar às alterações de funcionamento da válvula.

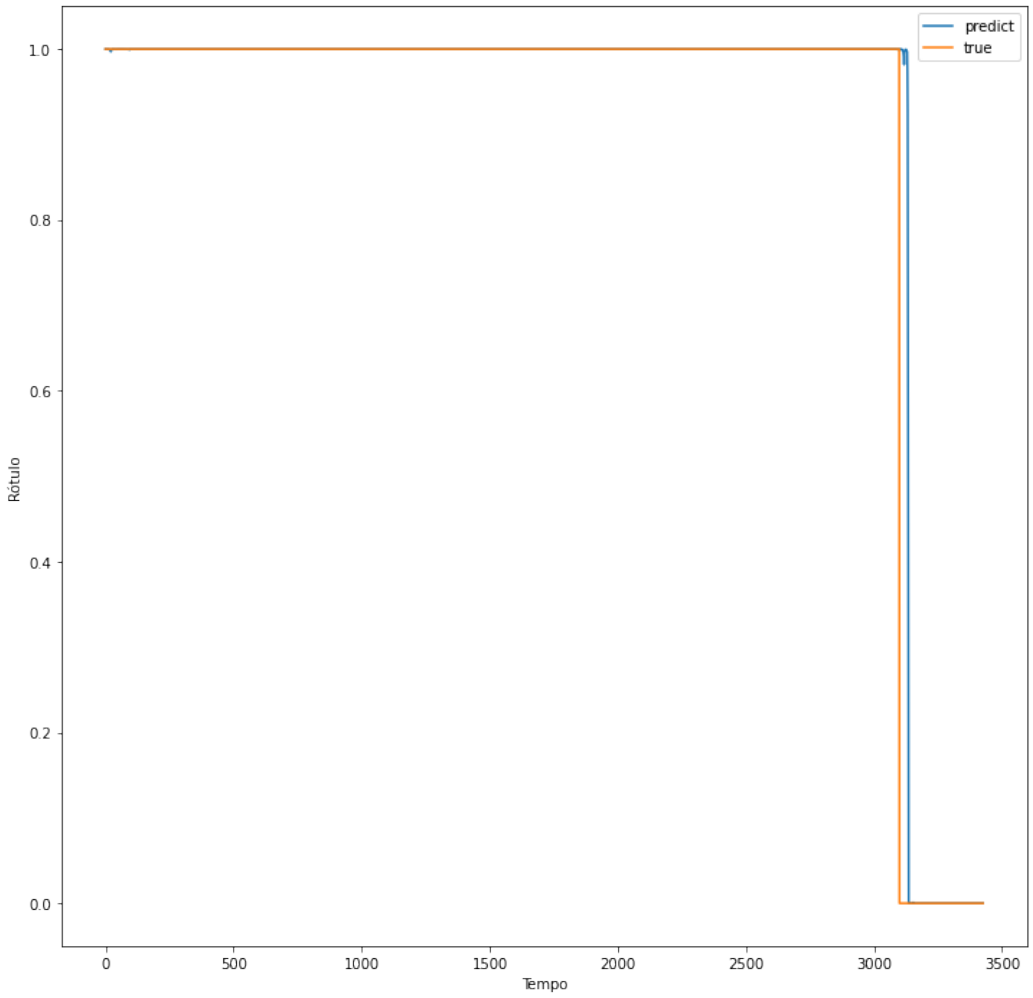


Figura 27 – Valor real e valor predito pela rede neural recorrente.

Fonte: Elaborado pela autora.

4.6 Comparação de resultados

Conforme citado nas seções acima, segue na Tabela 1 o resumo dos resultados obtidos pelos cinco métodos utilizados para a resolução do problema proposto:

Tabela 1 – Comparando os resultados

Fonte: Elaborado pela autora.

Algoritmo	**Acurácia**
Regressão logística	0.9903
Árvore de decisão	0.9910
TabNet	0.9970
XGBoost	0.9558
Rede Neural Recorrente	0.9936

Por meio dos dados acima é possível concluir que o algoritmo XGBoost apresentou a menor acurácia ao tentar classificar o conjunto de teste corretamente enquanto o modelo TabNet apresentou a maior acurácia, acertando quase 100% dos valores preditos.

Embora haja diferença nas acurácias, todos os métodos podem ser considerados como uma boa solução para classificar corretamente o funcionamento das válvulas. A escolha de qual método aplicar ficará a critério do especialista do sistema e poderá depender de outros fatores, como custo e velocidade de processamento.

Parte V

Conclusão

5 Conclusão

5.1 Considerações Finais

Tendo em vista que a proposta deste trabalho de conclusão de curso consistiu na implementação de técnicas de inteligência artificial baseada em métodos estatísticos e de redes neurais para classificar diferentes cenários referentes a defeitos em válvulas de um sistema de HVAC de uma indústria farmacêutica no norte de Minas Gerais, pode-se dizer que o objetivo foi alcançado com sucesso.

Ao fim deste trabalho foi possível concluir que métodos de inteligência artificial conseguem alcançar ótimos resultados em problemas de classificação binários. Durante o desenvolvimento foi possível observar que o método que apresentou melhor resultado foi o que utilizou a arquitetura de aprendizado de máquinas TabNet seguido do método baseado em redes neurais recorrentes. O método que apresentou menor acurácia foi o que utilizou a biblioteca XGBoost, no entanto, ainda que esse método tenha apresentado uma menor acurácia, ele apresentou um desempenho muito satisfatório e que seria capaz de contribuir positivamente na operação da indústria farmacêutica.

5.2 Sugestões para Trabalhos Futuros

Como sugestão para trabalhos futuros, pretende-se expandir a construção do *dataset*. Para isso pode-se alterar os hiperparâmetros dos métodos citados acima e realizar um comparativo aprofundado entre eles verificando se há uma melhora significativa no desempenho dos métodos e no processo de predição de falhas. Além disso, pode-se avaliar métodos de busca automática de arquiteturas de redes neurais (*NAS* - do inglês *Neural Architecture Search*), como sugerido em Tan et al. (2019).

Referências

AFRAM, Abdul et al. Artificial neural network (ANN) based model predictive control (MPC) and optimization of HVAC systems: A state of the art review and case study of a residential HVAC system. *Energy and Buildings*, Elsevier, v. 141, p. 96–113, 2017. Citado 2 vezes na página 18.

ARIK, Sercan Ö; PFISTER, Tomas. Tabnet: Attentive interpretable tabular learning. In: 8. PROCEEDINGS of the AAAI Conference on Artificial Intelligence. 2021. v. 35, p. 6679–6687. Citado 5 vezes nas páginas 27, 38, 39.

BORISOV, Vadim et al. Deep neural networks and tabular data: A survey. *arXiv preprint arXiv:2110.01889*, 2021. Citado 2 vezes na página 24.

CHARBUTY, Bahzad; ABDULAZEEZ, Adnan. Classification based on decision tree algorithm for machine learning. *Journal of Applied Science and Technology Trends*, v. 2, n. 01, p. 20–28, 2021. Citado 1 vez nas páginas 25, 26.

CHEN, Tianqi; GUESTRIN, Carlos. Xgboost: A scalable tree boosting system. In: PROCEEDINGS of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016. P. 785–794. Citado 3 vezes nas páginas 28, 49.

DIETTERICH, Tom. Overfitting and undercomputing in machine learning. *ACM computing surveys (CSUR)*, ACM New York, NY, USA, v. 27, n. 3, p. 326–327, 1995. Citado 1 vez na página 21.

ESLING, Philippe; AGON, Carlos. Time-series data mining. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 45, n. 1, p. 1–34, 2012. Citado 1 vez na página 20.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep learning*. MIT press, 2016. Citado 1 vez na página 15.

GOOT, Rob van der. We need to talk about train-dev-test splits. In: PROCEEDINGS of the 2021 Conference on Empirical Methods in Natural Language Processing. 2021. P. 4485–4494. Citado 1 vez na página 35.

HANDBOOK, ASHRAE. *HVAC systems and equipment*. chapter, 1996. v. 39. Citado 1 vez na página 14.

HILBE, Joseph M. Logistic regression. *International encyclopedia of statistical science*, v. 1, p. 15–32, 2011. Citado 1 vez na página 21.

KANANI, Pratik; PADOLE, Mamta. Deep learning to detect skin cancer using google colab. *International Journal of Engineering and Advanced Technology Regular Issue*, v. 8, n. 6, p. 2176–2183, 2019. Citado 1 vez na página 30.

KLEINBAUM, David G et al. *Logistic regression*. Springer, 2002. Citado 2 vezes nas páginas 21, 22.

LEE, Wo Jae et al. Predictive maintenance of machine tool systems using artificial intelligence techniques applied to machine condition data. *Procedia Cirp*, Elsevier, v. 80, p. 506–511, 2019. Citado 2 vezes na página 19.

LIVIERIS, Ioannis E; PINTELAS, Emmanuel; PINTELAS, Panagiotis. A CNN–LSTM model for gold price time-series forecasting. *Neural computing and applications*, Springer, v. 32, n. 23, p. 17351–17360, 2020. Citado 1 vez na página 43.

LOH, Wei-Yin. Classification and regression trees. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, Wiley Online Library, v. 1, n. 1, p. 14–23, 2011. Citado 2 vezes nas páginas 25, 26.

MADIRAJU, NISCHAL. *Problems faced while fitting data to a model*. 2022. Disponível em: https://www.medium.datadriveninvestor.com/problems-faced-while-fitting-data-to-a-model-845afe6b369. Acesso em: 16 de outubro de 2022. Citado 1 vez nas páginas 20, 21.

MIKOLOV, Tomáš et al. Recurrent neural network based language model. In: ELEVENTH annual conference of the international speech communication association. 2010. Citado 1 vez na página 23.

NILSSON, Nils J. *The quest for artificial intelligence*. Cambridge University Press, 2009. Citado 1 vez na página 15.

NVIDIA DEVELOPERS. *XGBoost*. 2022. Disponível em: https://www.nvidia.com/en-us/glossary/data-science/xgboost/. Acesso em: 12 de outubro de 2022. Citado 1 vez na página 28.

PANDAS, Development Team. *pandas.DataFrame.interpolate*. 2022. urlhttps://pandas.pydata.org/pan docs/stable/reference/api/pandas.DataFrame.interpolate.html. Citado 1 vez na página 34.

RUSCHEL, Edson; SANTOS, Eduardo Alves Portela; LOURES, Eduardo de Freitas Rocha. Industrial maintenance decision-making: A systematic literature review. *Journal of Manufacturing Systems*, Elsevier, v. 45, p. 180–194, 2017. Citado 1 vez na página 15.

SATRIO, Pujo et al. Optimization of HVAC system energy consumption in a building using artificial neural network and multi-objective genetic algorithm. *Sustainable Energy Technologies and Assessments*, Elsevier, v. 35, p. 48–57, 2019. Citado 1 vez na página 18.

SCIKIT-LEARN, Developers. *Decision Trees*. 2022. urlhttps://scikit-learn.org/stable/modules/tree. Citado 2 vezes na página 25.

SCIKIT-LEARN DEVELOPERS. *Glossary of Common Terms and API Elements*. 2022. Disponível em: https://www.scikit-learn.org/stable/glossary.html. Acesso em: 16 de outubro de 2022. Citado 1 vez na página 36.

TAN, Mingxing et al. Mnasnet: Platform-aware neural architecture search for mobile. In: PROCEEDINGS of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019. P. 2820–2828. Citado 1 vez na página 53.

VAN ROSSUM, Guido; DRAKE JR, Fred L. *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995. v. 620. Citado 1 vez na página 30.

WANG, Sun-Chong. Artificial neural network. In: INTERDISCIPLINARY computing in java programming. Springer, 2003. P. 81–100. Citado 2 vezes nas páginas 22, 23.

WEI, William WS. Time series analysis, 2013. Citado 1 vez na página 20.

XGBOOST DEVELOPERS. *Core Data Structure*. 2021. Disponível em: https://www.xgboost.readthedocs.io/en/stable/python/python_api.html#module-xgboost.core. Acesso em: 24 de outubro de 2022. Citado 1 vez na página 40.

XU, Lei et al. Modeling tabular data using conditional gan. *Advances in Neural Information Processing Systems*, v. 32, 2019. Citado 1 vez na página 24.

ZHANG, Aston et al. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021. Citado 1 vez nas páginas 23, 24.

ZHAO, Yang et al. Artificial intelligence-based fault detection and diagnosis methods for building energy systems: Advantages, challenges and the future. *Renewable and Sustainable Energy Reviews*, Elsevier, v. 109, p. 85–101, 2019. Citado 3 vezes nas páginas 19, 20.

Apêndices

APÊNDICE A – Código de implementação parte 1

```
from google.colab import drive
drive.mount('/content/drive')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
```

```
Mounted at /content/drive
```

```
#Dados
df1 = pd.read_csv('/content/drive/MyDrive/TCC/teste1.csv')
df2 = pd.read_csv('/content/drive/MyDrive/TCC/TT500.csv')
#Concatenando tabelas
df = pd.merge(df1, df2, how = 'left', on = 'MC1FC010V009 Time')
#Tratando dados
df = df.drop(columns = ['MC1FC010TT002 Time','MC1FC010MT510 Time'])
df = df.rename({'MC1FC010V009 Time': 'Time', 'MC1FC010V009 ValueY': 'Valve', 'MC1FC010T
#df = df.set_index(pd.DatetimeIndex(df['Time']))


```

```
#df.to_csv('/content/drive/MyDrive/TCC/dataset.csv')
```

```
#df.plot(x='Time', y=['Valve', 'Temperature', 'Humidity', 'Back_Temperature'])
```

```
#decompondo a série temporal
#usando multiplicative pq a variação n é linear
#decomp = seasonal_decompose(df['Valve'], model = 'additive', freq = 5000)
#decomp.plot()
```

```
# #Calculando o percentual de valores nulos
# total = np.product(df.shape)
# missing = df.isnull().sum().sum()
# perc_missing = missing/total*100
# print("valores nulos/total = ", perc_missing)
```

```
# #Valores nulos apenas na coluna Back_Temperature
# print("Quantidade de valores nulos por coluna")
# df.isnull().sum()
```

```
df = df.interpolate()
```

```
df.isnull().sum()
```

```
Time                0
Valve               0
Temperature         0
Humidity            0
```

```
Back_Temperature    0
dtype: int64
```

```
df.sort_values(by=['Time'],inplace=True)
df['Time'] = pd.to_datetime(df['Time'])
#df.plot(x='Time', y=['Valve', 'Temperature', 'Humidity', 'Back_Temperature'])
#df
```

```
cenarios = []

import datetime
Time_format = '%Y-%m-%d %H:%M:%S'
"bom cenário"
date = '2020-01-28 15:22:44'
cenarios.append(datetime.datetime.strptime(date, Time_format))
date = '2020-11-12 00:00:00'
cenarios.append(datetime.datetime.strptime(date, Time_format))

"cenário ruim"
date = '2020-11-12 00:01:00'
cenarios.append(datetime.datetime.strptime(date, Time_format))
date = '2021-05-15 00:00:00'
cenarios.append(datetime.datetime.strptime(date, Time_format))

"cenário pós conserto"
date = '2021-05-15 00:01:00'
cenarios.append(datetime.datetime.strptime(date, Time_format))
date = '2021-05-29 00:00:00'
cenarios.append(datetime.datetime.strptime(date, Time_format))
```

```
df
```

	Time	Valve	Temperature	Humidity	Back_Temperature

```
1 cenarios
```

```
[datetime.datetime(2020, 1, 28, 15, 22, 44),
 datetime.datetime(2020, 11, 12, 0, 0),
 datetime.datetime(2020, 11, 12, 0, 1),
 datetime.datetime(2021, 5, 15, 0, 0),
 datetime.datetime(2021, 5, 15, 0, 1),
 datetime.datetime(2021, 5, 29, 0, 0)]
```

	...	...	...	...	...

```
1 aux = []
2 i = 0
3 aux = df[df.Time > cenarios[i]]
4 aux = aux[aux.Time < cenarios[i+1]]
5 aux
```

	Time	Valve	Temperature	Humidity	Back_Temperature
1	2020-01-28 15:23:44	34.454697	17.187500	58.684170	21.701389
2	2020-01-28 15:24:44	34.304714	17.171225	58.640770	21.717665
3	2020-01-28 15:25:44	34.392952	17.203775	58.579281	21.750217
4	2020-01-28 15:26:44	33.891094	17.234520	58.557579	21.733940
5	2020-01-28 15:27:44	33.156586	17.250795	58.604599	21.701389
...	...	...	...	...	...
415178	2020-11-11 23:55:32	0.000000	17.731842	55.385559	22.844328
415179	2020-11-11 23:56:32	0.000000	17.731842	55.403645	22.828053
415180	2020-11-11 23:57:32	0.000000	17.746311	55.443432	22.828053
415181	2020-11-11 23:58:32	0.000000	17.762587	55.486832	22.828053
415182	2020-11-11 23:59:32	0.000000	17.762587	55.519386	22.811777

415182 rows × 5 columns

```
1 classe = {'Group':np.zeros(len(aux),dtype=int)}
2 classe = pd.DataFrame(classe)
3 base = pd.DataFrame(aux)
```

```
1 # base
```

```
1 i = 2
2 j = 2
3
4 while i < len(cenarios):
5   if i == 2:
6     aux = df[df.Time > cenarios[i]]
```

```
    aux = aux[aux.Time < cenarios[i+1]]
    aux1 = {'Group':1*np.ones(len(aux),dtype=int)}
    aux1 = pd.DataFrame(aux1)
    classe = pd.concat([classe,aux1], ignore_index = True)
    base = pd.concat([base,aux], ignore_index = True)
    i = i+2

  elif i == 4:
    aux = df[df.Time > cenarios[i]]
    aux = aux[aux.Time < cenarios[i+1]]
    aux1 = {'Group':0*np.ones(len(aux),dtype=int)}
    aux1 = pd.DataFrame(aux1)
    classe = pd.concat([classe,aux1], ignore_index = True)
    base = pd.concat([base,aux], ignore_index = True)
    i = i+2
```

```
# aux
```

```
classe = classe.astype(dtype=float)

#base.plot(x='Data', y=['TT001', 'TT500', 'V004','V005','classe'])

base['classe'] = classe
base.reindex()
base.set_index('Time',inplace=True)
#base=shuffle(base)
```

```
# base
```

```
# base.to_csv('/content/drive/MyDrive/TCC/basebase.csv')
```

```
# base.dtypes
```

```
# corr_matrix = base.corr()
# print("Correlação:")
# print(corr_matrix["Valve"].sort_values(ascending=False))
```

```
from sklearn.preprocessing import LabelEncoder
from keras.utils import np_utils
from sklearn.model_selection import train_test_split
labelencoder = LabelEncoder()
classe = labelencoder.fit_transform(base['classe'])

X = base.iloc[:, 0:4]
y = base.iloc[:, 4]

```

```
# y
```

```
1 # X
```

```
1 y = np.array(y)
```

```
1 X = np.array(X)
```

```
1 from sklearn.model_selection import train_test_split
2 X_train, X_test, y_train, y_test = train_test_split( X, y, test_size = 0.3, random_stat
```

```
1 from sklearn.linear_model import LogisticRegression
2 classifier = LogisticRegression(random_state = 0)
3 classifier.fit(X_train,y_train)
```

```
    LogisticRegression(random_state=0)
```

```
1 classifier.score(X_test,y_test)
```

```
    0.9903441705370966
```

```
1 y_pred = classifier.predict(X_test)
```

```
1 y_pred
```

```
    array([0., 0., 0., ..., 1., 1., 0.])
```

```
1 # from sklearn.metrics import confusion_matrix
2 # cm = confusion_matrix(y_test, y_pred)
3
4 # # Imprime a matriz confusão, tente entender o resultado
5 # cm
```

```
1 # import seaborn as sn
2 # sn.heatmap(cm, annot = True)
3 # plt.xlabel('Predicted')
4 # plt.ylabel('Truth')
```

```
1 from sklearn import tree
2 dt = tree.DecisionTreeClassifier(max_depth=5)
3
4 dt.fit(X_train,y_train)
```

```
    DecisionTreeClassifier(max_depth=5)
```

```
1 dt.score(X_test,y_test)
```

```
    0.9910015481719662
```

```
1 # dt.get_n_leaves()
```

```
# dt.get_depth()
```

```
dtypred = dt.predict(X_test)
```

```
dtypred
```

```
array([0., 0., 0., ..., 1., 1., 0.])
```

```
# from sklearn.metrics import confusion_matrix
# cm = confusion_matrix(y_test, dtypred)

# # Imprime a matriz confusão, tente entender o resultado
# cm
```

```
# import seaborn as sn
# sn.heatmap(cm, annot = True)
# plt.xlabel('Predicted')
# plt.ylabel('Truth')
```

```
# tree.plot_tree(dt)
```

```
pip install pytorch-tabnet
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Collecting pytorch-tabnet
  Downloading pytorch_tabnet-4.0-py3-none-any.whl (41 kB)
     |████████████████████████████████| 41 kB 560 kB/s
Requirement already satisfied: scipy>1.4 in /usr/local/lib/python3.7/dist-packages (1
Requirement already satisfied: torch<2.0,>=1.2 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: tqdm<5.0,>=4.36 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: numpy<2.0,>=1.17 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: scikit_learn>0.21 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-pac
Installing collected packages: pytorch-tabnet
Successfully installed pytorch-tabnet-4.0
```

```
import pytorch_tabnet
from pytorch_tabnet.tab_model import TabNetClassifier

import torch
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import roc_auc_score, accuracy_score
from sklearn.model_selection import KFold

np.random.seed(8)
```

```
clf1_nopreproc = TabNetClassifier(optimizer_fn=torch.optim.Adam,
                      optimizer_params=dict(lr=2e-2),
                      scheduler_params={"step_size":50, # how to use learning rate sch
                                        "gamma":0.9},
                      scheduler_fn=torch.optim.lr_scheduler.StepLR,
                      mask_type='entmax' # "sparsemax"
                     )
```

```
/usr/local/lib/python3.7/dist-packages/pytorch_tabnet/abstract_model.py:75: UserWarni
  warnings.warn(f"Device used : {self.device}")
```

```
1
```

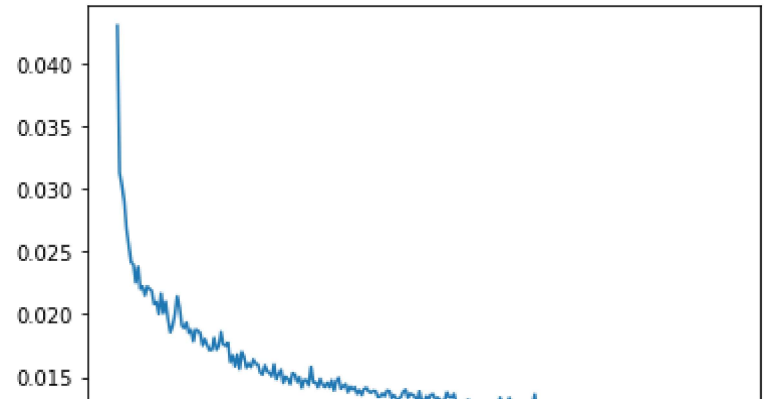
```
clf1_nopreproc.fit(
    X_train,y_train,
    eval_set=[(X_train, y_train), (X_test, y_test)],
    eval_name=['train', 'valid'],
    eval_metric=['accuracy'],
    max_epochs=1000 , patience=50,
    batch_size=1000, virtual_batch_size=500,
    num_workers=0,
    weights=1,
    drop_last=False
)
```

```
epoch 0  | loss: 0.04299 | train_accuracy: 0.98771 | valid_accuracy: 0.98792 |  0:0
epoch 1  | loss: 0.03128 | train_accuracy: 0.99281 | valid_accuracy: 0.99287 |  0:0
epoch 2  | loss: 0.03026 | train_accuracy: 0.99293 | valid_accuracy: 0.99303 |  0:0
epoch 3  | loss: 0.02903 | train_accuracy: 0.99273 | valid_accuracy: 0.99285 |  0:0
epoch 4  | loss: 0.02679 | train_accuracy: 0.99135 | valid_accuracy: 0.99126 |  0:0
epoch 5  | loss: 0.02553 | train_accuracy: 0.99341 | valid_accuracy: 0.99342 |  0:0
epoch 6  | loss: 0.02416 | train_accuracy: 0.99311 | valid_accuracy: 0.99336 |  0:0
epoch 7  | loss: 0.02396 | train_accuracy: 0.9942  | valid_accuracy: 0.99425 |  0:0
epoch 8  | loss: 0.02252 | train_accuracy: 0.97228 | valid_accuracy: 0.97184 |  0:0
epoch 9  | loss: 0.02379 | train_accuracy: 0.99383 | valid_accuracy: 0.99367 |  0:0
epoch 10 | loss: 0.02202 | train_accuracy: 0.99382 | valid_accuracy: 0.99395 |  0:0
epoch 11 | loss: 0.02222 | train_accuracy: 0.99406 | valid_accuracy: 0.99399 |  0:0
epoch 12 | loss: 0.0215  | train_accuracy: 0.98691 | valid_accuracy: 0.9872  |  0:0
epoch 13 | loss: 0.0222  | train_accuracy: 0.99418 | valid_accuracy: 0.99412 |  0:0
epoch 14 | loss: 0.02202 | train_accuracy: 0.99306 | valid_accuracy: 0.99298 |  0:0
epoch 15 | loss: 0.02185 | train_accuracy: 0.99398 | valid_accuracy: 0.99402 |  0:0
epoch 16 | loss: 0.02081 | train_accuracy: 0.99395 | valid_accuracy: 0.9939  |  0:0
epoch 17 | loss: 0.02097 | train_accuracy: 0.99428 | valid_accuracy: 0.99418 |  0:0
epoch 18 | loss: 0.02002 | train_accuracy: 0.99403 | valid_accuracy: 0.99405 |  0:0
epoch 19 | loss: 0.02165 | train_accuracy: 0.98938 | valid_accuracy: 0.98943 |  0:0
epoch 20 | loss: 0.02007 | train_accuracy: 0.99458 | valid_accuracy: 0.99461 |  0:0
epoch 21 | loss: 0.02101 | train_accuracy: 0.98228 | valid_accuracy: 0.98234 |  0:0
epoch 22 | loss: 0.01965 | train_accuracy: 0.99342 | valid_accuracy: 0.9937  |  0:0
epoch 23 | loss: 0.01858 | train_accuracy: 0.99387 | valid_accuracy: 0.99374 |  0:0
epoch 24 | loss: 0.01908 | train_accuracy: 0.99496 | valid_accuracy: 0.99506 |  0:
epoch 25 | loss: 0.02    | train_accuracy: 0.98728 | valid_accuracy: 0.98729 |  0:
epoch 26 | loss: 0.02144 | train_accuracy: 0.98873 | valid_accuracy: 0.98902 |  0:
epoch 27 | loss: 0.02051 | train_accuracy: 0.99464 | valid_accuracy: 0.99473 |  0:
epoch 28 | loss: 0.01912 | train_accuracy: 0.99542 | valid_accuracy: 0.99534 |  0:
epoch 29 | loss: 0.01891 | train_accuracy: 0.99484 | valid_accuracy: 0.99497 |  0:
```

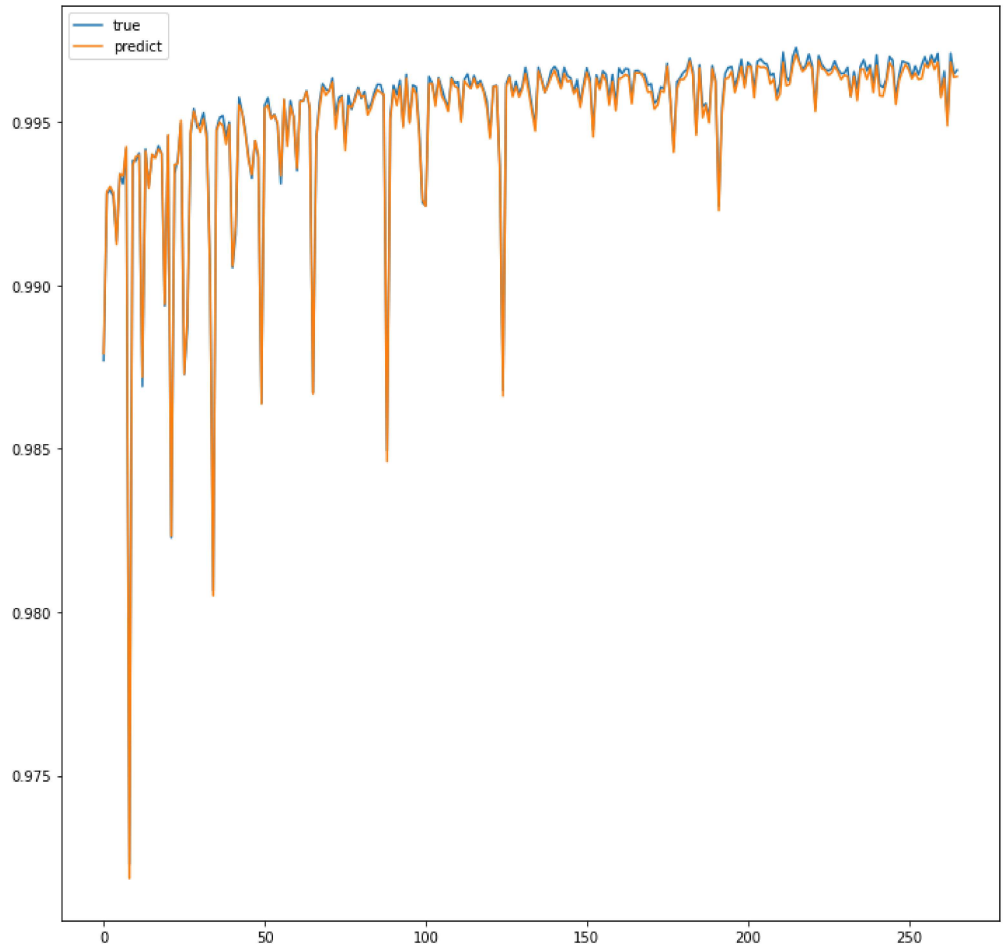
```
epoch 30 | loss: 0.01934 | train_accuracy: 0.99495 | valid_accuracy: 0.99469 |  0:
epoch 31 | loss: 0.01853 | train_accuracy: 0.99529 | valid_accuracy: 0.99511 |  0:
epoch 32 | loss: 0.01877 | train_accuracy: 0.99467 | valid_accuracy: 0.9945  |  0:
epoch 33 | loss: 0.01786 | train_accuracy: 0.9906  | valid_accuracy: 0.99048 |  0:
epoch 34 | loss: 0.01878 | train_accuracy: 0.98065 | valid_accuracy: 0.98051 |  0:
epoch 35 | loss: 0.01869 | train_accuracy: 0.99477 | valid_accuracy: 0.99474 |  0:
epoch 36 | loss: 0.01847 | train_accuracy: 0.99514 | valid_accuracy: 0.995   |  0:
epoch 37 | loss: 0.01751 | train_accuracy: 0.9952  | valid_accuracy: 0.99489 |  0:
epoch 38 | loss: 0.01802 | train_accuracy: 0.99454 | valid_accuracy: 0.99432 |  0:
epoch 39 | loss: 0.01754 | train_accuracy: 0.995   | valid_accuracy: 0.99493 |  0:
epoch 40 | loss: 0.01715 | train_accuracy: 0.99055 | valid_accuracy: 0.9906  |  0:
epoch 41 | loss: 0.01717 | train_accuracy: 0.99156 | valid_accuracy: 0.99185 |  0:
epoch 42 | loss: 0.01813 | train_accuracy: 0.99576 | valid_accuracy: 0.99553 |  0:
epoch 43 | loss: 0.01719 | train_accuracy: 0.99531 | valid_accuracy: 0.9953  |  0:
epoch 44 | loss: 0.01747 | train_accuracy: 0.99472 | valid_accuracy: 0.99463 |  0:
epoch 45 | loss: 0.01858 | train_accuracy: 0.99393 | valid_accuracy: 0.99385 |  0:
epoch 46 | loss: 0.01761 | train_accuracy: 0.99328 | valid_accuracy: 0.99339 |  0:
epoch 47 | loss: 0.01749 | train_accuracy: 0.99443 | valid_accuracy: 0.99444 |  0:
epoch 48 | loss: 0.01774 | train_accuracy: 0.99392 | valid_accuracy: 0.99403 |  0:
epoch 49 | loss: 0.01619 | train_accuracy: 0.9864  | valid_accuracy: 0.98637 |  0:
epoch 50 | loss: 0.0167  | train_accuracy: 0.99552 | valid_accuracy: 0.99543 |  0:
epoch 51 | loss: 0.01585 | train_accuracy: 0.99575 | valid_accuracy: 0.99552 |  0:
epoch 52 | loss: 0.01678 | train_accuracy: 0.99512 | valid_accuracy: 0.9951  |  0:
epoch 53 | loss: 0.01564 | train_accuracy: 0.99524 | valid_accuracy: 0.99522 |  0:
epoch 54 | loss: 0.017   | train_accuracy: 0.99492 | valid_accuracy: 0.99498 |  0:
epoch 55 | loss: 0.01658 | train_accuracy: 0.99311 | valid_accuracy: 0.99336 |  0:
epoch 56 | loss: 0.01574 | train_accuracy: 0.99569 | valid_accuracy: 0.9957  |  0:
```

```
1 import matplotlib.pyplot as plt
2 plt.plot(clf1_nopreproc.history['loss'])
```

```
[<matplotlib.lines.Line2D at 0x7fe4bf20dc50>]
```




```
1 plt.figure(figsize=(12,12))
2 plt.plot(clf1_nopreproc.history['train_accuracy'], label = 'true')
3 plt.plot(clf1_nopreproc.history['valid_accuracy'], label = 'predict')
4 plt.legend()
5 plt.show()
6
7 plt.savefig ("acuracia.png", dpi=300)
```




```
<Figure size 432x288 with 0 Axes>
```

```
1 preds = clf1_nopreproc.predict(X_test)
2 test_acc = accuracy_score(preds, y_test)
3
4 print(f"BEST ACCURACY SCORE ON TEST SET : {test_acc}")
```

```
BEST ACCURACY SCORE ON TEST SET : 0.9970751458854352
```

```
1 import xgboost as xgb
2 from sklearn.metrics import accuracy_score
```

```
1 data_dmatrix = xgb.DMatrix(data=X,label=y)
```

```
1 xg_reg = xgb.XGBClassifier(objective ='binary:logistic',
2                            colsample_bytree = 0.3,
3                            learning_rate = 0.1,
4                 max_depth = 5, alpha = 10, n_estimators = 10)
```

```
eval_set = [(X_train, y_train), (X_test, y_test)]
```

```
xg_reg.fit(X_train,y_train,eval_metric=["error"], eval_set=eval_set, verbose=False)

preds = xg_reg.predict(X_test)
```

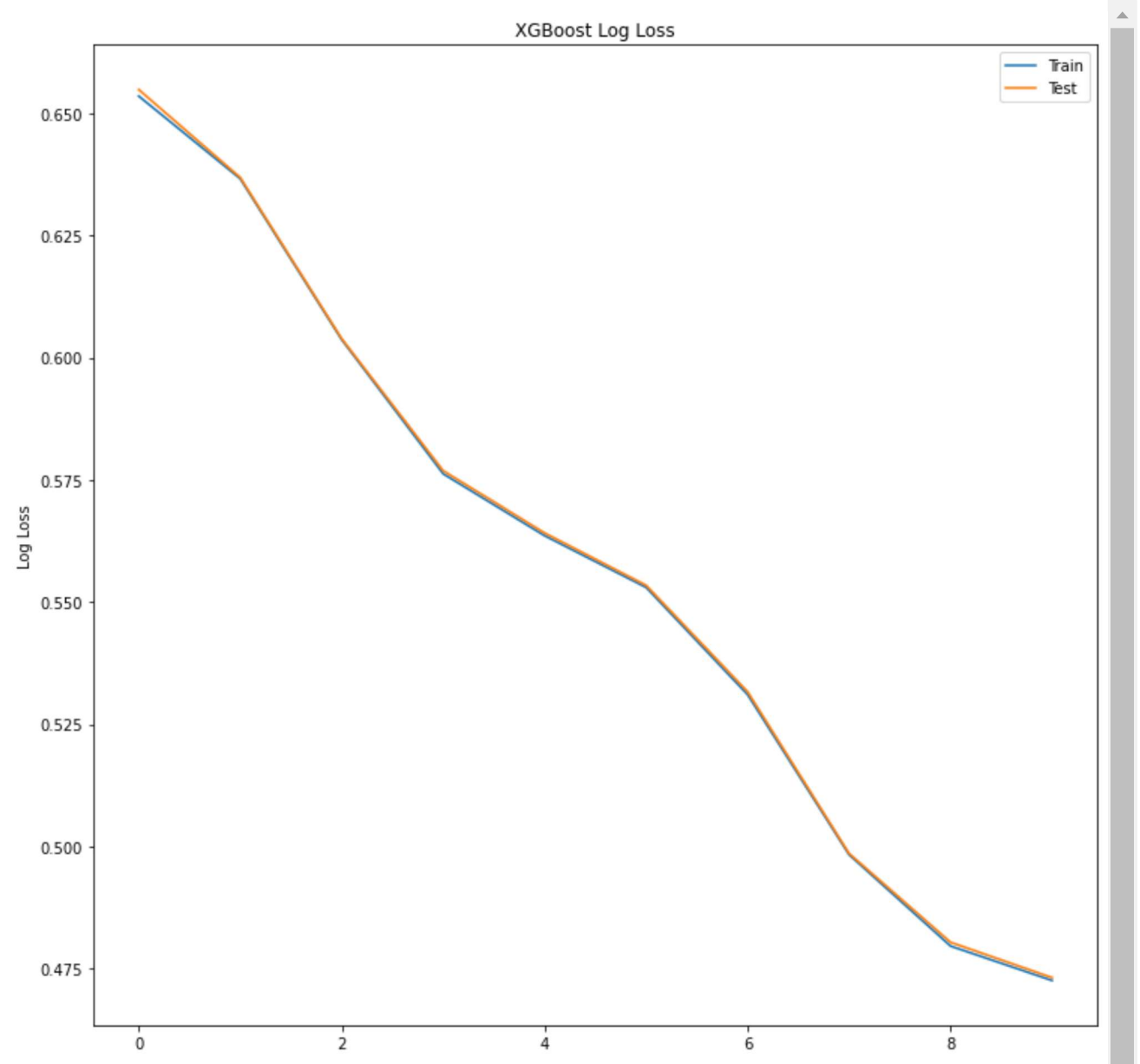
```
xg_reg.score(X_test,y_test)
```

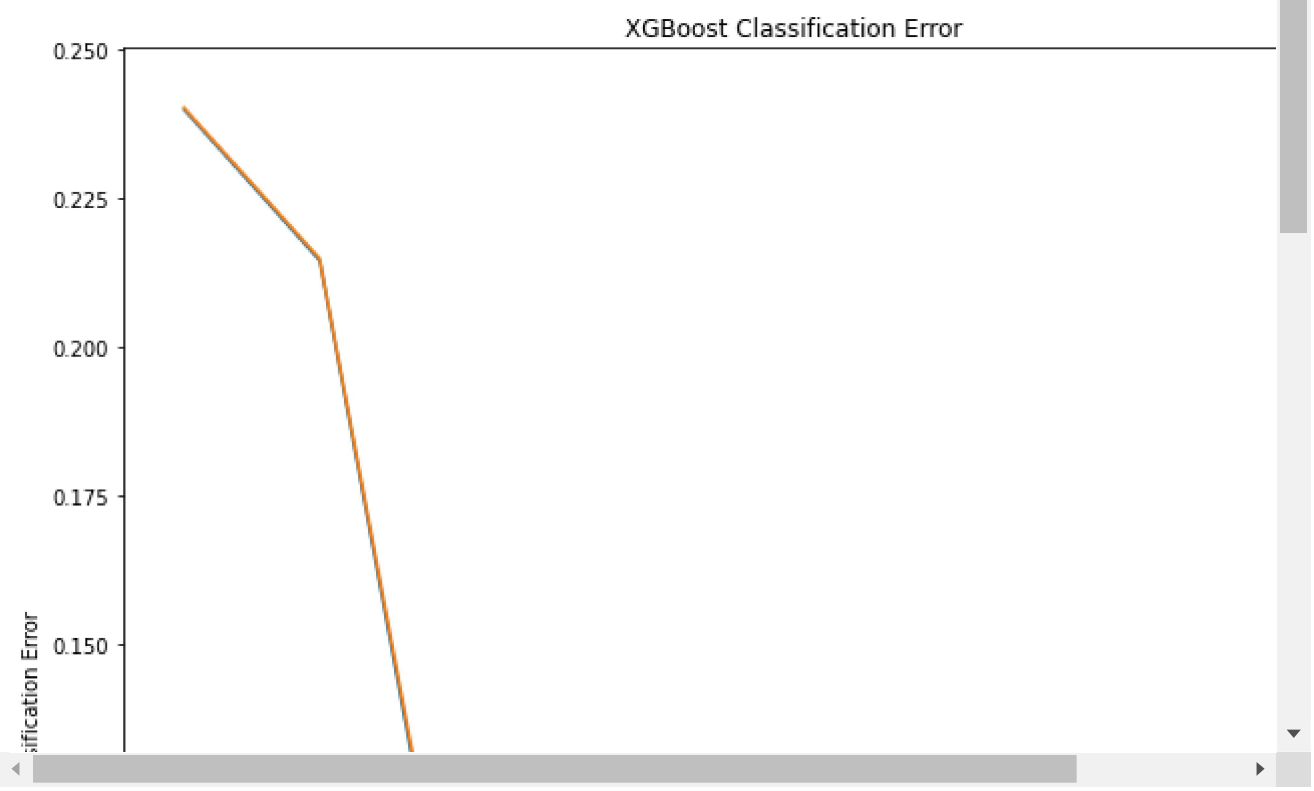
```
0.9558604263427415
```

```
y_pred = xg_reg.predict(X_test)
predictions = [round(value) for value in y_pred]

accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))

results = xg_reg.evals_result()
epochs = len(results["validation_0"]["error"])
x_axis = range(0, epochs)
```

```
Accuracy: 95.59%
```

```
# plot log loss
fig, ax = plt.subplots(figsize=(12,12))
ax.plot(x_axis, results["validation_0"]["logloss"], label="Train")
ax.plot(x_axis, results["validation_1"]["logloss"], label="Test")
ax.legend()
plt.ylabel("Log Loss")
plt.title("XGBoost Log Loss")
plt.show()

# plot classification error
fig, ax = plt.subplots(figsize=(12,12))
ax.plot(x_axis, results["validation_0"]["error"], label="Train")
ax.plot(x_axis, results["validation_1"]["error"], label="Test")
ax.legend()
plt.ylabel("Classification Error")
plt.title("XGBoost Classification Error")
plt.show()
```


XGBoost Log Loss
Train
Test
Log Loss
0.650
0.625
0.600
0.575
0.550
0.525
0.500
0.475
0
2
4
6
8


XGBoost Classification Error
0.250
0.225
0.200
0.175
0.150

APÊNDICE B – Código de implementação parte 2

```
from google.colab import drive
drive.mount('/content/drive')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
```

```
Mounted at /content/drive
```

```
#Dados
df1 = pd.read_csv('/content/drive/MyDrive/TCC/teste1.csv')
df2 = pd.read_csv('/content/drive/MyDrive/TCC/TT500.csv')
#Concatenando tabelas
df = pd.merge(df1, df2, how = 'left', on = 'MC1FC010V009 Time')
#Tratando dados
df = df.drop(columns = ['MC1FC010TT002 Time','MC1FC010MT510 Time'])
df = df.rename({'MC1FC010V009 Time': 'Time', 'MC1FC010V009 ValueY': 'Valve', 'MC1FC010T
#df = df.set_index(pd.DatetimeIndex(df['Time']))


```

```
df = df.interpolate()
```

```
df.sort_values(by=['Time'],inplace=True)
df['Time'] = pd.to_datetime(df['Time'])
#df.plot(x='Time', y=['Valve', 'Temperature', 'Humidity', 'Back_Temperature'])
#df
```

```
df = df.resample('60T', on = 'Time').mean()
```

```
df
```

Time	Valve	Temperature	Humidity	Back_Temperature
2020-01-28 15:00:00	30.961320	17.767250	59.953227	21.815987

```
cenarios = []

import datetime

Time_format = '%Y-%m-%d %H:%M:%S'
"bom cenário"
date = '2020-01-28 15:22:44'
cenarios.append(datetime.datetime.strptime(date, Time_format))
date = '2020-11-12 00:00:00'
cenarios.append(datetime.datetime.strptime(date, Time_format))

"cenário ruim"
date = '2020-11-12 00:01:00'
cenarios.append(datetime.datetime.strptime(date, Time_format))
date = '2021-05-15 00:00:00'
cenarios.append(datetime.datetime.strptime(date, Time_format))

"cenário pós conserto"
date = '2021-05-15 00:01:00'
cenarios.append(datetime.datetime.strptime(date, Time_format))
date = '2021-05-29 00:00:00'
cenarios.append(datetime.datetime.strptime(date, Time_format))
```

```
cenarios
```

```
[datetime.datetime(2020, 1, 28, 15, 22, 44),
 datetime.datetime(2020, 11, 12, 0, 0),
 datetime.datetime(2020, 11, 12, 0, 1),
 datetime.datetime(2021, 5, 15, 0, 0),
 datetime.datetime(2021, 5, 15, 0, 1),
 datetime.datetime(2021, 5, 29, 0, 0)]
```

```
aux = []
i = 0
aux = df[df.index > cenarios[i]]
aux = aux[aux.index < cenarios[i+1]]
aux
```

Time	Valve	Temperature	Humidity	Back_Temperature
2020-01-28 16:00:00	32.229765	18.304669	61.715554	22.048731
2020-01-28 17:00:00	26.595899	18.711721	63.039339	21.859085
2020-01-28 18:00:00	25.024177	19.824671	66.926660	22.037850
2020-01-28 19:00:00	10.292481	21.572778	66.956620	22.414460

```
classe = {'Group':np.zeros(len(aux),dtype=int)}
classe = pd.DataFrame(classe)
base = pd.DataFrame(aux)
```

```
i = 2
j = 2

while i < len(cenarios):
  if i == 2:
    aux = df[df.index > cenarios[i]]
    aux = aux[aux.index < cenarios[i+1]]
    aux1 = {'Group':1*np.ones(len(aux),dtype=int)}
    aux1 = pd.DataFrame(aux1)
    classe = pd.concat([classe,aux1])
    base = pd.concat([base,aux])
    i = i+2

  elif i == 4:
    aux = df[df.index > cenarios[i]]
    aux = aux[aux.index < cenarios[i+1]]
    aux1 = {'Group':0*np.ones(len(aux),dtype=int)}
    aux1 = pd.DataFrame(aux1)
    classe = pd.concat([classe,aux1])
    base = pd.concat([base,aux])
    i = i+2
```

```
aux
```

Time	Valve	Temperature	Humidity	Back_Temperature
2021-05-15 01:00:00	0.000000	22.591176	50.015311	22.200972
2021-05-15 02:00:00	0.000000	22.482729	49.694010	22.106753
2021-05-15 03:00:00	0.000000	22.392096	49.565369	22.019495
2021-05-15 04:00:00	0.000000	22.354992	49.379882	21.952793

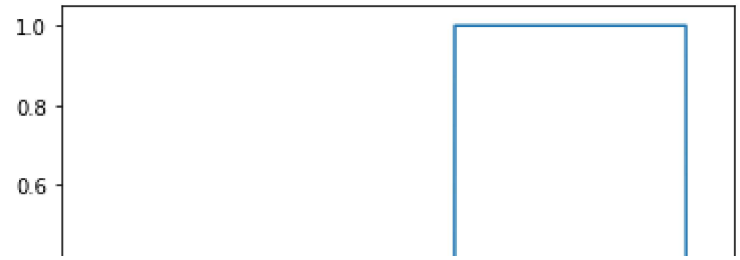
```
1 base
```

Time	Valve	Temperature	Humidity	Back_Temperature
2020-01-28 16:00:00	32.229765	18.304669	61.715554	22.048731
2020-01-28 17:00:00	26.595899	18.711721	63.039339	21.859085
2020-01-28 18:00:00	25.024177	19.824671	66.926660	22.037850
2020-01-28 19:00:00	10.292481	21.572778	66.956620	22.414460
2020-01-28 20:00:00	0.002314	24.263358	63.143084	23.568823
...	...	...	...	...
2021-05-28				

```
1 classe.reindex()
2 classe.set_index(base.index,inplace=True)
```

```
1 classe
2
3 plt.plot(classe)
```

```
[<matplotlib.lines.Line2D at 0x7f86c0493d90>]
```




```
#classe = classe.astype(dtype=float)
base['classe'] = classe
#base.reindex()
#base.set_index('Time',inplace=True)
```

```
base
```

Time	Valve	Temperature	Humidity	Back_Temperature	cla
2020-01-28 16:00:00	32.229765	18.304669	61.715554	22.048731	
2020-01-28 17:00:00	26.595899	18.711721	63.039339	21.859085	
2020-01-28 18:00:00	25.024177	19.824671	66.926660	22.037850	
2020-01-28 19:00:00	10.292481	21.572778	66.956620	22.414460	

```
from sklearn.preprocessing import

from keras.utils import np_utils
from sklearn.model_selection import train_test_split
labelencoder = LabelEncoder()
classe = labelencoder.fit_transform(base['classe'])

X = base.iloc[:, 0:4]
y = base.iloc[:, 4]

```

```
y = np.array(y)
```

```
1 y
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
1 X = np.array(X)
```

```
1 X
```

```
array([[32.2297651 , 18.30466922, 61.71555411, 22.04873133],
       [26.59589866, 18.71172094, 63.039339  , 21.85908505],
       [25.02417688, 19.82467073, 66.92666016, 22.03785035],
       ...,
       [31.37152357, 20.320939  , 53.45413678, 22.02344306],
       [39.12511152, 19.69021241, 52.01648038, 22.03504731],
       [30.53700489, 19.33695918, 51.88801989, 21.7478578 ]])
```

```
1 # vamos deixar 70% para treino e 30% para teste.
2 train_size = int(len(base) * 0.7)
3 test_size = len(base) - train_size
4 train, test = base.iloc[0:train_size,:], base.iloc[train_size:len(base),:]
5 print(len(train), len(test))
```

```
8163 3499
```

```
1 dias = 24*3
```

```
1 #função para criar os conjuntos de dados de treino
2 def create_dataset(dataset, look_back=dias):
3     dataX, dataY = [], []
4     for i in range(len(dataset) - look_back):
5         a = dataset.iloc[i:(i + look_back), 0:4]
6         dataX.append(a)
7         dataY.append(dataset.iloc[i + look_back, 4])
8     print(len(dataY))
9     return np.array(dataX), np.array(dataY)
```

```
1 look_back = dias
2 trainX, trainY = create_dataset(train, look_back)
3 testX, testY = create_dataset(test, look_back)
```

```
8091
3427
```

```
1 import tensorflow as tf
2 from tensorflow.keras.models import Sequential
3 from tensorflow.keras.layers import Dense
4 from tensorflow.keras.layers import LSTM
5 from tensorflow.keras.layers import RNN
```

```
1 input_size = (trainX.shape[1], trainX.shape[2])
```

```
model = tf.keras.Sequential()

model.add(tf.keras.layers.LSTM(120, input_shape = input_size,
                               return_sequences=False))
model.add(tf.keras.layers.Dense(1, activation="sigmoid"))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 120)               60000

 dense (Dense)               (None, 1)                 121

=================================================================
Total params: 60,121
Trainable params: 60,121
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss='binary_crossentropy',
              optimizer='Adam',
              metrics="accuracy")
```

```
#treine o modelo
history = model.fit(trainX,trainY, epochs=20, validation_data = (testX, testY))
```

```
Epoch 1/20
253/253 [==============================] - 26s 94ms/step - loss: 0.0508 - accuracy: 0
Epoch 2/20
253/253 [==============================] - 23s 91ms/step - loss: 8.9909e-04 - accurac
Epoch 3/20
253/253 [==============================] - 22s 86ms/step - loss: 0.0032 - accuracy: 0
Epoch 4/20
253/253 [==============================] - 27s 108ms/step - loss: 5.0719e-04 - accura
Epoch 5/20
253/253 [==============================] - 26s 102ms/step - loss: 2.8797e-04 - accura
Epoch 6/20
253/253 [==============================] - 22s 86ms/step - loss: 3.4413e-04 - accurac
Epoch 7/20
253/253 [==============================] - 24s 93ms/step - loss: 1.3192e-04 - accurac
Epoch 8/20
253/253 [==============================] - 23s 92ms/step - loss: 0.0023 - accuracy: 0
Epoch 9/20
253/253 [==============================] - 22s 86ms/step - loss: 4.8700e-04 - accurac
Epoch 10/20
253/253 [==============================] - 22s 86ms/step - loss: 4.3639e-04 - accurac
Epoch 11/20
253/253 [==============================] - 22s 87ms/step - loss: 0.0023 - accuracy: 0
Epoch 12/20
253/253 [==============================] - 22s 86ms/step - loss: 3.5216e-04 - accurac
Epoch 13/20
253/253 [==============================] - 24s 94ms/step - loss: 3.6796e-04 - accurac
Epoch 14/20
```

```
253/253 [==============================] - 22s 86ms/step - loss: 3.3541e-04 - accurac
Epoch 15/20
253/253 [==============================] - 22s 86ms/step - loss: 2.3616e-04 - accurac
Epoch 16/20
253/253 [==============================] - 22s 86ms/step - loss: 7.2299e-05 - accurac
Epoch 17/20
253/253 [==============================] - 22s 86ms/step - loss: 6.2808e-04 - accurac
Epoch 18/20
253/253 [==============================] - 22s 86ms/step - loss: 2.7648e-04 - accurac
Epoch 19/20
253/253 [==============================] - 24s 94ms/step - loss: 7.5815e-05 - accurac
Epoch 20/20
253/253 [==============================] - 22s 87ms/step - loss: 1.5136e-04 - accurac
```

```
1 import plotly.offline as py
2 import plotly.graph_objs as go
3 from matplotlib import pyplot
```

```
1 # plote as curvas, valor real e valor predito no mesmo gráfico
2 yhat = model.predict(testX)
3 pyplot.figure(figsize=(12,12))
4 pyplot.plot(yhat, label='predict')
5 pyplot.plot(testY, label='true')
6 pyplot.legend()
7 pyplot.show()
```

```
108/108 [==============================] - 3s 29ms/step
```


```
1 # plote a curva de LOSS
2 pyplot.plot(history.history['loss'], label='train')
3 pyplot.plot(history.history['val_loss'], label='test')
4 pyplot.legend()
5 pyplot.show()
```