UNIVERSIDADE FEDERAL DA BAHIA

ESCOLA POLITÉCNICA

DEPARTAMENTO DE ENGENHARIA MECÂNICA

TRABALHO DE CONCLUSÃO DE CURSO

GABRIEL VERAS LIMA

MANUTENÇÃO PREDITIVA NA INDÚSTRIA: DESENVOLVIMENTO E COMPARAÇÃO DE MODELOS DE *MACHINE LEARNING* PARA PREVISÃO DE FALHAS

SALVADOR

2024

GABRIEL VERAS LIMA

MANUTENÇÃO PREDITIVA NA INDÚSTRIA: DESENVOLVIMENTO E COMPARAÇÃO DE MODELOS DE *MACHINE LEARNING* PARA PREVISÃO DE FALHAS

Trabalho de Conclusão de Curso de Graduação em Engenharia Mecânica apresentado à Escola Politécnica da Universidade Federal da Bahia, como requisito para obtenção do grau de Bacharel em Engenharia Mecânica.

Orientador: Prof. Dr. Bruno da Cunha Diniz

SALVADOR

2024

UNIVERSIDADE FEDERAL DA BAHIA
ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA MECÂNICA
TRABALHO DE CONCLUSÃO DE CURSO

GABRIEL VERAS LIMA

MANUTENÇÃO PREDITIVA NA INDÚSTRIA: DESENVOLVIMENTO E COMPARAÇÃO DE MODELOS DE *MACHINE LEARNING* PARA PREVISÃO DE FALHAS

Trabalho de Conclusão de Curso de Graduação em Engenharia Mecânica apresentado à Escola Politécnica da Universidade Federal da Bahia, como requisito para obtenção do grau de Bacharel em Engenharia Mecânica.

Aprovado em: 04 de Setembro de 2024

Banca Examinadora

_____

Prof. Dr. Bruno da Cunha Diniz

Universidade Federal da Bahia

_____

Prof. Dr Antônio Carlos Lopes Fernandes Júnior

Universidade Federal da Bahia

_____

Prof. Dr Vergilio Torezan Silingardi Del Claro

Universidade Federal da Bahia

RESUMO

A manutenção preditiva desempenha um papel vital na indústria moderna, sendo essencial para a antecipação de falhas em equipamentos, o que, por sua vez, contribui para a maximização da eficiência operacional e a minimização de custos associados a reparos inesperados e paradas não planejadas. A identificação antecipada de falhas não só previne interrupções nas operações, mas também prolonga a vida útil dos equipamentos, resultando em uma gestão mais eficiente dos recursos. Este trabalho visa contribuir para essa área, apresentando o desenvolvimento de um modelo preditivo de falhas em equipamentos, utilizando o conjunto de dados *AI4I 2020 Predictive Maintenance Dataset*, disponível no *UC Irvine Machine Learning Repository*. A abordagem adotada incluiu técnicas de engenharia de *features* e uma análise exploratória detalhada dos dados, garantindo a qualidade das variáveis utilizadas nos modelos. Como ponto de partida, foi empregada uma regressão logística como *baseline*, seguida pela aplicação de modelos mais sofisticados, como *Random Forest* e XGBoost, com o objetivo de aprimorar a performance preditiva. A análise comparativa dos resultados destacou o *XGBoost* como o modelo mais eficiente, apresentando métricas superiores de *F1-score* e ROC AUC. Este estudo reforça a importância da utilização de algoritmos de *machine learning* manutenção preditiva como uma estratégia essencial para a detecção antecipada de falhas, oferecendo uma solução robusta que pode resultar em reduções significativas de custos, aumento da confiabilidade operacional e mitigação de riscos em ambientes industriais.

Palavras-chave: Manutenção Preditiva, Aprendizado de Máquina, Modelo Preditivo, *XGBoost*, Regressão Logística, *Random Forest*

ABSTRACT

Predictive maintenance plays a vital role in modern industry, being essential for anticipating equipment failures, which in turn contributes to maximizing operational efficiency and minimizing costs associated with unexpected repairs and unplanned downtime. Early identification of failures not only prevents interruptions in operations, but also extends the useful life of equipment, resulting in more efficient management of resources. This work aims to contribute to this area by presenting the development of a predictive model for equipment failures, using the AI4I 2020 Predictive Maintenance Dataset, available at the UC Irvine Machine Learning Repository. The approach adopted included feature engineering techniques and a detailed exploratory analysis of the data, ensuring the quality of the variables used in the models. As a starting point, logistic regression was used as a baseline, followed by the application of more sophisticated models, such as Random Forest and XGBoost, with the aim of improving predictive performance. The comparative analysis of the results highlighted XGBoost as the most efficient model, with superior F1-score and ROC AUC metrics. This study reinforces the importance of using machine learning algorithms for predictive maintenance as an essential strategy for early failures detection, offering a robust solution that can result in significant cost reductions, increased operational reliability and risk mitigation in industrial environments.

Keywords: Predictive Maintenance, Machine Learning, Predictive Model, XGBoost, Logistic Regression, Random Forest.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

A manutenção preditiva tem se tornado um componente essencial na gestão de operações industriais, permitindo a identificação antecipada de falhas em equipamentos e, consequentemente, a redução de custos e tempos de inatividade. A manutenção preditiva utiliza dados em tempo real para prever falhas antes que elas ocorram, reduzindo custos de manutenção e melhorando a eficiência operacional. Este tipo de manutenção é especialmente relevante em indústrias que dependem de equipamentos críticos, onde a falha inesperada pode resultar em paradas prolongadas e prejuízos significativos. Além disso, a manutenção preditiva não só aumenta a disponibilidade e confiabilidade dos ativos, mas também contribui para um ambiente de trabalho mais seguro, ao minimizar o risco de falhas catastróficas (Jardine *et al*, 2006).

O avanço das técnicas de aprendizado de máquina nos últimos anos possibilitou o desenvolvimento de modelos preditivos capazes de analisar grandes volumes de dados e fornecer *insights* valiosos para a tomada de decisões. A capacidade dessas técnicas de identificar padrões complexos e prever comportamentos futuros com alta precisão tem revolucionado a maneira como a manutenção é gerenciada, promovendo uma transição de abordagens tradicionais, como a manutenção preventiva, para estratégias mais dinâmicas e adaptativas (Zhao & Zuo, 2012).

Com o uso de aprendizado de máquina, a manutenção preditiva se tornou ainda mais precisa e eficiente. Essa tecnologia permite que as empresas substituam estratégias de manutenção reativas ou preventivas por abordagens baseadas na condição real dos equipamentos. Isso não apenas melhora a eficiência operacional, mas também contribui para a longevidade dos ativos e a redução de falhas catastróficas. Além disso, a manutenção preditiva baseada em aprendizado de máquina pode ser continuamente aprimorada à medida que mais dados são coletados e analisados, permitindo uma adaptação rápida às mudanças nas condições operacionais e nos padrões de falhas.

A aplicação de aprendizado de máquina em manutenção preditiva envolve várias etapas, desde a coleta e processamento de dados até a seleção e ajuste de modelos preditivos. A engenharia de *features*, que consiste na criação e seleção de variáveis relevantes a partir dos dados, desempenha um papel crucial na melhoria da qualidade dos modelos preditivos. Assim como uma análise exploratória dos dados é de grande importância para compreender as características dos dados e identificar padrões que possam indicar os tipos de modelos que terão melhor desempenho. Esse processo é fundamental para garantir que os modelos desenvolvidos sejam capazes de capturar as nuances do comportamento dos equipamentos e fornecer previsões precisas.

Modelos de *ensembles* de árvores de decisão como *Random Forest* e *XGBoost* têm se mostrado particularmente eficazes na detecção de padrões que indicam falhas iminentes. A análise comparativa dos resultados de diferentes modelos permite identificar a abordagem mais eficaz para o problema especifico, junto a isso, é preciso utilizar técnicas rigorosas de validação para garantir que as previsões sejam precisas e confiáveis. Além disso, a escolha adequada do modelo não só impacta a qualidade das previsões, mas também a capacidade de generalização para novos dados, fator essencial para a aplicação prática em ambientes industriais.

1.2 Objetivo Geral

O objetivo geral deste trabalho é desenvolver e avaliar um modelo preditivo capaz de prever falhas em equipamentos industriais, utilizando o *AI4I 2020 Predictive Maintenance Dataset*. Esse conjunto de dados, embora sintético, reflete cenários reais de manutenção preditiva encontrados na indústria, oferecendo um contexto realista para a aplicação das técnicas de aprendizado de máquina.

1.3 Objetivos Específicos

Para atingir o objetivo geral, foram estabelecidos os seguintes objetivos específicos:

a) Realizar técnicas de engenharia de *features*: Melhorar a qualidade das variáveis preditivas, aumentando a eficácia dos modelos.

b) Realizar uma análise exploratória dos dados (EDA): Compreender as características dos dados e identificar padrões relevantes.

c) Desenvolver e validar modelos preditivos: Implementar modelos de regressão logística, *Random Forest* e *XGBoost*, comparando seu desempenho para determinar a melhor abordagem.

d) Analisar as métricas de desempenho dos modelos: Avaliar os modelos utilizando métricas como precisão, *recall*, *F1-score* e ROC AUC, para encontrar o melhor modelo que garanta robustez e confiabilidade dos resultados.

1.4 Estrutura do Trabalho

Este trabalho está estruturado em capítulos que conduzem o leitor desde o contexto teórico até os resultados práticos e conclusões.

No segundo capítulo, é explorado o referencial teórico, abordando conceitos essenciais de técnicas de aprendizado de máquina e métricas de validação. O terceiro capítulo detalha a metodologia utilizada, desde a obtenção e preparação dos dados até a avaliação dos modelos preditivos.

O quarto capítulo apresenta os resultados obtidos, acompanhados de uma análise comparativa dos modelos testados. No quinto e último capítulo, são discutidas as conclusões do estudo, junto a sugestões para trabalhos futuros.

O trabalho é concluído com a lista de referências bibliográficas, seguida pelo anexo que contém o código utilizado para todo o desenvolvimento e avaliação dos modelos preditivos.

2 REFERENCIAL TEÓRICO

2.1 Introdução ao Aprendizado de máquina

Aprendizado de máquina (*Machine Learning*) é um subconjunto da inteligência artificial que permite que as máquinas aprendam a partir de dados sem serem explicitamente programados (Alpaydin, 2020; NG, 2022). Ao contrário das abordagens tradicionais de resolução de problemas, os sistemas de aprendizado de máquina utilizam os dados para identificar padrões, tomar decisões e melhorar continuamente o seu desempenho, aprendendo com os próprios dados. Através da utilização de diferentes abordagens como a aprendizagem supervisionada e não supervisionada, as máquinas podem melhorar as suas capacidades de forma adaptativa à medida que recebem mais exemplos de aprendizagem. O objetivo do aprendizado de máquina é equipar as máquinas com a capacidade de resolver problemas complexos, partindo dos dados e de conhecimentos base, tornando-o uma ferramenta poderosa com aplicações em diversos domínios, como predição de falhas, medicina e visão computacional (Murphy, 2012; Ng, 2022; Géron, 2024)

Neste trabalho será utilizado o aprendizado supervisionado, que envolve o treino de modelos utilizando dados rotulados, em que os resultados corretos são fornecidos juntamente com os dados de entrada, permitindo às máquinas prever resultados com base nos dados de treino (Hastie, Tibshirani, & Friedman, 2009; Ng, 2022; Géron, 2024).

Os problemas de aprendizado supervisionado geralmente se dividem em duas categorias principais(Ng, 2022; Géron, 2024):

a) **Classificação:** Onde a tarefa é atribuir rótulos a instâncias de entrada a partir de um conjunto finito de classes. Exemplo: classificação de falhas como "falha" ou "não falha".

b) **Regressão:** Onde a tarefa é prever um valor contínuo. Exemplo: prever o preço de uma casa com base em características como tamanho, localização e número de quartos.

Vários algoritmos podem ser utilizados para aprendizado supervisionado, como:

a) Regressão Linear e Logística
b) Árvores de Decisão
c) Máquinas de Vetores de Suporte (SVM)
d) k-Nearest Neighbors (k-NN)
e) Redes Neurais

Neste trabalho o problema é de classificação e foram utilizados apenas regressão logística e árvores de decisão, mas também será abordado sobre regressão linear, pois ela funciona como base para a regressão logística.

2.2 Regressão Linear

A regressão linear é uma técnica utilizada para modelar a relação entre uma variável resposta contínua e uma ou mais variáveis explicativas. O objetivo é ajustar uma linha reta (ou hiperplano no caso de múltiplas variáveis) que melhor descreva a relação entre as variáveis (Seber & Lee, 2012; Ng, 2022; Géron, 2024).

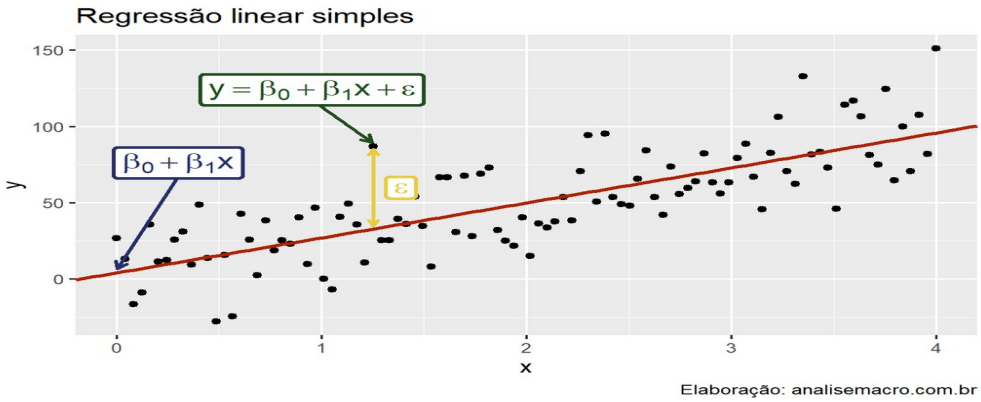
O modelo de regressão linear simples é representado pela equação 2.1.

$$Y = \beta_0 + \beta_1 x + \varepsilon \tag{2.1}$$

onde, Y é a variável resposta, β_0 o intercepto, β_1 o coeficiente angular, x a variável explicativa e ε é o erro.

O ajuste de uma regressão linear simples é apresentado na figura 2.1.

Figura 2.1 - Ajuste de uma regressão linear simples



Fonte: Análise Macro, 2023

Muitas vezes os problemas são mais complexos que isso e existirão múltiplas variáveis explicativas e coeficientes angulares, se tornando uma regressão linear múltipla (Ng, 2022).

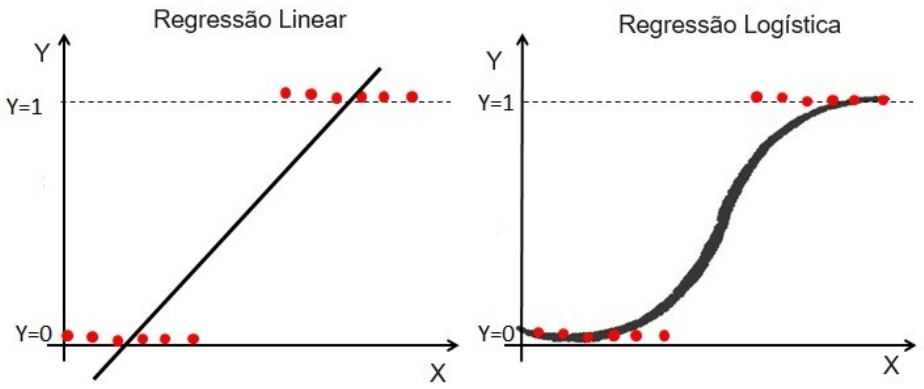
A regressão linear é amplamente utilizada em:

a) Previsão de preços.
b) Análise de tendência.
c) Avaliação de impacto de variáveis explicativas.

2.3 **Regressão Logística**

A regressão logística tem como base a regressão linear e é utilizada para modelar a probabilidade de um evento binário ocorrer. Diferentemente da regressão linear, que prevê valores contínuos, a regressão logística prevê a probabilidade de classes binárias (1 ou 0). Ela utiliza a função sigmoide na equação da regressão linear para atingir a transformação desejada, como observado na figura 2.2 (Hosmer, Lemeshow, & Sturdivant, 2013; Ng, 2022; Géron, 2024).

Figura 2.2 - Transformação da função sigmoide



Fonte: DataCamp, 2024

A função sigmoide da regressão logística é apresentada de acordo com a equação 2.2.

$$P(Y=1)=\frac{1}{1+e^{-g(x)}} \tag{2.2}$$

onde, *P(Y=1)* é a probabilidade do evento ocorrer e *g(x)* é a equação de uma regressão linear.

Para realizar a predição em valores binários ao invés de uma probabilidade é utilizado um limiar de decisão, logo, quando a probabilidade for maior do que limiar de decisão o valor previsto será 1, caso contrário será 0 (Ng, 2022; Géron, 2024)

Geralmente utiliza-se um valor padrão de 0,5 para o limiar de decisão, porém é possível utilizar qualquer valor desejado para esse limiar de forma a atingir a melhor performance do modelo (Ng, 2022; Géron, 2024)

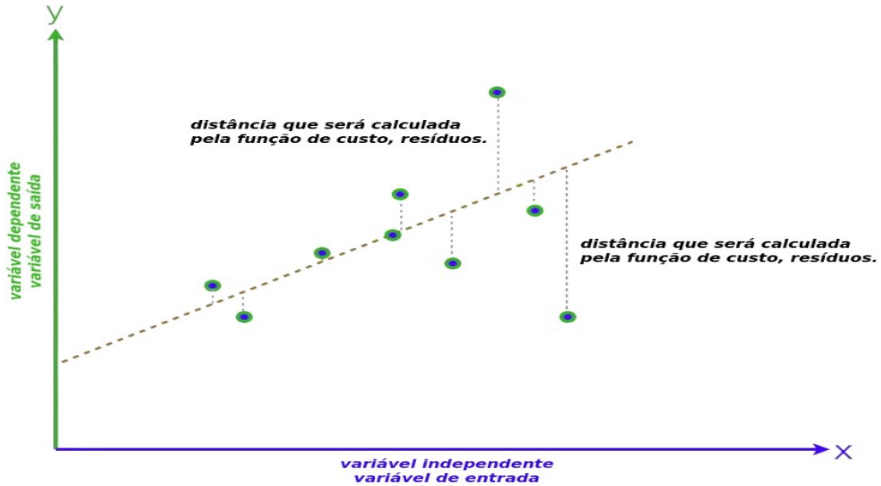
A regressão logística é amplamente utilizada em problemas como:

a) Predição de falhas.
b) Diagnóstico médico (presença ou ausência de doença).
c) Classificação de e-mails (spam ou não spam).
d) Modelagem de risco de crédito.

2.4 **Função de Custo**

A função de custo é utilizada para medir a diferença entre os valores previstos pelo modelo e os valores reais dos dados. O objetivo do treinamento do modelo é minimizar essa diferença utilizando funções como o Erro Quadrático Médio (MSE) e a *Log-Loss* (Murphy, 2012; Ng, 2022; Géron, 2024), pode ser observado na figura 2.3 como a diferença é calculada.

Figura 2.3 - Distância utilizada na função custo



Fonte: Tableless, 2019

Na regressão linear a função de custo mais comum é o Erro Quadrático Médio (*MSE - Mean Squared Error*) representado pela equação 2.3 (Ng, 2022; Géron, 2024).

$$J = \frac{1}{m} \sum_{i=1}^{m} \left(Yi - \overline{Yi}\right)^2 \tag{2.3}$$

onde, *m* é o número de exemplos, *Yi* é o valor real e $\overline{Yi}$ é o valor previsto *Yi*.

Já na regressão logística a função de custo mais comum é a de Entropia Cruzada ou *Log-Loss*, representada pela equação 2.4 (Ng, 2022; Géron, 2024).

$$J = -\frac{1}{m}\sum_{i=1}^{m}\left[Yi\log(\overline{Yi}) + (1-Yi)\log(1-\overline{Yi})\right] \tag{2.4}$$

onde, *m* é o número de exemplos, *Yi* é a classe real (0 ou 1) e $\overline{Yi}$ a probabilidade prevista.

2.5 **Gradiente Descendente**

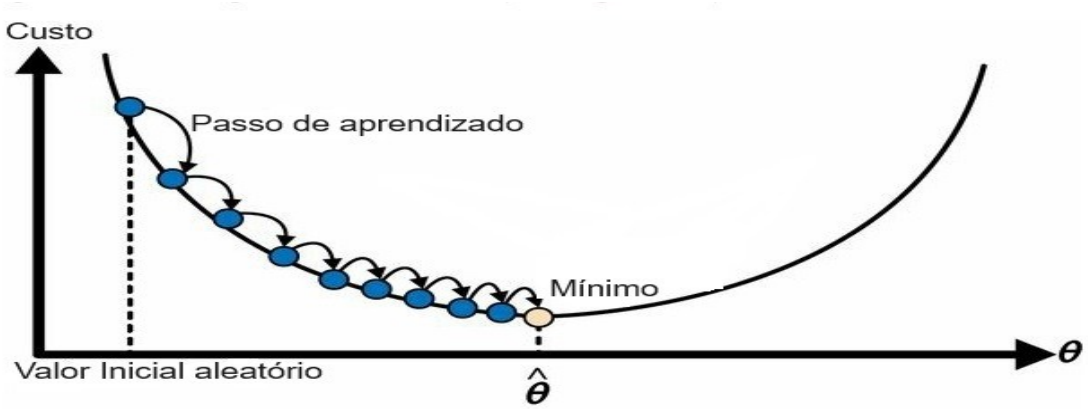
O gradiente descendente é um algoritmo de otimização utilizado para encontrar os valores dos parâmetros β que minimizam a função de custo. É uma técnica iterativa que ajusta os parâmetros na direção negativa do gradiente da função de custo (Ruder, 2016; Ng, 2022; Géron, 2024).

$$\beta novo_J = \beta_J - \alpha\frac{\partial J}{\partial \beta_J} \tag{2.5}$$

onde, βj são os parâmetros do modelo, α é a taxa de aprendizado e $\frac{\partial J}{\partial \beta j}$ *é* o gradiente da função de custo em relação a βj.

O gradiente, derivada da função custo em relação ao parâmetro, é a inclinação da reta tangente à curva nesse ponto, logo, ao multiplicá-lo por uma taxa de aprendizado α e subtrair esse valor do valor inicial do parâmetro repetidas vezes, são dados pequenos passos até encontrar o ponto mínimo da função, isto é realizado para todos os parâmetros existentes na função (NG, 2022). A figura 2.4 demonstra o funcionamento do gradiente descendente, onde θ é o vetor com todos os parâmetros.

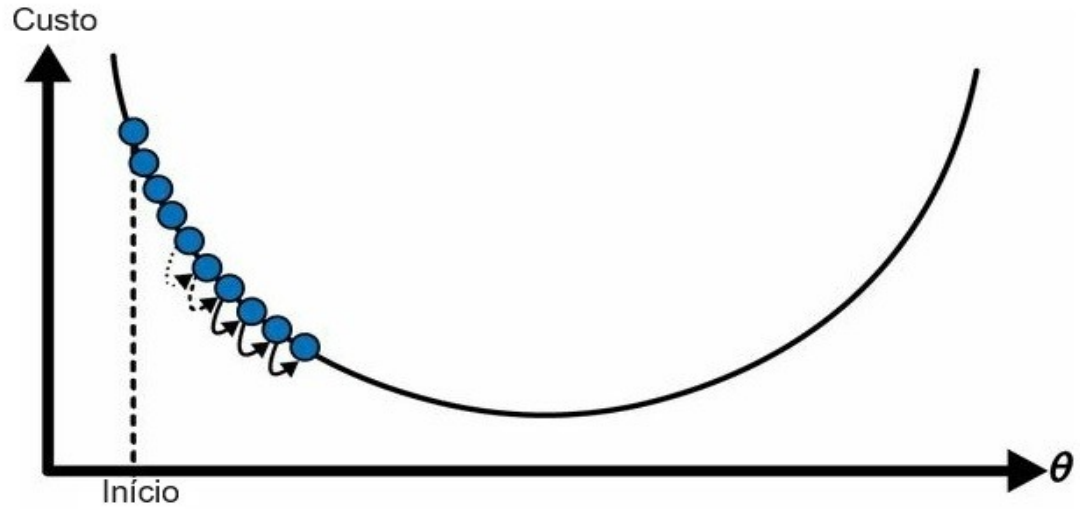
Figura 2.4 - Gradiente descendente



Fonte: Géron, 2024.

A taxa de aprendizado é o tamanho dos passos dados pelo gradiente descendente e é um fator importante, pois se ela for muito pequena o modelo irá demorar no seu processo de aprendizado, como apresentado na figura 2.5 (Ng, 2022; Géron, 2024).

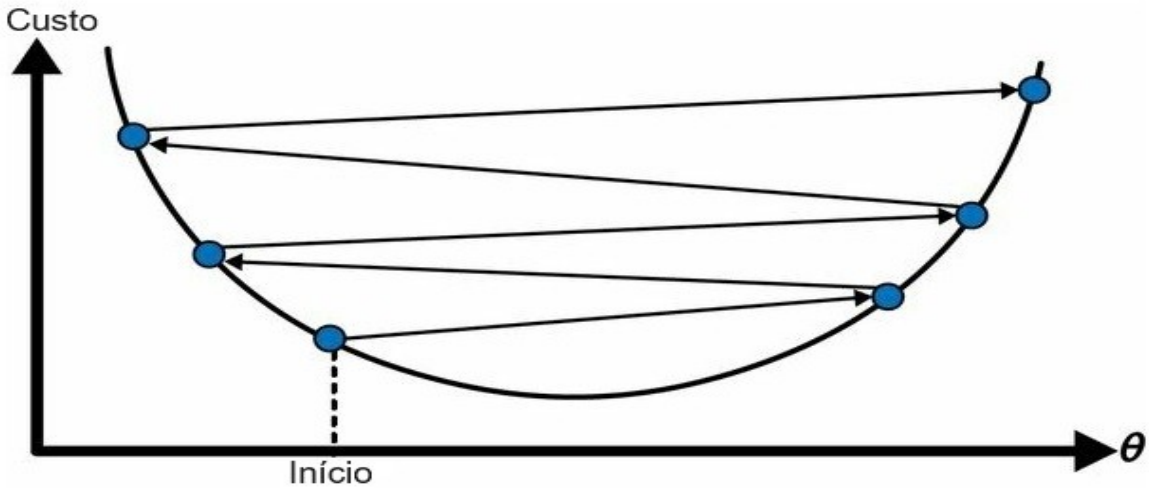
Figura 2.5 - Gradiente descendente com valor de α muito pequeno



Fonte: Géron, 2024.

Por outro lado, caso a taxa seja muito alta você pode passar do ponto mínimo e acabar em um ponto mais alto do que estava antes, fazendo com que o algoritmo divirja e não encontre uma boa solução, como apresentado na figura 2.6 (Ng, 2022; Géron, 2024).

Figura 2.6 - Gradiente descendente com valor de α muito alto



Fonte: Géron, 2024.

Além disso, existem diferentes tipos de gradiente descendentes(Ruder, 2016; Géron, 2024):

a) **Batch Gradient Descent:** Calcula o gradiente usando todo o conjunto de dados.

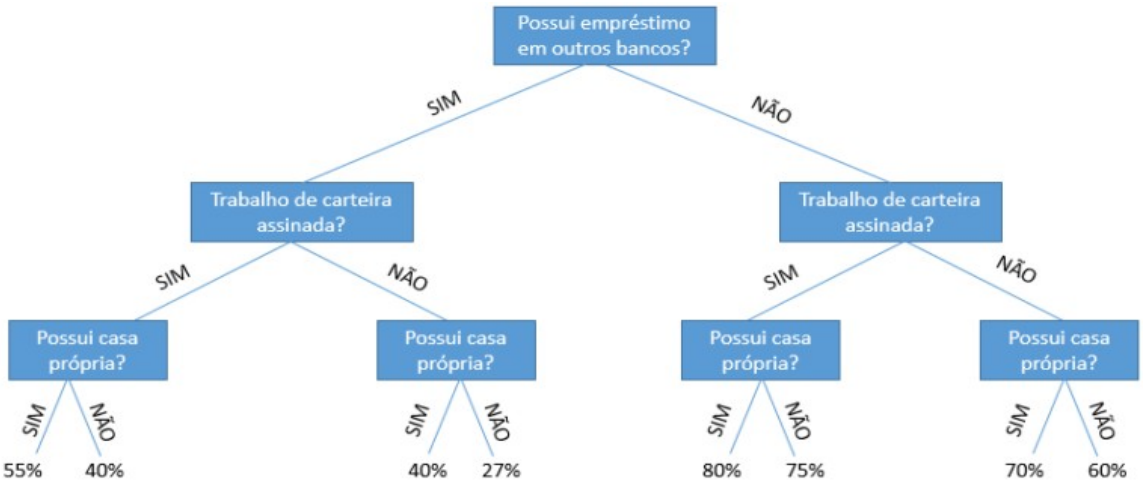
b) **Stochastic Gradient Descent (SGD):** Calcula o gradiente usando um único exemplo de cada vez.

c) **Mini-batch Gradient Descent:** Calcula o gradiente usando pequenos lotes de exemplos.

2.6 **Árvores de decisão**

Árvores de decisão são modelos preditivos utilizados tanto para tarefas de classificação quanto de regressão. Elas funcionam dividindo repetidamente os dados de acordo com um conjunto de critérios de divisão, criando uma estrutura em forma de árvore onde cada nó interno representa uma condição sobre uma característica dos dados (Breiman et al., 1984; Quinlan, 1986; Ng, 2022; Géron, 2024), pode-se observar na figura 2.7 um exemplo de árvore de decisão.

Figura 2.7 - Exemplo de árvore de decisão



Fonte: Estatsite, 2016

2.6.1 Estrutura de uma Árvore de Decisão

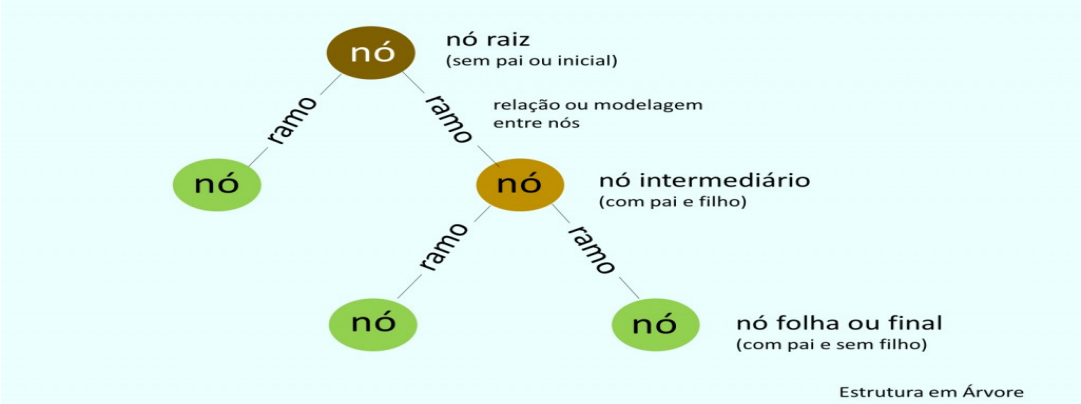
A estrutura de uma árvore de decisão possui diferentes elementos, que são representados na figura 2.8:

a) **Nó Raiz e Nós Intermediários:** Representam uma condição de teste sobre um atributo (por exemplo, se a temperatura > 30°C).

b) **Ramos:** Representam o resultado de um teste e conectam um nó interno ao próximo nó (interno ou folha).

c) **Nós Folha:** Representam uma previsão ou um valor de classe (no caso de classificação) ou um valor contínuo (no caso de regressão)

Figura 2.8 - Estrutura de uma árvore de decisão



Fonte: Colaboraе, 2023

2.6.2 Funcionamento do algoritmo

O algoritmo de uma árvore de decisão acontece através das seguintes etapas (Ng, 2022; Géron, 2024):

a) **Selecionar a melhor Variável:** Escolha a variável que melhor separa os dados em diferentes classes (ou reduz a incerteza no caso de regressão).

b) **Dividir o Conjunto de Dados:** Divida o conjunto de dados em subconjuntos de acordo com a variável selecionada.

c) **Repetir o Processo:** Aplique recursivamente os passos *a* e *b* para cada subconjunto, criando novos nós internos, até que os dados sejam perfeitamente classificados ou um critério de parada seja atendido (como profundidade máxima da árvore ou número mínimo de amostras em um nó).

2.6.3 Cálculo da Impureza de Gini

A impureza de Gini é uma métrica usada para avaliar a qualidade das divisões em uma árvore de decisão e selecionar a melhor variável para divisão. Ela mede a frequência com que um elemento escolhido aleatoriamente seria incorretamente classificado (Géron,2024).

Para um nó *t* com n classes, a impureza de Gini é representada pela equação 2.6.

$$Gini(t) = 1 - \sum_{i=1}^{n} P_i^2 \tag{2.6}$$

onde, P_i é a proporção de exemplos da classe *i* no nó *t*.

Durante o treinamento de uma árvore de decisão, o algoritmo seleciona recursivamente as melhores variáveis para dividir os dados com base na redução da impureza (ou aumento do ganho de informação). O objetivo é criar uma estrutura que minimize a impureza em cada divisão, resultando em nós folhas que são tão puros quanto possível (Géron, 2024).

2.7 Ensembles de árvores de decisão: Random Forest e XGBoost

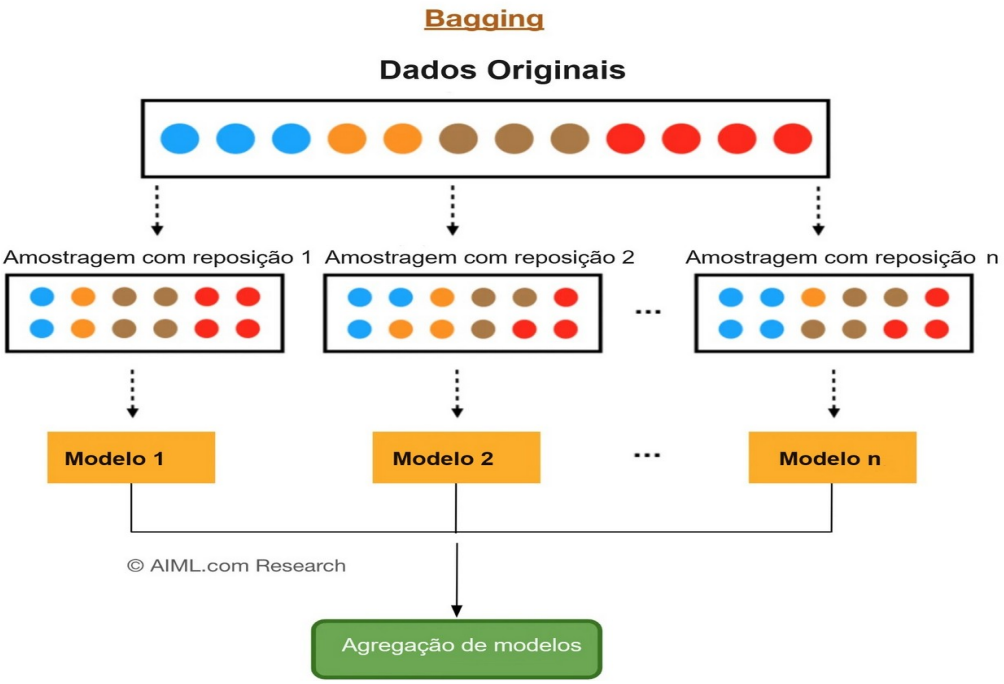
Antes de entrar nos detalhes de algoritmos específicos como *Random Forest* e *XGBoost*, é importante entender os conceitos fundamentais de técnicas de *ensemble*. Essas técnicas combinam múltiplos modelos fracos para criar um modelo forte, melhorando a

precisão e a robustez das previsões, entre elas temos *bagging* e *boosting* (Ng, 2022; Géron, 2024).

2.7.1 *Bagging (*Agregação por amostragem com reposição*)*

Bagging é uma técnica de ensemble que melhora a precisão e a estabilidade dos modelos de *machine learning*. Ele funciona criando e treinando múltiplos modelos em subconjuntos diferentes do conjunto de dados de treinamento original, utilizando amostragem com reposição, e então combina as previsões de todos e através de um sistema de votação é encontrada a predição final (Ng, 2022; Géron, 2024). O processo é representado em um fluxograma na figura 2.9.

Figura 2.9 - Fluxograma de funcionamento do bagging



Fonte: AIML, 2023

O Bagging possui as seguintes vantagens (Ng, 2022; Géron, 2024):

a) **Redução da Variância:** Ao combinar múltiplos modelos, o *bagging* reduz a variância e melhora a generalização.

b) **Robustez:** A técnica é robusta contra *overfitting*, que ocorre quando um modelo se ajusta muito bem aos dados de treinamento, mas não generaliza bem

para novos dados. Em outras palavras, o modelo se torna "muito complexo" e aprende o "ruído" dos dados em vez dos padrões subjacentes.

2.7.2 *Random Forest*

Random Forest é um algoritmo de bagging que constrói uma "floresta" de árvores de decisão e combina suas previsões para melhorar a precisão e a robustez do modelo (Breiman, 2001; Ng, 2022; Géron, 2024), que funciona através de diferentes etapas:

a) **Amostragem com Reposição:** Cria múltiplos subconjuntos do conjunto de dados original.

b) **Treinamento de Árvores:** Treina uma árvore de decisão em cada subconjunto. Cada árvore é construída usando um subconjunto aleatório das características (features).

c) **Votação majoritária:** Para classificação, usa votação majoritária para combinar as previsões das árvores; para regressão, usa a média das previsões.

2.7.3 *Boosting*

Boosting é uma técnica de *ensemble* que constrói modelos sequencialmente, onde cada novo modelo tenta corrigir os erros dos modelos anteriores, como representado no fluxograma da figura 2.10, durante o seu treinamento ocorrem diferentes etapas (Ng, 2022; Géron, 2024):

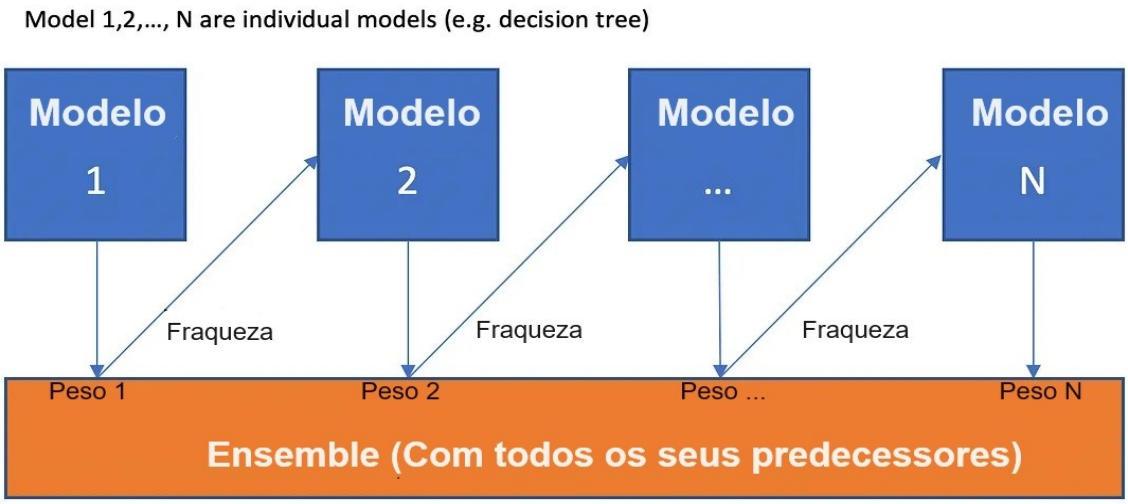
a) **Inicialização:** Começa com um modelo base simples.

b) **Iterações:** Em cada iteração, calcula-se o erro residual do modelo atual e com base nisso são atribuídos pesos às folhas das árvores.

c) **Ajuste do Modelo:** Treina um novo modelo para prever os erros residuais.

d) **Combinação:** Combina o modelo novo com os anteriores para melhorar a precisão.

Figura 2.10 - Fluxograma de funcionamento do boosting



Fonte: Zhang, 2019

2.7.4 *XGBoost* (*Extreme Gradient Boosting*)

XGBoost é uma poderosa implementação do algoritmo de *gradient boosting*, projetada para ser altamente eficiente e eficaz em uma variedade de tarefas de *machine learning*. Ele introduz várias otimizações e características que o tornam um dos métodos mais populares para modelagem preditiva (NG,2022).

2.7.4.1 Regularização

O *XGBoost* utiliza a regularização L2 por padrão para penalizar a complexidade do modelo. Embora a regularização L1 não seja ativada por padrão, ela também pode ser utilizada (Chen & Guestrin, 2016).

a) **Regularização L1 (Lasso):** Adiciona um termo que penaliza a soma dos valores absolutos dos pesos das folhas das árvores e é apresentada pela equação 2.7. Isso incentiva a redução de alguns pesos a zero, potencialmente simplificando a estrutura das árvores (Géron, 2024).

$$L1 = \alpha \sum_{i=1}^{m} |w_J| \tag{2.7}$$

onde, α é o parâmetro de regularização L1 e *wj* é o peso da folha.

b) **Regularização L2 (Ridge):** Adiciona um termo à função de custo que penaliza a soma dos quadrados dos pesos das folhas das árvores e é apresentada

pela equação 2.8. Isso tende a reduzir uniformemente a magnitude dos pesos, sem necessariamente zerá-los(Géron, 2024).

$$L_2 = \lambda \sum_{i=1}^{m} w_J^2 \tag{2.8}$$

onde, λ é o parâmetro de regularização L2 e *wj* é o peso da folha.

2.7.4.2 *Pruning* de Árvores

Pruning (poda) é a técnica de cortar partes da árvore de decisão que têm pouca importância na previsão final. Isso ajuda a evitar o *overfitting* ao remover ramos que adicionam complexidade sem melhorar significativamente o desempenho do modelo. O XGBoost realiza poda de árvores durante o treinamento ao definir um parâmetro ɣ. ɣ especifica o ganho mínimo necessário para fazer uma divisão em um nó da árvore, se a redução no erro ao fazer a divisão for menor que o valor de ɣ, a divisão não é feita. Isso ajuda a manter a árvore mais simples e eficiente (Chen & Guestrin, 2016).

2.7.4.3 Manipulação de Valores Ausentes

Valores ausentes ocorrem quando alguns dados não estão disponíveis. Isso pode acontecer por vários motivos, como falhas na coleta de dados ou problemas na transmissão de dados. O XGBoost é capaz de aprender automaticamente a melhor maneira de tratar os valores ausentes durante o treinamento. Quando um valor ausente é encontrado, o XGBoost decide qual caminho seguir na árvore de decisão com base em qual opção resulta em menor erro para as previsões. Isso torna o algoritmo robusto e capaz de lidar eficientemente com dados incompletos (Chen & Guestrin, 2016).

2.7.4.4 Paralelismo

Paralelismo refere-se à capacidade de executar múltiplas operações simultaneamente, aproveitando os recursos de múltiplos núcleos de CPU para acelerar o processamento. O XGBoost implementa paralelismo ao dividir o conjunto de dados e realizar operações de construção de árvores em paralelo. Isso é especialmente útil para grandes conjuntos de dados, pois reduz significativamente o tempo de treinamento. Ele utiliza técnicas avançadas de

paralelismo para processar diferentes partes da árvore de decisão simultaneamente, aumentando a eficiência do treinamento (Chen & Guestrin, 2016).

2.7.4.5 Aproximação por Histograma

Aproximação por histograma é uma técnica para acelerar a busca pelo melhor ponto de divisão em cada nó da árvore, agrupando os dados em intervalos. Para conjuntos de dados grandes, calcular exatamente o melhor ponto de divisão pode ser computacionalmente caro. O XGBoost usa a aproximação de histograma para acelerar esse processo. Em vez de calcular a divisão exata para cada ponto de dados, ele agrupa os dados em intervalos e calcula a melhor divisão com base nesses grupos. Isso reduz a quantidade de cálculos necessários e acelera o processo de treinamento, mantendo uma boa precisão (Chen & Guestrin, 2016).

2.8 Métricas de Validação para Modelos de Classificação

A avaliação de modelos de classificação é fundamental para determinar a eficácia de um modelo preditivo. Diversas métricas de validação são utilizadas para medir o desempenho desses modelos, cada uma com suas próprias vantagens e limitações.

2.8.1 Matriz de Confusão

A matriz de confusão é uma ferramenta essencial para a avaliação de modelos de classificação, pois fornece uma visão detalhada do desempenho do modelo, contabilizando as previsões corretas e incorretas. Ela é uma tabela que resume as previsões de um modelo em termos de verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos, a tabela 2.1 representa a sua estrutura (Ng, 2022; Géron, 2024).

Tabela 2.1 - Matriz de confusão

	Previsto Positivo	**Previsto Negativo**
Real Positivo	**Verdadeiros Positivos(TP)**	**Falsos Negativos(FN)**
Real Negativo	**Falsos Positivos(FP)**	**Verdadeiros Negativos(TN)**

Fonte: NG,2022

TP (True Positives): Instâncias corretamente previstas como positivas.

TN (True Negatives): Instâncias corretamente previstas como negativas.

FP (False Positives): Instâncias incorretamente previstas como positivas.

FN (False Negatives): Instâncias incorretamente previstas como negativas.

A matriz de confusão é a base para calcular várias métricas de desempenho, permitindo uma compreensão mais profunda dos tipos de erros cometidos pelo modelo (Ng, 2022; Géron, 2024).

2.8.2 Acurácia

A acurácia é a métrica mais básica e amplamente utilizada para avaliar modelos de classificação. Ela é definida como a proporção de previsões corretas em relação ao total de previsões e representa pela equação 2.9 (Ng, 2022; Géron, 2024).

$$Acur\acute{a}cia = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.9)$$

A acurácia é uma medida de simples entendimento, porém possui limitações como:

a) Desequilíbrio de Classes: A acurácia pode ser enganosa em conjuntos de dados desbalanceados. Por exemplo, em um conjunto de dados onde 95% das amostras pertencem a uma classe, um modelo que sempre prevê essa classe terá uma alta acurácia (95%), mas não será útil para identificar a minoria(Ng, 2022; Géron, 2024).

b) Cegueira para Tipos de Erros: A acurácia não diferencia entre os tipos de erros (FP e FN), o que pode ser crítico em certas aplicações, como na detecção de fraudes ou diagnósticos médicos(Ng, 2022; Géron, 2024).

2.8.3 Precisão e *Recall*

Para suprir as limitações da acurácia, métricas como precisão e *recall* são utilizadas para fornecer uma visão mais detalhada do desempenho do modelo.

A precisão mede a proporção de verdadeiros positivos entre todas as instâncias que foram previstas como positivas e é representada pela equação 2.10 (Ng, 2022; Géron, 2024):

$$Pr\,e\,cis\tilde{a}\,o = \frac{TP}{TP + FP} \quad (2.10)$$

Pode-se interpretar da seguinte forma:

a) Alta precisão significa que a maioria das instâncias previstas como positivas são realmente positivas (Ng, 2022).

b) É particularmente importante em situações onde o custo de falsos positivos é alto (Ng, 2022).

O recall (sensibilidade ou revocação) mede a proporção de verdadeiros positivos entre todas as instâncias que são realmente positivas e é representado pela equação 2.11 (NG,2022; GERÓN, 2024):

$$Recall = \frac{TP}{TP + FN} \quad (2.11)$$

Interpretando-se da seguinte forma:

a) Alto *recall* significa que a maioria das instâncias realmente positivas são identificadas pelo modelo (Ng, 2022).

b) É crucial em cenários onde o custo de falsos negativos é alto, como na detecção de doenças (Ng, 2022).

2.8.4 *F1-Score*

O *F1-score* é a média harmônica entre precisão e *recall*, proporcionando um equilíbrio entre essas duas métricas e é representado pela equação 2.12. É particularmente útil quando há um desequilíbrio entre as classes positivas e negativas(Ng, 2022; Géron, 2024).

$$F1 - Score = 2 \times \frac{Precisão \times Recall}{Precisão + Recall} \quad (2.12)$$

Interpretando-se da seguinte forma:

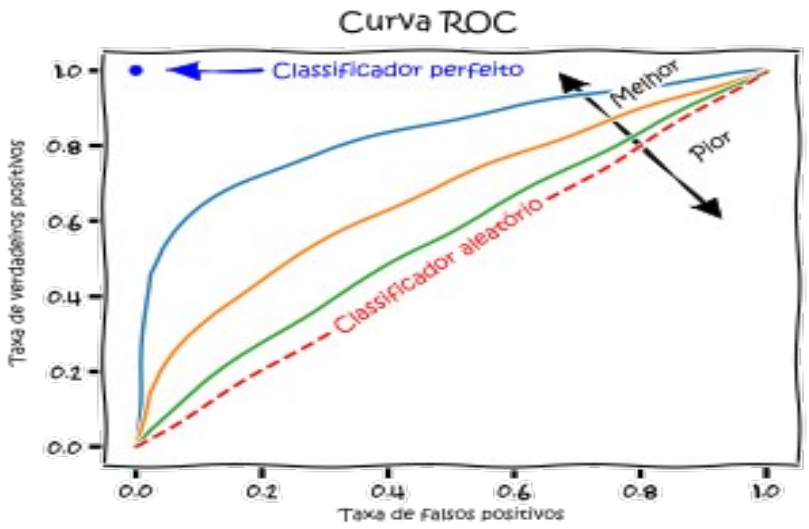
a) *O F1-score* é mais informativo do que a acurácia em conjuntos de dados desbalanceados(NG,2022).

b) É uma métrica robusta quando se deseja um equilíbrio entre precisão e recall(NG,2022).

2.8.5 ROC AUC (*Receiver Operating Characteristic - Area Under the Curve*)

A curva ROC é uma representação gráfica mostrada na figura 2.11 que mostra a performance de um modelo de classificação em diferentes limiares de decisão. A área sob a curva resume essa performance em um único valor entre 0 e 1(Géron, 2024).

Figura 2.11 - Gráfico da curva ROC



Fonte: Wikipedia, 2021

Componentes da Curva ROC:

a) Taxa de Verdadeiros Positivos (TPR ou Recall);
b) Taxa de Falsos Positivos (FPR) representada pela equação 2.13.

$$FPR = \frac{FP}{TP + FN} \quad (2.13)$$

O cálculo da área sob a curva é realizado utilizando a integral representada na equação 2.14.

$$ROC\ AUC = \int_0^1 TPR\, d(FPR) \quad (2.14)$$

O valor ROC AUC pode ser interpretado da seguinte forma:

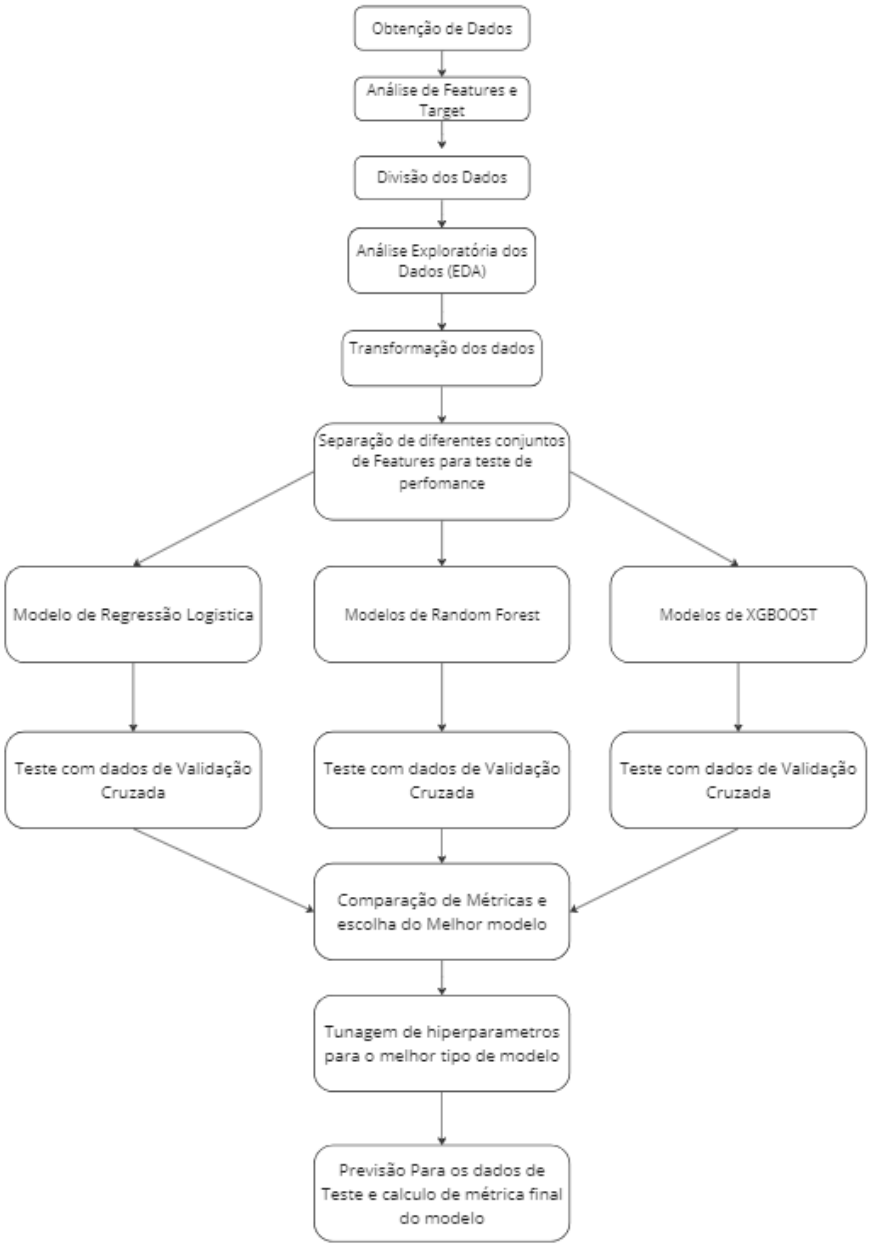
a) Um AUC próximo de 1 indica um modelo excelente, enquanto um AUC próximo de 0.5 indica um modelo que não é melhor do que um classificador aleatório (Géron, 2024).

b) A ROC AUC é útil para comparar modelos e avaliar a capacidade discriminativa de um modelo, especialmente em cenários de desequilíbrio de classes(Géron, 2024).

3 METODOLOGIA

O fluxograma apresentado na figura 3.1 ilustra as etapas metodológicas seguidas no desenvolvimento do modelo preditivo para manutenção preditiva.

Figura 3.1 - Fluxograma da metodologia utilizada no projeto



Fonte: Autor

O processo começa com a obtenção dos dados, seguida pela análise de *features* e *target*, onde as variáveis relevantes são identificadas, é realizada a engenharia de *features* e verificação do balanceamento de classes. Após a divisão dos dados, é realizada uma análise exploratória dos dados (EDA) nos dados de treino para entender suas características. Os dados são então transformados para adequação aos modelos preditivos. Em seguida, diferentes conjuntos de *features* são separados para testar o desempenho dos modelos, que incluem regressão logística como *baseline*, *Random Forest*, e *XGBoost*. Cada modelo é testado usando validação cruzada, e suas métricas de desempenho são comparadas. O melhor modelo é então selecionado e submetido a um ajuste de hiperparâmetros para otimização, antes de ser aplicado aos dados de teste para o cálculo das métricas finais de desempenho.

3.1 Obtenção de Dados

Neste Projeto foi Utilizado o *AI4I 2020 Predictive Maintenance Dataset* Encontrado no *UC Irvine Machine Learning Repository*, O conjunto de dados de manutenção preditiva AI4I 2020 é um conjunto de dados sintético que reflete dados reais de manutenção preditiva encontrados no setor. Na tabela 3.1 encontra-se as informações fornecidas sobre como o *dataset* foi gerado sinteticamente:

Tabela 3.1 - Informações sobre o *dataset*

UID	Identificador exclusivo que varia de 1 a 10.000.
ID do produto	Consiste em uma letra L, M ou H para baixo (50% de todos os produtos), médio (30%) e alto (20%) como variantes de qualidade do produto e um número de série específico da variante.
Qualidade	Indicador de qualidade L(Baixa), M(Média) ou H(Alta).
Temperatura do ar [K]	Gerada usando um processo de passeio aleatório posteriormente normalizado para um desvio padrão de 2 K em torno de 300 K.

Temperatura do processo [K]	Gerada usando um processo de passeio aleatório normalizado para um desvio padrão de 1 K, adicionado à temperatura do ar mais 10 K.
Torque [Nm]	Os valores de torque são normalmente distribuídos em torno de 40 Nm com um Ïf = 10 Nm e nenhum valor negativo.
Velocidade de rotação [rpm]	Calculada a partir de uma potência de 2860 W, sobreposta a um ruído normalmente distribuído.
Desgaste da ferramenta (*Tool wear*) [min]	As variantes de qualidade H/M/L adicionam 5/3/2 minutos de desgaste da ferramenta à ferramenta usada no processo.
Falha	Indica se a máquina falhou nesse ponto de dados específico para qualquer tipo de falha.

Fonte: Autor

A falha da máquina consiste em cinco modos de falha independentes:

a) Falha por desgaste da ferramenta: a ferramenta será substituída ou falhará em um tempo de desgaste da ferramenta selecionado aleatoriamente entre 200 e 240 minutos (120 vezes em nosso conjunto de dados). Nesse momento, a ferramenta é substituída 69 vezes e falha 51 vezes (atribuídas aleatoriamente).

b) Falha na dissipação de calor: a dissipação de calor causa uma falha no processo se a diferença entre a temperatura do ar e a do processo for inferior a 8,6 K e a velocidade de rotação da ferramenta for inferior a 1380 rpm. Esse é o caso de 115 pontos de dados.

c) Falha de potência: o produto do torque e da velocidade de rotação (em rad/s) é igual à potência necessária para o processo. Se essa potência estiver abaixo de 3.500 W ou acima de 9.000 W, o processo falha, o que ocorre 95 vezes em nosso conjunto de dados.

d) Falha por sobretensão: se o produto do desgaste da ferramenta e do torque exceder 11.000 minNm para a variante de produto L (12.000 M, 13.000 H), o processo falha devido à sobretensão. Isso se aplica a 98 pontos de dados.

e) Falhas aleatórias: cada processo tem uma chance de 0,1% de falhar, independentemente dos parâmetros do processo. Esse é o caso de apenas 5 pontos de dados, menos do que se poderia esperar de 10.000 pontos de dados em nosso conjunto de dados.

Se pelo menos um dos modos de falha acima for verdadeiro, o processo falha e o *Target* (falha da máquina) foi definido como 1. Portanto, não é transparente qual dos modos de falha causou a falha do processo, sendo um caso de classificação binaria.

3.2 Análise de *Features* e Target

Ao analisar as colunas do *dataset*, pode-se identificar duas colunas de ID que não são necessárias para o modelo, deixando-nos com seis *features* relevantes: temperatura do ar, temperatura do processo, velocidade de rotação, torque, qualidade e desgaste da ferramenta.

A temperatura do ar e do processo são correlacionadas e a diferença entre as duas trazem mais informação que as mesmas individualmente, através de engenharia de *features* essas duas variáveis foram transformadas em apenas uma equivalente a diferença de temperatura.

Em seguida, é importante avaliar a proporção entre as classes 0 e 1 (casos com e sem falha) para determinar se o *dataset* está balanceado. Um conjunto de dados é considerado balanceado quando as classes estão distribuídas de forma aproximadamente igual, a tabela 3.2 apresenta o número de ocorrência de falhas do *dataset.*

Tabela 3.2 - Número de ocorrência de falhas

Falha	Número de ocorrências
0	9661
1	339

Fonte: Autor

É possível perceber que as classes estão altamente desbalanceadas, com apenas cerca de 3% de ocorrência de falhas. Esse desequilíbrio é problemático, pois dificulta o aprendizado do modelo. Para lidar com essa questão, uma abordagem comum é atribuir um peso à classe minoritária. Isso significa que o modelo dará mais importância aos exemplos da classe com menor ocorrência, ajudando-o a aprender com mais eficácia os padrões associados às falhas.

Para calcular o peso, utiliza-se a fórmula apresentada na equação 3.1.

$$Peso\,de\,classes = \frac{Ocorrências\,de\,classe\,0}{Ocorrências\,de\,classe\,1} \tag{3.1}$$

logo, Peso = 9661 / 339 = 28,5.

Será aplicado esse peso durante o treinamento do modelo para mitigar o desbalanceamento e melhorar sua capacidade de predição da classe minoritária.

3.3 Divisão dos Dados

Para evitar o *Data Leakage*(Vazamento de dados) é de extrema importância que os dados sejam divididos em diferentes conjuntos de treino, validação cruzada e teste antes de realizar qualquer pré-processamento ou análise nos dados. Desta forma, é possível garantir que as métricas de validação generalizem bem e reflitam a performance do modelo em dados nunca vistos.

Como foi visto que as classes são altamente desbalanceadas, é preciso realizar uma divisão de dados estratificada, assim mantendo a proporção de falhas em todos os conjuntos de dados. Desta forma, foi realizada uma divisão estratificada de 70% dos dados para treino, 15% para validação cruzada e 15% para teste, resultando nos conjuntos de dados mostrados na tabela 3.3:

Tabela 3.3 - Divisão dos dados

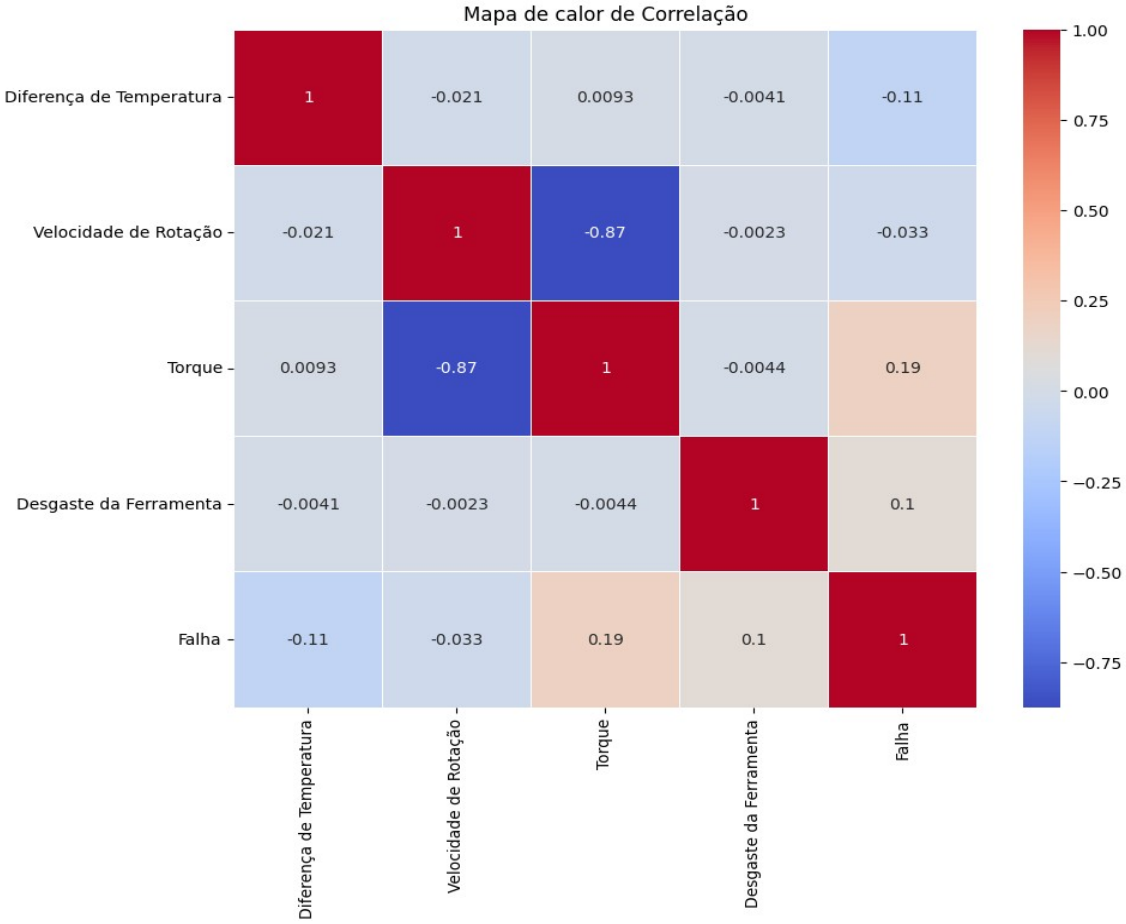
Tamanho do conjunto de treinamento	7000
Tamanho do conjunto de validação cruzada	1500
Tamanho do conjunto de teste	1500

Fonte: Autor

3.4 Análise Exploratória dos Dados (EDA)

Após a divisão dos dados, foi realizada a EDA apenas nos dados de treino. Primeiramente foram calculados os coeficientes de correlação entre as variáveis e gerado um mapa de calor apresentado na figura 3.2.

Figura 3.2 - Mapa de calor dos coeficientes de correlação



Fonte: Autor

Percebe-se que o Torque e a Velocidade de rotação são altamente correlacionados, com um coeficiente -0,87, isso levanta o questionamento se todas as *features* realmente são necessárias ou se é possível remover alguma *feature* do modelo, simplificando o mesmo e mantendo a performance.

Também é importante avaliar a variância das *features*, que são apresentadas na tabela 3.4.

Tabela 3.4 - Variância das features numéricas

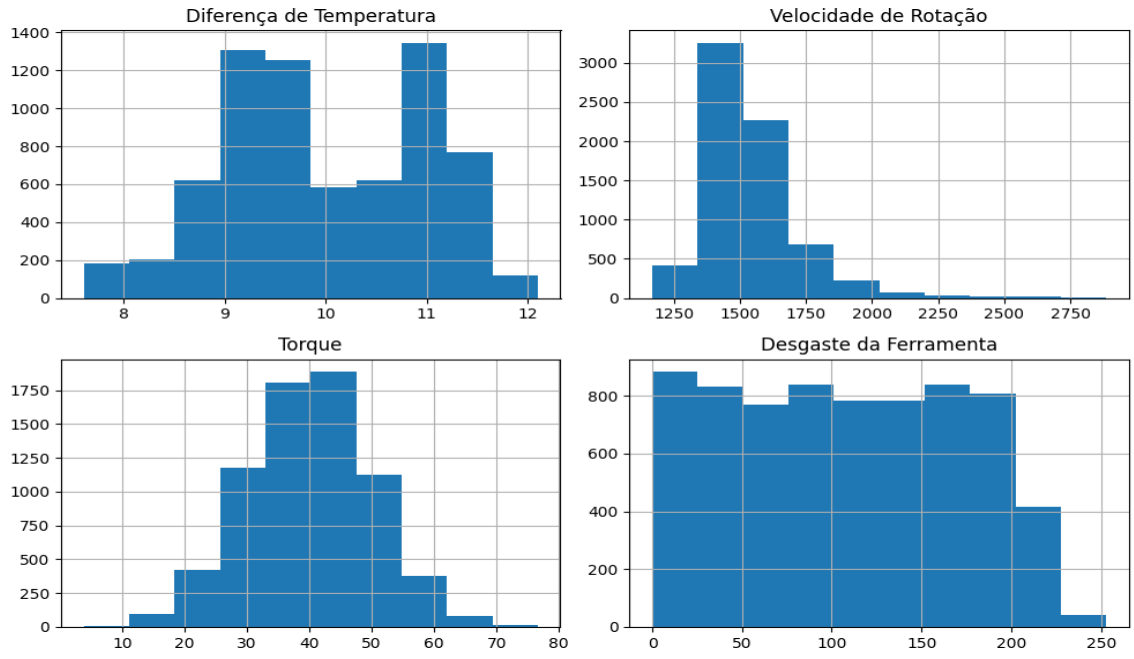
Variável numérica	Variância
Diferença de Temperatura	0,99829
Velocidade de Rotação	32485
Torque	100,12
Desgaste da Ferramenta	4078,6

Fonte: Autor

Realizando uma análise de variância, percebe-se grandes diferenças em suas magnitudes, com a velocidade de rotação possuindo a maior entre elas, logo, será vantajoso realizar uma transformação de escalonamento posteriormente para normalizar os valores dessas variáveis.

Em seguida, foram gerados os gráficos de histograma apresentados na figura 3.3 para se analisar como estão as distribuições dos dados.

Figura 3.3 - Histogramas das variáveis numéricas



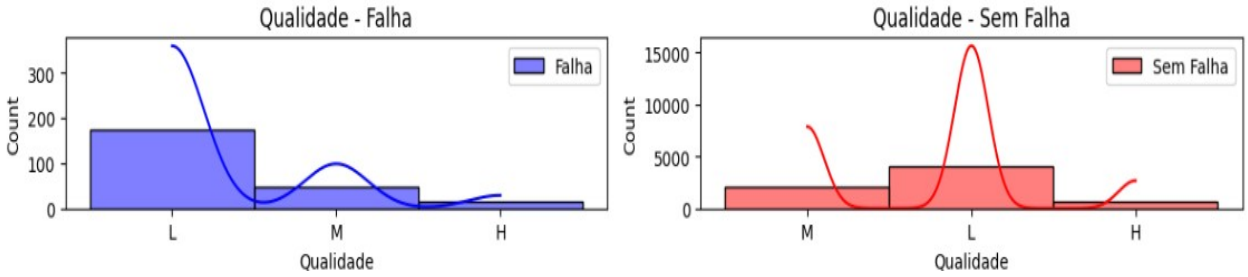
Fonte: Autor

Percebe-se que o Torque e a Velocidade de rotação possuem uma diferença significativa em suas distribuições, levando em conta que são altamente correlacionados, com o torque se aproximando mais do normal e a velocidade de rotação possuindo uma maior variância, como foi visto anteriormente.

O desgaste da ferramenta possui uma distribuição bem uniforme, tendo uma queda apenas em valores mais altos.

A seguir foi realizada uma divisão em 2 *datasets*, um apenas com as falhas e outro apenas com os casos sem falha, para gerar graficamente as distribuições dos dados para cada caso específico, assim tendo um entendimento melhor do que contribui com a falha, os gráficos gerados são apresentados nas figuras 3.4 a 3.7.

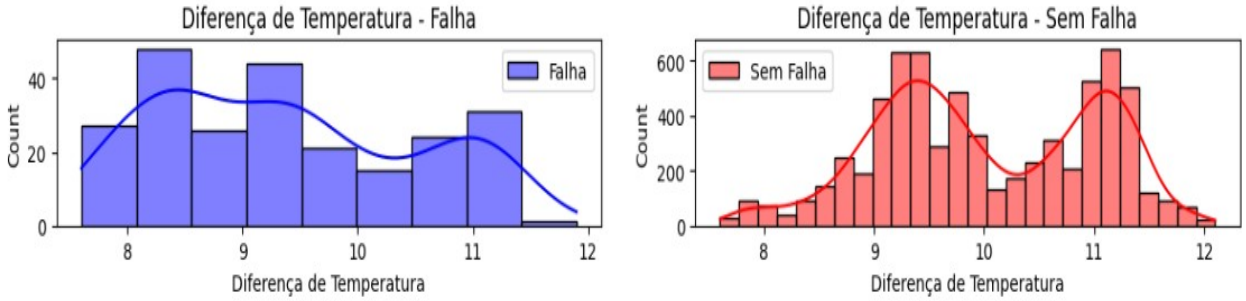
Figura 3.4 - Distribuição da variável qualidade por classes



Fonte: Autor

Pode-se claramente perceber que a maioria das falhas ocorrem na qualidade L, seguido do M e apenas uma minoria no H, porém não possui exatamente uma relação linear.

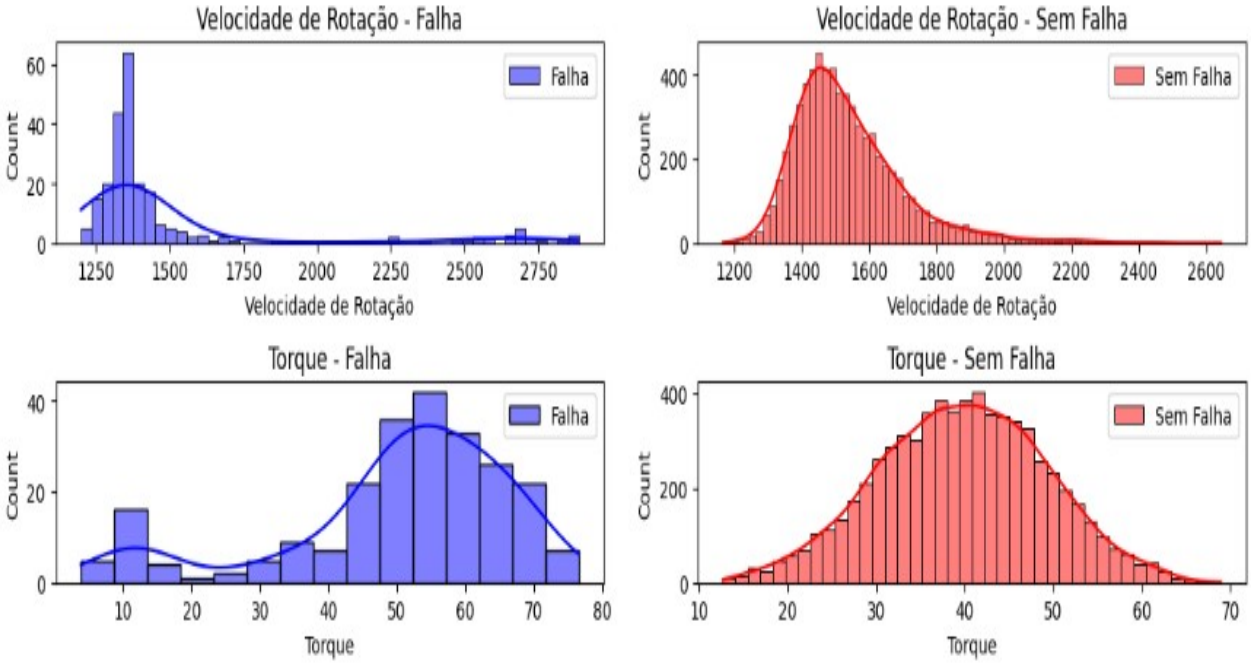
Figura 3.5 - Distribuição da variável diferença de temperatura por classes



Fonte: Autor

Percebe-se que para menores diferenças de temperatura ocorrem mais falhas do que em diferenças elevadas.

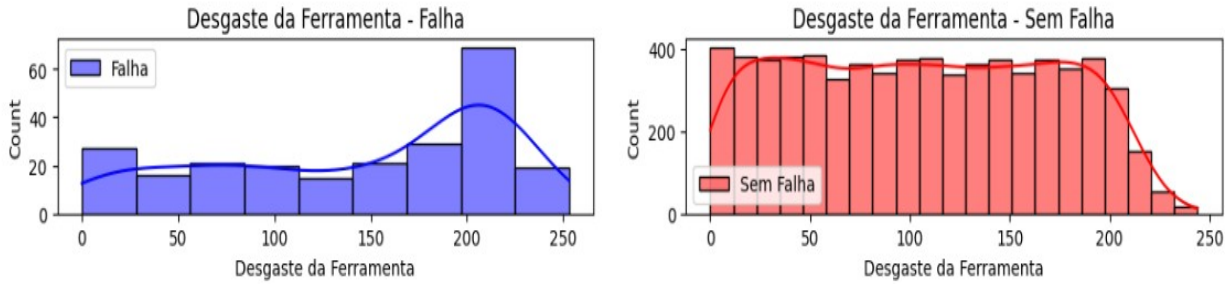
Figura 3.6 - Distribuição das variáveis velocidade de rotação e torque por classes



Fonte: Autor

Pode-se perceber que no geral quanto maior torque e menor rotação mais falhas. Porém, embora em um menor número, também ocorrem falhas para baixos torques e altas rotações. Isso indica que modelos que utilizam equações lineares como regressão logística e afins podem ter dificuldades em prever as falhas e modelos baseados em *ensembles* de árvores de decisões consigam um melhor resultado.

Figura 3.7 - Distribuição da variável desgaste da ferramenta por classes



Fonte: Autor

Percebe-se claramente um maior número de falhas para um maior tempo de desgaste da ferramenta.

3.5 Transformação dos dados

Ao lidar com dados categóricos, como a variável Qualidade com os valores "L" (baixa), "M" (média) e "H" (alta), é preciso representá-los de uma maneira que seja compreensível para modelos de aprendizado de máquina. O *one-hot encoding* é uma técnica amplamente utilizada para essa finalidade.

Nesse processo, são criadas novas colunas para cada categoria distinta presente na variável categórica. No caso de qualidade como existem três categorias, serão três novas colunas: "L", "M" e "H". Em seguida, para cada observação, atribuímos o valor 1 à coluna correspondente à sua categoria e 0 às outras colunas.

Por exemplo, se tivermos um caso com qualidade igual a "L", será atribuído 1 à coluna "L" e 0 às colunas "M" e "H". Se Qualidade for "M", será atribuído 1 à coluna "M" e 0 às outras. Esse processo cria uma representação binária das categorias, onde apenas uma das novas colunas terá o valor 1 para cada observação, enquanto as outras terão o valor 0.

Essa abordagem é vantajosa porque permite que modelos de aprendizado de máquina processem eficientemente informações categóricas, transformando-as em vetores numéricos que podem ser facilmente manipulados em cálculos matemáticos. Isso é essencial para muitos algoritmos de aprendizado de máquina, que geralmente operam melhor com dados numéricos.

Para as outras 5 variáveis serão realizados escalonamentos *StandardScaler representado pela equação 3.1*, já que *features* com diferentes magnitudes podem dificultar o aprendizado do modelo.

$$Xnovo = \frac{Xi - Xmedio}{Desvio\ Padr\tilde{a}o} \tag{3.1}$$

É importante ressaltar que o cálculo da média e desvio padrão devem ser feitos apenas através dos dados de treino e a transformação deve ser apenas replicada com os mesmos valores para os conjuntos de validação e teste. Caso contrário, estaria ocorrendo vazamento de dados ao calcular a média de todas as amostras.

3.6 Separação de diferentes conjuntos de *features* para teste de performance

Como mencionado anteriormente existem 2 *features* altamente correlacionadas, então se torna válido testar diferentes combinações de *features*. Na ciência de dados, quando se possui 2 variáveis altamente correlacionadas geralmente se opta por realizar um teste removendo uma delas, nesse caso foi escolhido o torque. Porém, levando em conta os conhecimentos em Engenharia Mecânica, faz sentido manter todas as variáveis no modelo, já que elas podem impactar em diferentes aspectos mesmo sendo altamente correlacionadas. Logo, serão realizados testes com as 2 possibilidades.

Serão separados 2 conjuntos de dados, sendo eles:

a) Todas as *features*
b) Removendo o torque

3.7 Modelos de Regressão Logística

Será ajustado um modelo de regressão Logística utilizando peso para balanceamento de classes para cada um dos conjuntos de *features* para se utilizar como modelo de *baseline,* ou seja, as métricas de validação desse modelo serão utilizadas como parâmetro para avaliar os outros.

3.8 Modelos de *Random Forest*

Será ajustado um modelo de *Random Forest* utilizando peso para balanceamento de classes, para cada um dos conjuntos de *features*. Para o modelo de *Random Forest* inicialmente serão feitas limitações no hiperparâmetro *max_depth,* de acordo com o número de *features*:

a) *max_depth* = 6
b) *max_depth* = 5

3.9 Modelos de *XGBoost*

Será ajustado um modelo de *XGBoost* utilizando peso para balanceamento de classes, para cada um dos 2 conjuntos de *features*, utilizando a mesma lógica mencionada para os modelos de *Random Forest*:

a) *max_depth* = 6
b) *max_depth* = 5

3.10 Teste com dados de validação cruzada

Após os ajustes dos modelos nos respectivos dados de treino, serão feitas as predições de cada modelo utilizando os dados de validação, para assim compará-los e encontrar o melhor modelo. As métricas utilizadas para comparação serão:

a) Acurácia
b) AUC ROC
c) Recall
d) Precisão
e) F1 Score

3.11 Ajuste de hiperparâmetros para o melhor tipo de modelo

Após escolha do melhor modelo, foram realizados testes com diferentes hiperparâmetros através de uma busca em grade para verificar se existe uma melhor combinação do que a utilizada anteriormente que aumente a performance.

Os hiperparâmetros foram ajustados realizando uma busca entre os seguintes valores:

a) *n_estimators* (Número de árvores no conjunto de boosting): 50, 75, 100, 120, 150

b) *max_depth*(Profundidade máxima de uma árvore): 5, 6, 7

c) *learning_rate* (Taxa de aprendizado. Reduz o peso de cada árvore, controlando a velocidade de aprendizado do modelo): 0,1, 0,01

d) *subsample* (Fração de amostras usadas para treinar cada árvore): 0,8, 0,9, 1

e) *gamma* (Valor mínimo de redução da perda necessária para fazer uma nova divisão (split) em uma folha da árvore): 0, 0,1, 0,2, 0,5

f) *reg_alpha*(Termo de regularização L1): 0, 0,001, 0,01

g) *reg_lambda* (Termo de regularização L2): 0, 0,001, 0,01

h) *min_child_weight* (Peso mínimo de uma folha. Maior valor impede a criação de folhas com amostras muito pequenas, ajudando a evitar *overfitting*): 1, 2, 3

3.12 Previsão Para os dados de Teste e cálculo de métrica final do modelo

Após encontrar o modelo final, foram feitas as predições para os dados de teste e calculadas as métricas definitivas do modelo que representam dados de produção nunca vistos pelo mesmo.

4 RESULTADOS E DISCUSSÃO

Após o treinamento dos modelos foram calculadas as métricas de validação para os conjuntos de treino e validação cruzada.

4.1 Modelos de Regressão Logística

As tabelas 4.1 e 4.2 apresentam as métricas alcançadas para os dados de treino e validação cruzada respectivamente.

Tabela 4.1 - Métricas de treino da regressão logística

Métrica	Combinação *a*	Combinação *b*
Recall	0,823	0,700
Precisão	0,143	0,068
F1 Score	0,243	0,124
ROC AUC	0,825	0,682
Acurácia	0,827	0,666

Fonte: Autor

Tabela 4.2 - Métricas de validação cruzada da regressão logística

Métrica	Combinação *a*	Combinação *b*
Recall	0,726	0,745
Precisão	0,138	0,078
F1 Score	0,231	0,141

ROC AUC	0,783	0,717
Acurácia	0,836	0,691

Fonte: Autor

Percebe-se que ao utilizar todas as *features* obteve-se uma performance superior, com melhor precisão, porém no geral o modelo de regressão logística alcança métricas abaixo do esperado para esse problema, com muitos falsos positivos

4.2 Modelos de *Random Forest*

As tabelas 4.3 e 4.4 apresentam as métricas alcançadas para os dados de treino e validação cruzada respectivamente.

Tabela 4.3 - Métricas de treino da *Random Forest*

Métrica	Combinação *a*	Combinação *b*
Recall	0,958	0,878
Precisão	0,484	0,313
F1 Score	0,643	0,461
ROC AUC	0,961	0,905
Acurácia	0,964	0,931

Fonte: Autor

Tabela 4.4 - Métricas de validação cruzada da *random forest*

Métrica	Combinação *a*	Combinação *b*
Recall	0,804	0,863
Precisão	0,436	0,296
F1 Score	0,566	0,440
ROC AUC	0,884	0,895
Acurácia	0,958	0,925

Fonte: Autor

Percebe-se que a primeira combinação continuou sendo a melhor devido a uma maior precisão e consequentemente menos falsos positivos, o que é importante pois não é viável ter que parar a produção a todo tempo sem necessidade. Além disso, houve uma grande melhoria de todas as métricas ao utilizar um modelo de *random forest* em comparação com regressão logística, porém embora o *recall* tenha atingido um bom valor, a precisão ainda ficou abaixo do desejado.

4.3 Modelos de XGBoost

As tabelas 4.5 e 4.6 apresentam as métricas alcançadas para os dados de treino e validação cruzada respectivamente.

Tabela 4.5 - Métricas de treino do XGBoost

Métrica	Combinação *a*	Combinação *b*
Recall	1	1
Precisão	0,996	0,801
F1 Score	0,998	0,889
ROC AUC	0,999	0,996
Acurácia	0,999	0,992

Fonte: Autor

Tabela 4.6 - Métricas de validação cruzada do XGBoost

Métrica	Combinação *a*	Combinação *b*
Recall	0,804	0,667
Precisão	0,804	0,558
F1 Score	0,804	0,607
ROC AUC	0,899	0,824

Acurácia	0,987	0,971

Fonte: Autor

Percebe-se uma grande melhoria com os modelos *XGBoost* , alcançando um *recall* e precisão balanceados com um alto valor de 80% de previsão de falhas nos dados de validação, sendo este o melhor modelo encontrado.

4.4 Ajuste de Hiperparâmetros

Após realizar a busca em grade foram encontrados os seguintes hiperparâmetros:

a) ' *gamma*': 0,1;
b) ' *learning_rate*': 0,1;
c) '*max_depth*': 7;
d) '*min_child_weight*': 2;
e) '*n_estimators'*: 120;
f) '*reg_alpha*': 0,001;
g) '*reg_lambda*': 0,01;
h) '*subsample*': 0,8;

A tabela 4.7 apresenta as métricas de validação cruzada encontradas no melhor modelo após o ajuste de parâmetros.

Tabela 4.7 - Métricas de validação cruzada do melhor modelo

Métrica	Melhor modelo
Recall	0,824
Precisão	0,807
F1 Score	0,816
ROC AUC	0,908

Acurácia	0,987

Fonte: Autor

Foi possível adquirir um aumento considerável no *recall*, assim como uma pequena melhoria na precisão e em todas as outras métricas, tornando um sucesso o ajuste de hiperparâmetros.

4.5 Métricas finais nos dados de teste

Para a conclusão do Projeto foi realizado o teste final com o conjunto de teste, a tabela 4.8 apresenta a performance final nos dados de teste.

Métrica	Melhor modelo
Recall	0,843
Precisão	0,796
F1 Score	0,819
ROC AUC	0,918
Acurácia	0,987

Fonte: Autor

Percebe-se que o modelo manteve a sua boa performance e até teve uma pequena melhoria nos diferentes dados, alcançando um *recall* de 0,843 e mantendo uma precisão similar com um *F1 Score* maior. Além disso, alcançou um ROC AUC de 0,918 o que é bastante satisfatório.

5 CONCLUSÃO

Este trabalho desenvolveu um modelo preditivo para prever a ocorrência de falhas em equipamentos, utilizando o *AI4I 2020 Predictive Maintenance Dataset*. Foram aplicados e comparados diferentes algoritmos de aprendizado de máquina, incluindo Regressão Logística, *Random Forests e XGBoost*. Após uma análise comparativa detalhada, o modelo *XGBoost* demonstrou ser o mais eficaz, alcançando métricas de desempenho superiores.

Os objetivos propostos foram atingidos com sucesso. A regressão logística, utilizada como modelo *baseline*, proporcionou uma referência inicial, enquanto os modelos mais avançados, como *Random Forests e XGBoost*, apresentaram melhorias significativas em todas as métricas. O ajuste de hiperparâmetros no modelo *XGBoost* resultou em um aumento considerável no recall mantendo uma boa precisão.

A importância deste estudo para a indústria é significativa, pois a previsão precisa de falhas permite não apenas reduzir custos operacionais, mas também evitar interrupções inesperadas e prolongar a vida útil dos equipamentos. A aplicação prática dos modelos desenvolvidos pode resultar em uma gestão mais eficiente dos recursos e em uma manutenção mais proativa, contribuindo para uma operação industrial mais estável e econômica.

Para trabalhos futuros, sugere-se a implementação de sistemas de monitoramento em tempo real integrados ao modelo para a manutenção preditiva, permitindo resposta rápida às condições variáveis dos equipamentos. Outra sugestão é a análise de impacto econômico detalhada para quantificar os benefícios financeiros da implementação de tais modelos em ambientes industriais.

REFERÊNCIAS

AIML. What is Bagging? Figura. 2023. Disponível em: https://aiml.com/what-is-bagging/. Acesso em: 15 ago. 2024.

Alpaydin, E. (2020). Introduction to Machine Learning (4th ed.). MIT Press.

Análise Macro. Regressão linear: teoria e prática. Imagem. 2023. Disponível em: https://analisemacro.com.br/econometria-e-machine-learning/regressao-linear-teoria-e-pratica/#:~:text=Uma%20regress%C3%A3o%20linear%20%C3%A9%20uma,linear%20com%20outra%20vari%C3%A1vel%20x. Acesso em: 15 ago. 2024.

Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5-32.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). Classification and Regression Trees. Wadsworth International Group.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785-794.

Colaborae. Árvore de Decisão. Figura. 2023. Disponível em: https://colaborae.com.br/blog/2023/07/19/arvore-de-decisao/. Acesso em: 15 ago. 2024.

Datacamp. Understanding Logistic Regression in Python. Figura. 2024. Disponível em: https://www.datacamp.com/pt/tutorial/understanding-logistic-regression-python. Acesso em: 15 ago. 2024.

Estatsite. Árvore de decisão. Figura. 2016. Disponível em: https://estatsite.com.br/2016/06/11/1970/. Acesso em: 15 ago. 2024.

Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. Annals of Statistics, 29(5), 1189-1232.

Géron, A. (2024). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (3rd ed.). O'Reilly Media.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.). Springer.

Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). Applied Logistic Regression (3rd ed.). Wiley.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning: With Applications in R. Springer.

Jardine, A.K.S., Lin, D., & Banjevic, D. (2006). A Review on Machinery Diagnostics and Prognostics Implementing Condition-Based Maintenance. Mechanical Systems and Signal Processing, 20(7), 1483-1510.

Murphy, K. P. (2012). Machine Learning: A Probabilistic Perspective. MIT Press.

Ng, Andrew. Machine learning specialization. Coursera, 2022. Disponível em: https://www.coursera.org/specializations/machine-learning. Acesso em: 11 mar. 2024.

Quinlan, J. R. (1986). Induction of Decision Trees. Machine Learning, 1(1), 81-106.

Ruder, S. (2016). An Overview of Gradient Descent Optimization Algorithms. Retrieved from arXiv.

Seber, G. A. F., & Lee, A. J. (2012). Linear Regression Analysis (2nd ed.). Wiley.

Tableless. Regressão linear simples. Figura. 2019. Disponível em: https://tableless.com.br/regressao-linear-simples/. Acesso em: 15 ago. 2024.

Wikipedia. Característica de Operação do Receptor. Figura. 2021. Disponível em: https://pt.wikipedia.org/wiki/Caracter%C3%ADstica_de_Opera
%C3%A7%C3%A3o_do_Receptor. Acesso em: 15 ago. 2024.

Zhang, Zixuan. Boosting Algorithms Explained. Figura. 2019. Disponível em: https://towardsdatascience.com/boosting-algorithms-explained-d38f56ef3f30. Acesso em: 15 ago. 2024.

Zhao, Y., & Zuo, M.J. (2012). Maintenance Optimization Using Machine Learning Techniques: A Review. Expert Systems with Applications, 39(1), 1047-1055.

ANEXO A - Código do Modelo Preditivo de falhas

O código a seguir foi utilizado para realizar todo o processamento, análise, modelagem e validação :

Importação de bibliotecas:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, roc_auc_score, make_scorer, log_loss
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
```

Leitura dos dados e renomeação de colunas:

```
df = pd.read_csv("C:\\Users\\gabri\\OneDrive\\Documentos\\predictive_maintenance.csv")
df = df.rename(columns=lambda x: x.split(' [')[0])
```

```
df['Diferença de Temperatura'] = df['Process temperature'] - df['Air temperature']
df.rename(columns={
    'Type': 'Qualidade',
    'Rotational speed': 'Velocidade de Rotação',
    'Torque': 'Torque',
    'Tool wear': 'Desgaste da Ferramenta',
    'Target': 'Falha'
}, inplace=True)
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   UDI                     10000 non-null  int64
 1   Product ID              10000 non-null  object
 2   Qualidade               10000 non-null  object
 3   Air temperature         10000 non-null  float64
 4   Process temperature     10000 non-null  float64
 5   Velocidade de Rotação   10000 non-null  int64
 6   Torque                  10000 non-null  float64
 7   Desgaste da Ferramenta  10000 non-null  int64
 8   Falha                   10000 non-null  int64
 9   Failure Type            10000 non-null  object
 10  Diferença de Temperatura  10000 non-null  float64
dtypes: float64(4), int64(4), object(3)
```

```
memory usage: 859.5+ KB

Análise da proporção entre as classes e cálculo de pesos:

df['Falha'].value_counts()

Falha

0    9661

1     339

Name: count, dtype: int64

weight = 9661 / 339

class_weight = {0: 1, 1: weight}

class_weight

# Divisão dos dados em Features e Target

x = df[['Qualidade', 'Diferença de Temperatura', 'Velocidade de Rotação', 'Torque', 'Desgaste da Ferramenta']]

y = df[['Falha']]

# Dividindo os dados em conjunto de treinamento (70%) e conjunto de teste (30%) de forma estratificada

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42, stratify=y)


# Dividindo o conjunto de teste em conjunto de validação cruzada (50%) e conjunto de teste (50%) de forma estratificada

x_cv, x_test, y_cv, y_test = train_test_split(x_test, y_test, test_size=0.5, random_state=42, stratify=y_test)


# Verificando os tamanhos dos conjuntos de dados

print(f"Tamanho do conjunto de treinamento: {x_train.shape[0]}, {y_train.shape[0]}")

print(f"Tamanho do conjunto de validação cruzada: {x_cv.shape[0]}, {y_cv.shape[0]}")
```

```
print(f"Tamanho do conjunto de teste: {x_test.shape[0]}, {y_test.shape[0]}")
```

Tamanho do conjunto de treinamento: 7000, 7000

Tamanho do conjunto de validação cruzada: 1500, 1500

Tamanho do conjunto de teste: 1500, 1500

Junção do x e y de treino para estudo de correlações e Análise exploratória dos dados(EDA):

```
train_data = x_train.copy()
train_data['Falha'] = y_train
corr = train_data[['Diferença de Temperatura', 'Velocidade de Rotação', 'Torque', 'Desgaste da Ferramenta', 'Falha']].corr()


# Configurar o tamanho do gráfico
plt.figure(figsize=(10, 8))


# Gerar o heatmap
sns.heatmap(corr, annot=True, cmap='coolwarm', linewidths=0.5)


# Título do heatmap
plt.title('Mapa de calor de Correlação')


# Mostrar o gráfico
plt.show()
# Calculo de variâncias
train_data[['Diferença de Temperatura', 'Velocidade de Rotação', 'Torque', 'Desgaste da Ferramenta']].var()
train_data.hist(figsize=(10, 10))  # geração de histogramas, figsize define o tamanho da figura
plt.tight_layout()  # Para evitar que os histogramas se sobreponham
```

```
plt.show()
```

visualização da distribuição dos dados por classes:

```
# Filtrando o DataFrame para os valores de target 1 e 0
train_data_1 = train_data[train_data['Falha'] == 1]
train_data_0 = train_data[train_data['Falha'] == 0]
train_data_1 = train_data_1.drop('Falha', axis = 1)
train_data_0 = train_data_0.drop('Falha', axis = 1)
# Definindo o número de colunas para plotagem
num_cols = len(train_data_1.columns)

# Criando subplots para cada coluna
fig, axs = plt.subplots(nrows=num_cols, ncols=2, figsize=(12, 2*num_cols))

# Plotando os histogramas para cada coluna
for i, col in enumerate(train_data_1.columns):
    sns.histplot(train_data_1[col], ax=axs[i, 0], kde=True, color='blue', label='Falha')
    sns.histplot(train_data_0[col], ax=axs[i, 1], kde=True, color='red', label='Sem Falha')
    axs[i, 0].legend()
    axs[i, 1].legend()
    axs[i, 0].set_title(f'{col} - Falha')
    axs[i, 1].set_title(f'{col} - Sem Falha')

plt.tight_layout()
plt.show()
```

Transformação dos dados:

• Transformar a variavel categorica em valores 0 e 1 através do One Hot Enconding

• Escalar as variaveis numericas através do StandardScaler, para que todas se encontrem na mesma faixa de valores, facilitando o aprendizado dos algoritmos

```
columns_to_encode = ['Qualidade']

columns_to_scale = ['Diferença de Temperatura', 'Velocidade de Rotação', 'Torque', 'Desgaste da Ferramenta']

ct = ColumnTransformer(

   [('one_hot_encoder', OneHotEncoder(), columns_to_encode),

    ('scaler', StandardScaler(),columns_to_scale)],

   remainder='passthrough'  # Leave other columns unchanged

)

df_encoded = ct.fit_transform(x_train)

feature_names = ct.get_feature_names_out()

x_traint= pd.DataFrame(df_encoded, columns = feature_names)

df_encoded = ct.transform(x_cv)

feature_names = ct.get_feature_names_out()

x_cvt= pd.DataFrame(df_encoded, columns = feature_names)

df_encoded = ct.transform(x_test)

feature_names = ct.get_feature_names_out()

x_testt= pd.DataFrame(df_encoded, columns = feature_names)

#Baseado na EDA, será gerado mais 1 datasets para testar qual combinação de features trará um melhor resultado

x_traint2 = x_traint.drop('scaler__Torque', axis=1)

x_cvt2 = x_cvt.drop('scaler__Torque', axis=1)

random_state=42
```

modelos de regressão logistica:

```
logistic = LogisticRegression(class_weight='balanced')

logistic2 = LogisticRegression(class_weight='balanced')

logistic.fit(x_traint, y_train)

logistic2.fit(x_traint2, y_train)

yp_log = logistic.predict(x_traint)

yp_log2 = logistic2.predict(x_traint2)


ypcv_log = logistic.predict(x_cvt)

ypcv_log2 = logistic2.predict(x_cvt2)
```

Função de validação:

```
def validação(y, ypred):
    # Calcular precisão
    precision = precision_score(y, ypred)
    # Calcular recall
    recall = recall_score(y, ypred)
    # Calcular F1 score
    f1 = f1_score(y, ypred)
    accuracy = accuracy_score(y,ypred)
    roc = roc_auc_score(y,ypred)
    logloss = log_loss(y,ypred)
    print("Recall:", recall)
    print("Precision:", precision)
    print("F1 Score:", f1)
    print("ROC AUC:", roc)
    print("Acurácia:", accuracy)
    print("Log Loss:", logloss)
```

```
validação(y_train, yp_log)

validação(y_cv, ypcv_log)

validação(y_train, yp_log2)

validação(y_cv, ypcv_log2)
```

Modelos random forest:

```
rf = RandomForestClassifier(class_weight= 'balanced', max_depth = 6)

rf2 = RandomForestClassifier(class_weight= 'balanced', max_depth = 5)


rf.fit(x_traint,y_train)

rf2.fit(x_traint2,y_train)

yp_rf = rf.predict(x_traint)

ypcv_rf = rf.predict(x_cvt)

yp_rf2 = rf2.predict(x_traint2)

ypcv_rf2 = rf2.predict(x_cvt2)

validação(y_train, yp_rf)

validação(y_cv, ypcv_rf)

validação(y_train, yp_rf2)

validação(y_cv, ypcv_rf2)
```

Modelos XGBoost:

```
xgb = XGBClassifier(scale_pos_weight=weight, objective='binary:logistic',
max_depth=6 ,n_estimators= 100)
```

```
xgb2 = XGBClassifier(scale_pos_weight=weight, objective='binary:logistic',
max_depth=5 ,n_estimators= 100)


xgb.fit(x_traint, y_train)

xgb2.fit(x_traint2, y_train)

yp_xgb = xgb.predict(x_traint)

ypcv_xgb = xgb.predict(x_cvt)

yp_xgb2 = xgb2.predict(x_traint2)

ypcv_xgb2 = xgb2.predict(x_cvt2)

validação(y_train, yp_xgb)

validação(y_cv, ypcv_xgb)

validação(y_train, yp_xgb2)

validação(y_cv, ypcv_xgb2)
```

Ajuste de hiperparâmetros:

```
param_grid = {

'n_estimators': [50,75,100,120,150],

'max_depth': [5,6,7],

'learning_rate': [0.1, 0.01],

'subsample': [0.8, 0.9, 1],

'gamma': [0, 0.1, 0.2, 0.5],

'reg_alpha': [0, 0.001, 0.01],

'reg_lambda': [0, 0.001, 0.01],

'min_child_weight': [1, 2, 3],

'scale_pos_weight': [(9661 / 339)]
```

```
}
scoring = {
   'recall_score': make_scorer(recall_score),
   'f1_score': make_scorer(f1_score),
}
xgb_model = XGBClassifier()
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=3, scoring=scoring,refit='f1_score')
grid_search.fit(x_traint, y_train)
print("Melhores hiperparâmetros encontrados:")
print(grid_search.best_params_)
```

Melhores hiperparâmetros encontrados:

{'gamma': 0.1, 'learning_rate': 0.1, 'max_depth': 7, 'min_child_weight': 2, 'n_estimators': 120, 'reg_alpha': 0.001, 'reg_lambda': 0.01, 'scale_pos_weight': 28.49852507374631, 'subsample': 0.8}

```
best_model = grid_search.best_estimator_
y_pred = best_model.predict(x_cvt)
validação(y_cv,y_pred)
y_predtest = best_model.predict(x_testt)
validação(y_test,y_predtest)
```

Recall: 0.8431372549019608

Precision: 0.7962962962962963

F1 Score: 0.819047619047619

ROC AUC: 0.9177729062639549

Acurácia: 0.9873333333333333

Log Loss: 0.45655294292881743