PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Ricardo Jonas Faria

Análise de Dados de Sensores para Classificação de Falhas em Motores Elétricos

Belo Horizonte
2024

Ricardo Jonas Faria

Análise de Dados de Sensores para Classificação de Falhas em Motores Elétricos

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Belo Horizonte
2024

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1. Introdução

A Indústria 4.0, também chamada de Quarta Revolução Industrial, apresenta um conceito que une automação e tecnologia da informação. Este conceito possibilita a criação de redes inteligentes na cadeia de produção, permitindo a previsão e prevenção de falhas, adaptação mais rápida a mudanças imprevistas, e uma maior autonomia no controle de diversas operações, tendo sempre o foco direcionado à execução de processos mais rápidos, eficientes e seguros (SACOMANO, José Benedito et al., 2018).

Posto isso, a implantação da Indústria 4.0 nas empresas não se trata apenas de uma questão de modernização, mas sim de uma necessidade para que possam se manter competitivas e inovadoras ao usufruir de técnicas de monitoramento em tempo real de processos físicos que facilitam a tomada de decisões descentralizadas.

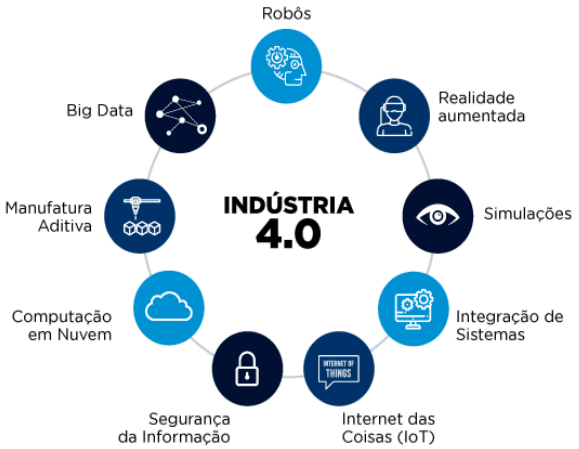


Figura 1: Pilares da Indústria 4.0.

Fonte: Tecnicon, 2022.

Ao integrar diversas tecnologias, a Indústria 4.0 produz uma enorme quantidade de dados que, quando usados da forma correta, permitem aplicar uma série de melhorias nos processos produtivos e na eficiência da produção. Uma técnica que utiliza intensivamente estes dados é a manutenção preditiva. Ela busca analisar dados como vibração, temperatura e ruído, usados para prever quando e que tipo de falha pode ocorrer, evitando paradas de linhas de produção, reduzindo custos e aumentando a vida útil do equipamento. A realização da manutenção preditiva pode reduzir o tempo de inatividade da máquina em 30% a 50% e

aumentar a vida útil da máquina em 20% a 40% (Mckinsey, 2017).

1.1. Contextualização

O uso de motores em diferentes tipos de ambientes produtivos, atuando em condições adversas, pode gerar algum tipo de falha ou desgaste do mesmo, o que faz com que uma demanda considerável seja direcionada para as áreas de manutenção das empresas. Falhas podem acarretar em paradas de linhas de produção, desde as de pequena escala até as de grande porte, trazendo possíveis atrasos em entregas, e custos adicionais ou perdas no faturamento do processo. Para mitigar esse tipo de situação, é recomendado o uso de manutenção preditiva, uma técnica que utiliza ferramentas e procedimentos de análise de dados para detectar anomalias no funcionamento e possíveis defeitos nos equipamentos e processos, de tal modo que possam ser resolvidos antes que a falha aconteça.

Por se tratar de algo relevante, este trabalho tem por objetivo explorar a identificação e classificação de falhas que venham a viabilizar a implantação de manutenção preditiva no acompanhamento de motores em operação.

1.2. O problema proposto

Considerando um motor em funcionamento, propõe-se avaliar o seu comportamento, por meio de tacômetro, acelerômetros e microfone, nas condições de funcionamento classificadas como normal, desbalanceado e com falha no rolamento inferior.

Para realizar estas classificações foram utilizados dados coletados do equipamento de simulação de falha em máquinas da SpectraQuest®, de agora em diante chamado de MFS (Machinery Fault Simulator).

O uso de um conjunto de dados obtido a partir deste simulador, se justifica por oferecer controle preciso sobre os tipos e intensidades de defeitos, permitindo a criação de um conjunto de dados abrangente e com maior variedade de condições de falha. Isso é de grande relevância para treinar um modelo robusto e generalizável. Além disso, o simulador

também possibilita a simulação de falhas raras ou complexas que seriam difíceis de reproduzir em um ambiente real. O resultado esperado é um modelo pré treinado de aprendizado de máquina (machine learning) de classificação de alta performance, que poderá ser usado como base para treinamento e classificação de dados de motores reais.

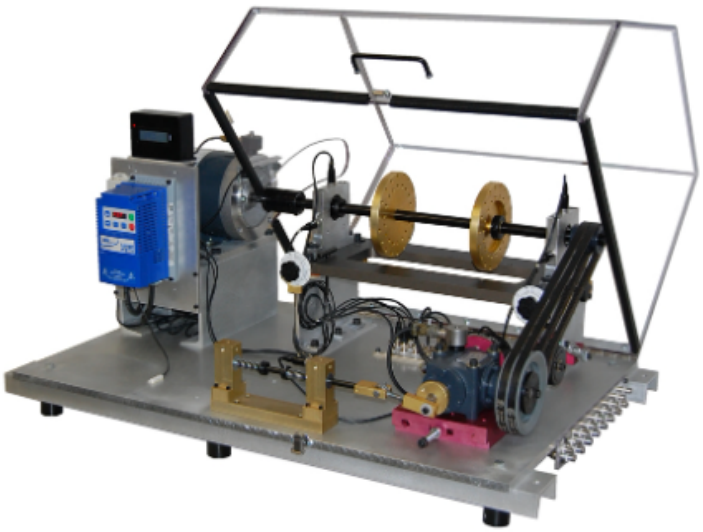


Figura 2: Simulador de falha de máquinas da SpectraQuest®.

Fonte: Spectraquest.

A tabela 1 apresenta detalhes técnicos sobre o motor do simulador.

Especificação	Valor	Unidade
Motor	1/4	CV
Rotação	700 a 3600	rpm
Peso	22	kg
Diâmetro do eixo	16	mm
Comprimento do eixo	520	mm
Rotor	15,24	cm
Distância dos rolamentos	390	mm

Tabela 1: Informações do motor do simulador.
Fonte: Elaboração própria.

Para a geração de dados, foram usados sensores com as características listadas

abaixo.

- 3 acelerômetros IMI Sensors, modelo 601A01, nas direções radial, axial e tangencial
    - sensibilidade: 10.2 mV por m/s²
    - intervalo de frequência: 0,27 a 10000 Hz
    - Intervalo de medida: ±490 m/s²
- 1 acelerômetro triaxial IMI Sensors, modelo 604B31, retornando dados das direções radial, axial e tangencial
    - sensibilidade: 10.2 mV por m/s²
    - intervalo de frequência: 0,5 a 5000 Hz
    - Intervalo de medida: ±490 m/s²
- Tacômetro Monarch Instrument, modelo MT-190
- Microfone Shure, modelo SM81
    - intervalo de frequência: 20 a 20000 Hz
- 2 módulos de aquisição de dados National Instruments, modelo NI 9234
    - taxa de amostragem: 51200 Hz

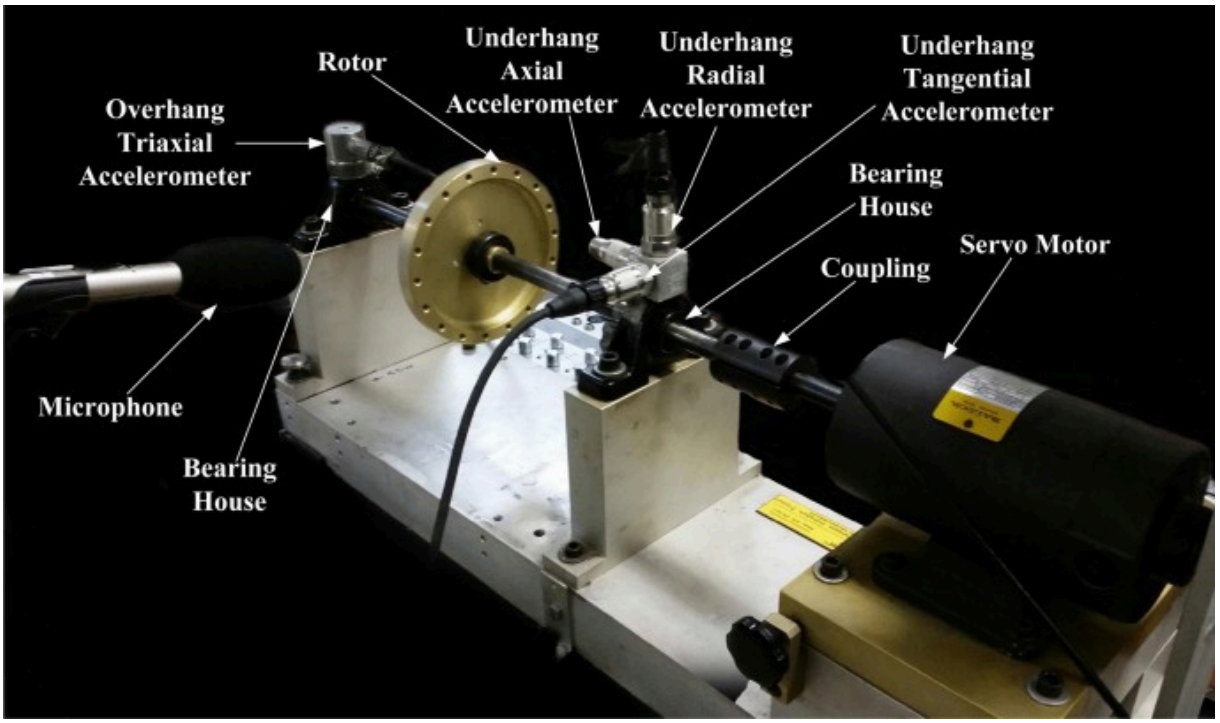


Figura 3: Componentes da estrutura usada para geração de dados - motor + sensores.
Fonte: Das, O., Bagci Das, D., 2023

1.3. Objetivo

O objetivo deste trabalho é treinar um modelo de machine learning capaz de identificar qual o estado de operação de um motor a partir de dados coletados de sensores.

1.4. Ferramentas utilizadas

No desenvolvimento deste projeto foi utilizado o Google Colab, uma plataforma online para execução de códigos Python utilizando Jupyter Notebooks. O ambiente utilizado conta com uma Unidade de Processamento Gráfico (GPU) para acelerar o processamento.

```
import sys
print(sys.version)
```

```
3.10.12 (main, Sep 11 2024, 15:47:36) [GCC 11.4.0]
```

Figura 4: Versão do python no ambiente Colab.
Fonte: Elaboração própria

```
Thu Oct  3 00:51:42 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA L4                      Off | 00000000:00:03.0 Off |                    0 |
| N/A   46C    P8              12W /  72W |      1MiB / 23034MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

Figura 5: Informações sobre GPU no ambiente Colab.
Fonte: Elaboração própria

2. Coleta de Dados

Os dados utilizados neste projeto foram extraídos da base dados Machinery Fault Database, a partir deste momento denominada como MaFaulDa, disponibilizada pela Universidade Federal do Rio de Janeiro (COPPE UFRJ, 2021). Vale ressaltar algumas de suas características:

- Esta base de dados é composta na íntegra por 1951 séries temporais multivariadas.
- Apresenta dados de 6 estados diferentes de operação, das quais serão usados apenas 3 para simplificação e redução do tempo de treinamento:
    - Normal: motor operando normalmente sem nenhum tipo de falha
    - Desbalanceado (Imbalance): Ocorre quando o peso não está distribuído de maneira uniforme e simétrica
    - Falha no rolamento inferior (Underhang bearing faults): Máquina com rolamento defeituoso. O rolamento inferior é o localizado entre o motor e o rotor. Estas falhas podem ser subclassificadas em: defeito nas esferas, defeito na gaiola e defeito na pista. Entretanto, num contexto de manutenção, é mais viável a troca do rolamento com um todo, sendo assim, serão todas classificadas de maneira geral, como defeito no rolamento inferior.
- Cada série foi gerada numa taxa de amostragem de 50 kHz durante 5 segundos, totalizando 250.000 amostras por série.
- Cada série está em um arquivo CSV que contém 8 colunas:
    - Coluna 1: Dados do tacômetro, permite estimar a rotação do motor;
    - Colunas 2, 3 e 4: Dados de acelerômetro do rolamento inferior (underhang bearing). Contém dados de vibração nas direções axial, radial e tangencial;
    - Colunas 5, 6 e 7: Dados de acelerômetro do rolamento superior (overhang bearing). Contém dados de vibração nas direções axial, radial e tangencial;
    - Coluna 8: Dados de microfone;

Das 1951 séries disponibilizadas na base principal, foram utilizadas 940 séries, de acordo com as 3 classificações de uso definidas para este trabalho, totalizando 235.000.000 registros.

Classificação		Quantidade de séries
Normal		49
Desbalanceamento		333
Rolamento inferior	Defeito na gaiola	188
	Defeito na pista	184
	Defeito nas esferas	186
TOTAL		**940**

Tabela 2: Classificação das séries temporais utilizadas.
Fonte: Elaboração própria.

A simulação de defeito de desbalanceamento é feita adicionando uma carga assimétrica no rotor. Para simular vários cenários, foram utilizadas cargas de 6g a 35g. Abaixo é indicado quantas séries temporais foram consideradas para cada carga de desbalanceamento utilizada.

Peso da carga(g)	Quantidade de Séries
6	49
10	48
15	48
20	49
25	47
30	47
35	45
Total	**333**

Tabela 3: Séries temporais de simulação de desbalanceamento.
Fonte: Elaboração própria.

Como mencionado, as falhas no rolamento podem ser de 3 tipos diferentes: falha nas esferas, falha na gaiola e falha na pista. Falhas em rolamentos são pouco perceptíveis quando não há desbalanceamento, por isso foram induzidos desbalanceamentos para evidenciar os defeitos nos rolamentos, utilizando pesos diferentes, variando de 0g a 35g. A quantidade de séries temporais para cada um destes casos é apresentado na tabela abaixo

Tipo de falha	Peso da carga(g)	Quantidade de séries
Falha na pista	0	49
	6	48
	20	49
	35	42
Falha nas esferas	0	49
	6	49
	20	49
	35	37
Falha na gaiola	0	50
	6	49
	20	49
	35	38
Total		**558**

Tabela 4: Séries temporais de simulação de falha no rolamento.
Fonte: Elaboração própria.

Por se tratar de um estudo que visa classificar dados de sensores de um motor específico, não será possível incluir novos datasets, uma vez que somente dados deste motor devem ser considerados.

Para coleta dos dados foi utilizado o script da figura 6, que é capaz de realizar o download dos arquivos necessários, extraí-los e excluir os arquivos compactados que não

serão utilizados. Após a descompactação, o volume dos dados foi de aproximadamente 14,9 Gb.

```
# Download dos arquivos
urllib.request.urlretrieve('https://www02.smt.ufrj.br/~offshore/mfs/database/mafaulda/imbalance.zip', BASE_PATH + '/data/imbalance.zip')
urllib.request.urlretrieve('https://www02.smt.ufrj.br/~offshore/mfs/database/mafaulda/normal.zip', BASE_PATH + '/data/normal.zip')
urllib.request.urlretrieve('https://www02.smt.ufrj.br/~offshore/mfs/database/mafaulda/underhang.zip', BASE_PATH + '/data/underhang.zip')

# extração dos arquivos
with zipfile.ZipFile(BASE_PATH + '/data/imbalance.zip', 'r') as zip_ref:
    zip_ref.extractall(BASE_PATH + 'data')
with zipfile.ZipFile(BASE_PATH + '/data/normal.zip', 'r') as zip_ref:
    zip_ref.extractall(BASE_PATH + 'data')
with zipfile.ZipFile(BASE_PATH + '/data/underhang.zip', 'r') as zip_ref:
    zip_ref.extractall(BASE_PATH + 'data')

# excluir arquivos .zip
os.remove(BASE_PATH + '/data/normal.zip')
os.remove(BASE_PATH + '/data/imbalance.zip')
os.remove(BASE_PATH + '/data/underhang.zip')
```

Figura 6: Script de coleta de dados.
Fonte: Elaboração própria

3. Tratamento de Dados

Inicialmente, os dados originais não possuíam um cabeçalho, o que tornava a identificação da feature mais difícil. Para facilitar o manuseio das informações, a inserção do cabeçalho foi o primeiro passo no tratamento de dados.

Índice	Cabeçalho	Informação
0	tacometro	tacômetro
1	rol_int_axial	acelerômetro axial do rolamento inferior
2	rol_int_radial	acelerômetro radial do rolamento inferior
3	rol_int_tangencial	acelerômetro tangencial do rolamento inferior
4	rol_ext_axial	acelerômetro axial do rolamento externo
5	rol_ext_radial	acelerômetro radial do rolamento externo
6	rol_ext_tangencial	acelerômetro tangencial do rolamento externo
7	microfone	microfone

Tabela 5: Cabeçalhos dos dados.
Fonte: Elaboração própria.

Nesta etapa de tratamento dos dados, serão verificados dados faltantes, dados com tipo inconsistentes e outliers. Para simplificar estas verificações de outliers, foi escolhida aleatoriamente apenas uma série temporal de cada classe, uma vez que os dados de sensores de máquinas tendem a manter o mesmo padrão de operação, trazendo características semelhantes para séries de mesma classe.

Para busca de dados faltantes ou com tipo incorreto, utilizou-se a função **check_inconsistent_values.** Porém, não foi detectado nenhum registro com essas características.

```
def check_inconsistent_values(path_names):
    for file in path_names:
        data = pd.read_csv(file, header=None)
        nullValues = data.isnull().sum()
        inconsistent_type = data.dtypes != 'float64'
        if any(nullValues > 0):
            print('CSV file {} has null values'.format(file))
        if any(inconsistent_type):
            print('CSV file {} has inconsistent type'.format(file))


normal_file_names = glob.glob(BASE_PATH + '/data/normal/*.csv')
imbalance_file_names = glob.glob(BASE_PATH + '/data/imbalance/**/*.csv')
underhang_files_names = glob.glob(BASE_PATH + '/data/underhang/**/*.csv', recursive=True)

check_inconsistent_values(normal_file_names)
check_inconsistent_values(imbalance_file_names)
check_inconsistent_values(underhang_files_names)
```

Figura 7: Função para validação do dataset.
Fonte: Elaboração própria

Para verificar como os valores das features estão distribuídos, foi utilizado um boxplot em que foi plotada uma das séries temporais classificada como normal, disponível no arquivo **61.44.csv**.

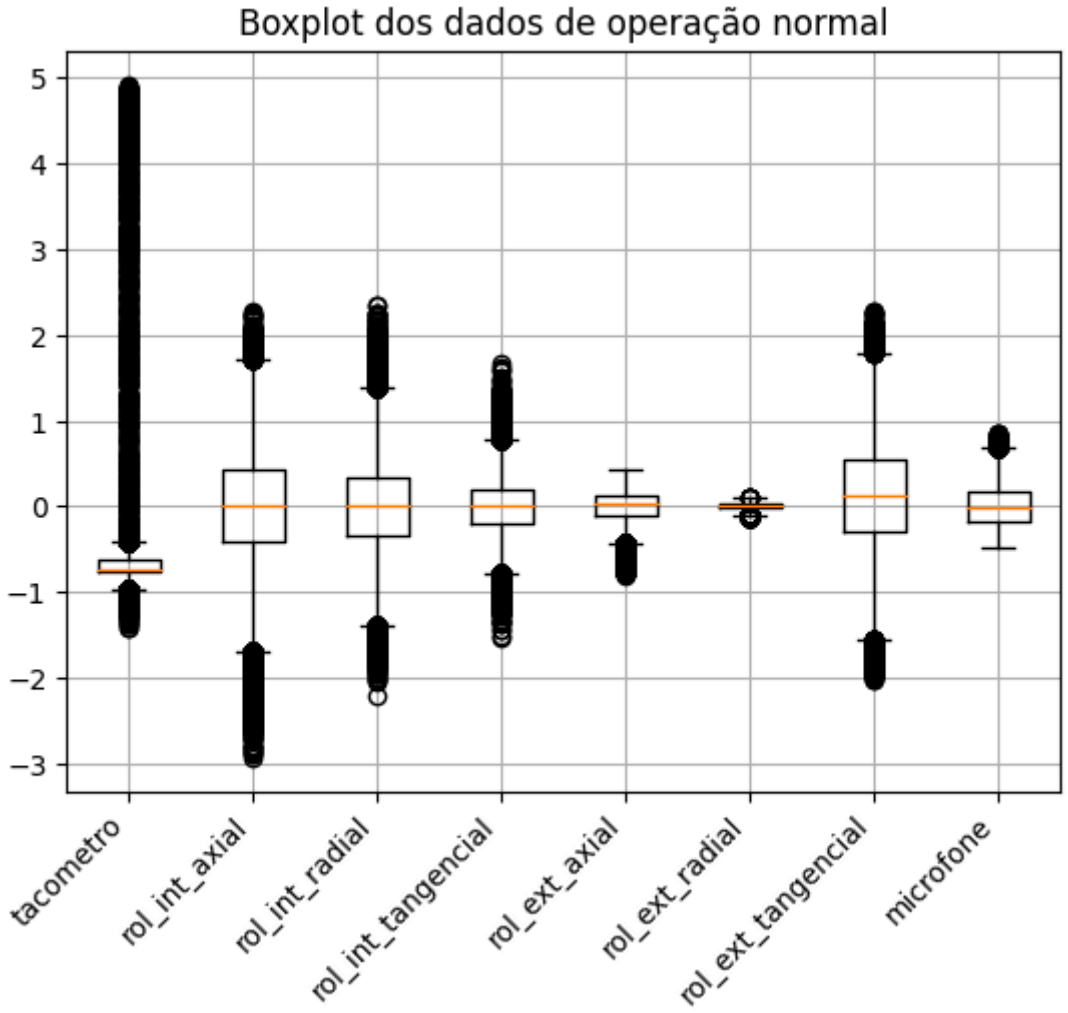


Figura 8: Boxplot da série disponível em 61.44.csv.
Fonte: Elaboração própria.

Os círculos no boxplot, são dados que podem ser considerados outliers, pois estão fora do intervalo de $\pm\ 1,5\ *\ IQR$, onde IQR é a diferença entre o terceiro e primeiro quartil. A maioria destes outliers são dados do tacômetro. Segundo a documentação do dataset, as medições dos sensores foram feitas em várias rotações diferentes, variando de 737 rpm a 3686 rpm. Essa grande variação da rotação durante a leitura dos sensores, pode explicar a presença de dados considerados outliers.

Usando um gráfico pôde-se fazer uma análise melhor destes dados de outliers. Com a função **plot_outliers** foi possível criar um gráfico dos dados da feature, marcando os outliers e exibindo os limites dos dados considerados outliers ($\pm\ 1,5\ *\ IQR$).

```python
def plot_outliers(feature_data, feature_name):
  # Calcular o IQR e identificar outliers
  Q1 = feature_data.quantile(0.25)
  Q3 = feature_data.quantile(0.75)
  IQR = Q3 - Q1
  lower_limit_outliers = Q1 - 1.5 * IQR
  upper_limit_outliers = Q3 + 1.5 * IQR
  outliers = (feature_data < lower_limit_outliers) | (feature_data > upper_limit_outliers)

  # reduzir quantidade de dados para melhor visualização
  sample_size = len(feature_data) // 10
  sample_data = feature_data[:sample_size]
  sample_outliers = outliers[:sample_size]

  # Plotar a série temporal e destacar os outliers e limites dos dados considerados outliers
  plt.figure(figsize=(12, 6))
  plt.plot(sample_data, label='Dados')
  plt.scatter(sample_data.index[sample_outliers], sample_data[sample_outliers], color='red', label='Outliers')
  plt.axhline(y=lower_limit_outliers, color='blue', linestyle='--', label='lower limit outliers')
  plt.axhline(y=upper_limit_outliers, color='green', linestyle='--', label='upper limit outliers')
  plt.title(f'Dados do {feature_name} com Outliers Destacados')
  plt.xlabel('Amostras')
  plt.ylabel('Valores')
  plt.legend(loc='upper right')
  plt.show()
```

Figura 9: Função plot_outliers.
Fonte: Elaboração própria

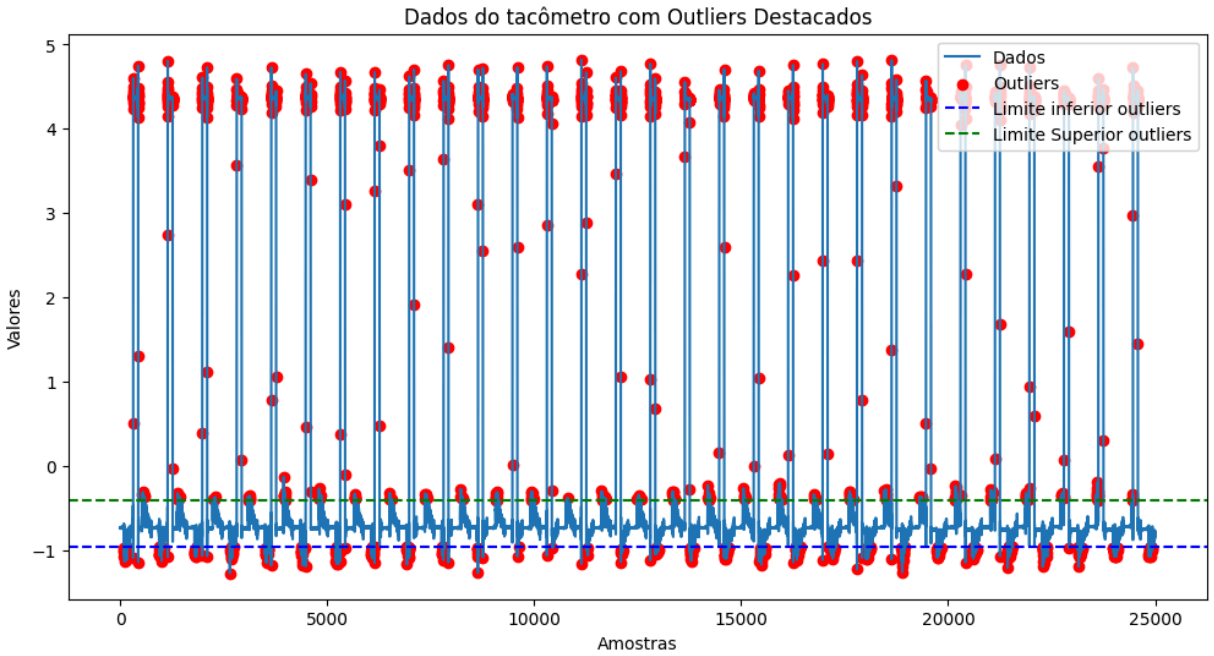


Figura 10: Outliers dos dados do tacômetro de série normal e limites de $\pm\ 1,5\ *\ IQR$.
Fonte: Elaboração própria.

Como é possível verificar no gráfico acima, a grande maioria dos dados estão concentrados por volta do valor -0.72, que coincide com sua respectiva mediana. Os possíveis outliers seguem um padrão repetitivo no tempo, o que pode ser entendido como sendo dados válidos dos sensores, e não outliers.

O mesmo acontece para demais as features e para outras séries temporais. O gráfico abaixo, com dados do microfone, apresenta o mesmo comportamento cíclico de outliers. Trata-se da série disponível no arquivo **13.312.csv**, classificada como desbalanceada e com uma massa de 15g para simular este desbalanceamento.

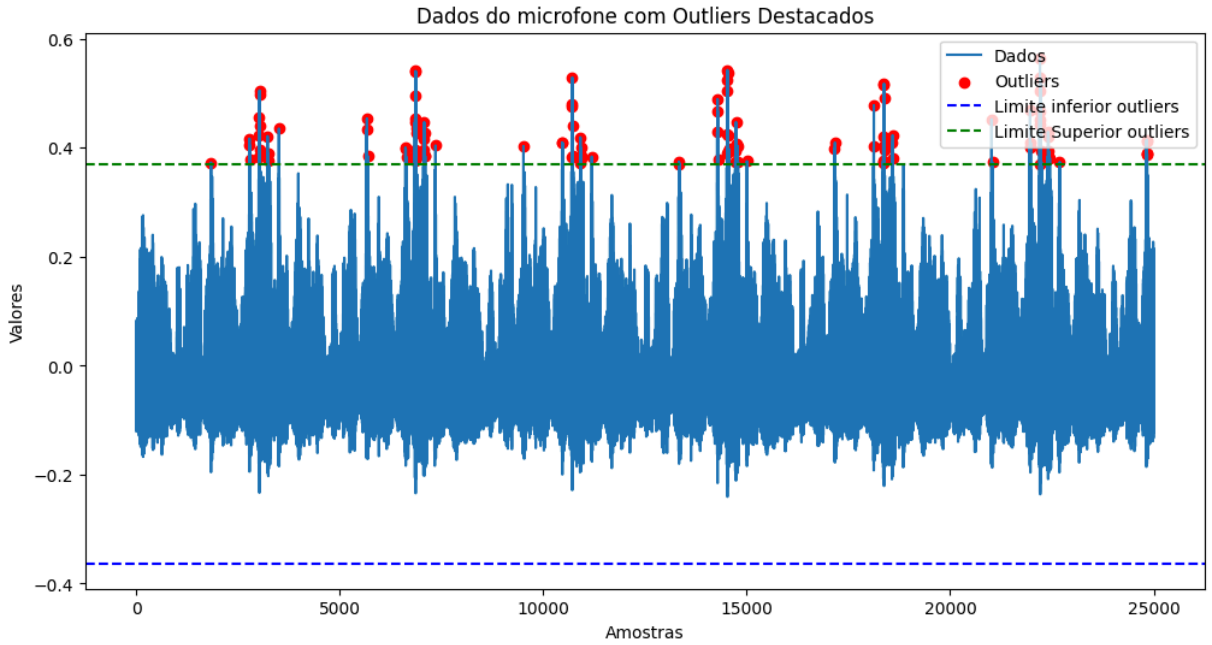


Figura 11: Outliers de dados do microfone de série desbalanceada e limites de $\pm\ 1,5 * IQR$.
Fonte: Elaboração própria.

Também há casos onde os outlier se justificam por caracterizar o defeito na máquina, portanto é esperado que haja dados discrepantes. Como por exemplo, no gráfico abaixo onde são exibidos dados do arquivo **15.36.csv**, que é um série com defeito nas esferas do rolamento inferior com um desbalanceamento com uma massa de 20g.

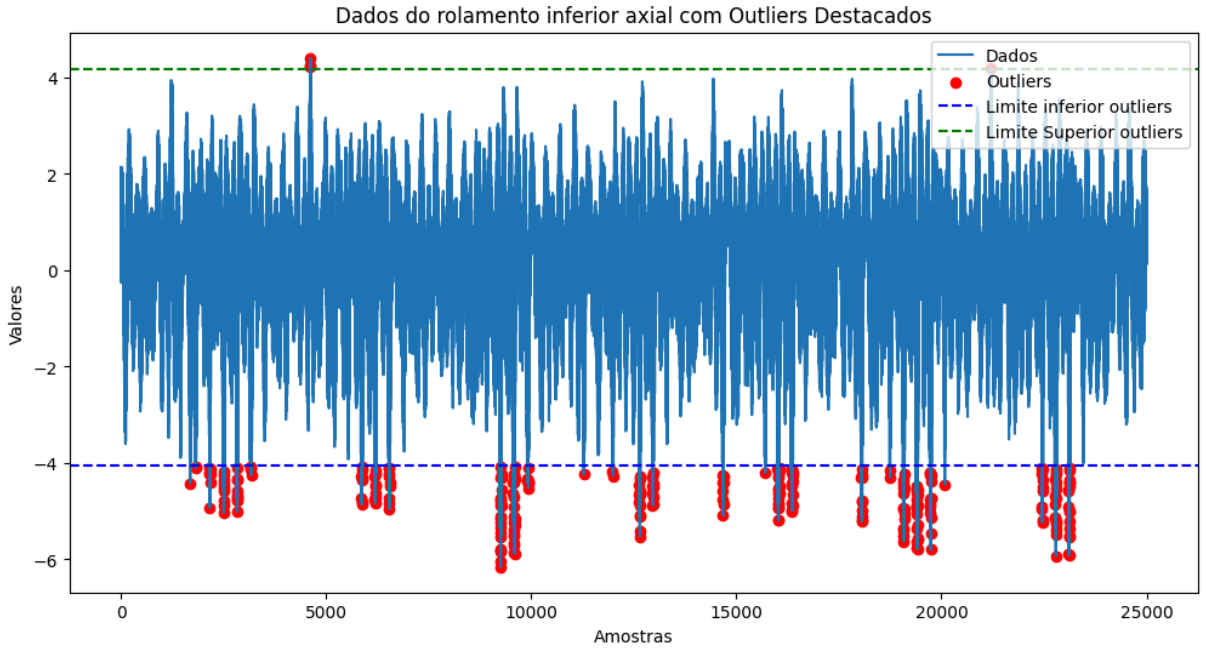


Figura 12: Outliers de dados do rolamento inferior axial de série com falha no rolamento e limites de $\pm\ 1,5\ *\ IQR$.
Fonte: Elaboração própria.

Como não foi identificado nenhum dado faltante, inválido ou outlier, não há necessidade de nenhum tratamento com relação a esses pontos.

4. Análise e Exploração dos Dados

Para exploração dos dados, foi utilizada uma série de cada classificação estabelecida, escolhida aleatoriamente.

Exibindo dados de 1 segundo de cada feature separadamente num gráfico, podemos identificar alguns padrões:

- O padrão formado pelos sensores axial, radial e tangencial do rolamento externo é bastante semelhante.
- O tacômetro é bastante influenciado pelas falhas de desbalanceamento e rolamento inferior, quando comparados com o gráfico em operação normal é possível perceber diferenças na frequência dos picos formados.

- O defeito no rolamento interno, causou muita variação na leitura do acelerômetro axial do rolamento externo. Em operação normal e desbalanceado, este sensor media em torno de 0,5V e -0,5V, em operação com defeito no rolamento, apresentou leitura entre 5V e -5V.

  Os gráficos a seguir apresentam estes padrões.

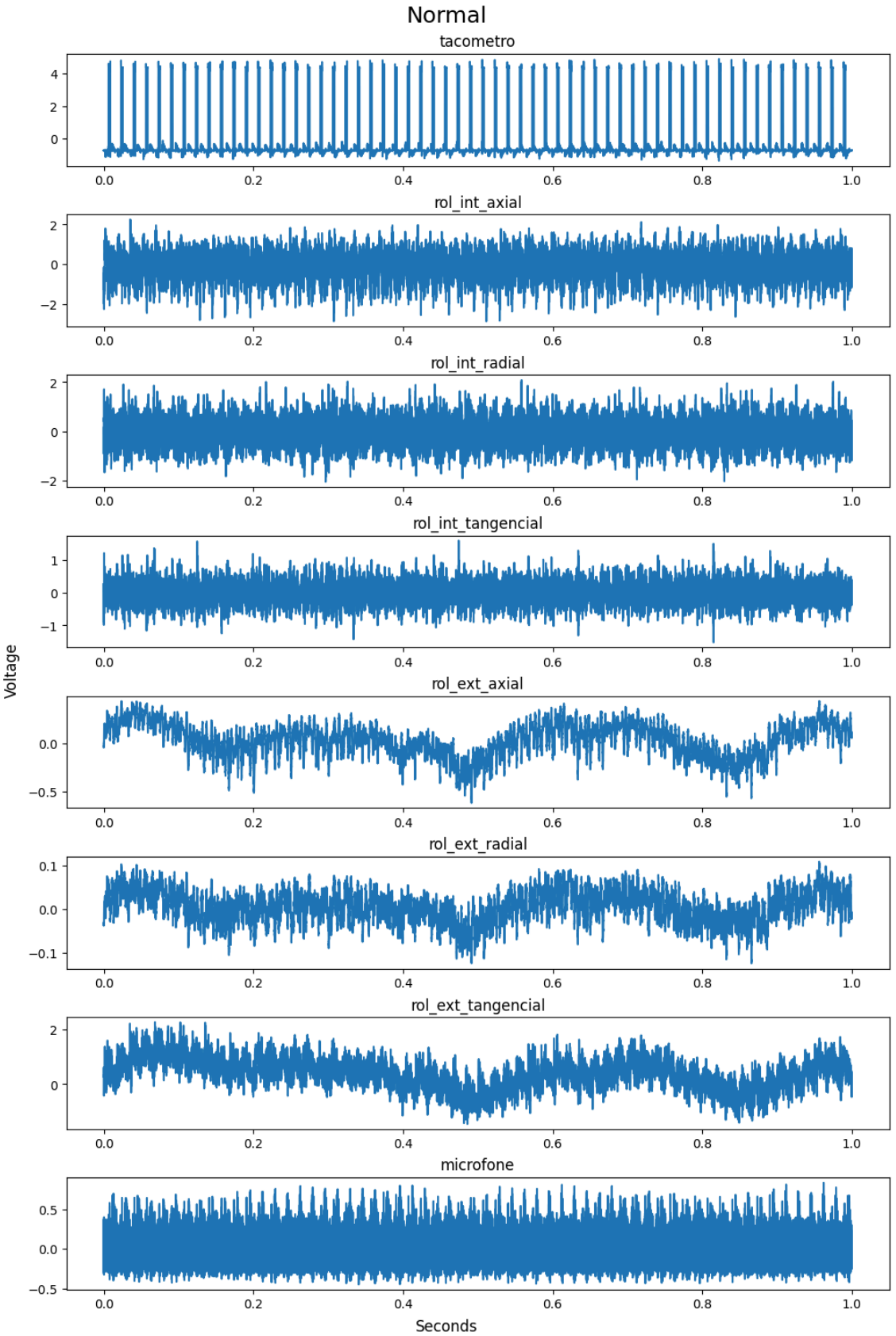


Figura 13: Sinal dos sensores em operação Normal.
Fonte: Elaboração própria.

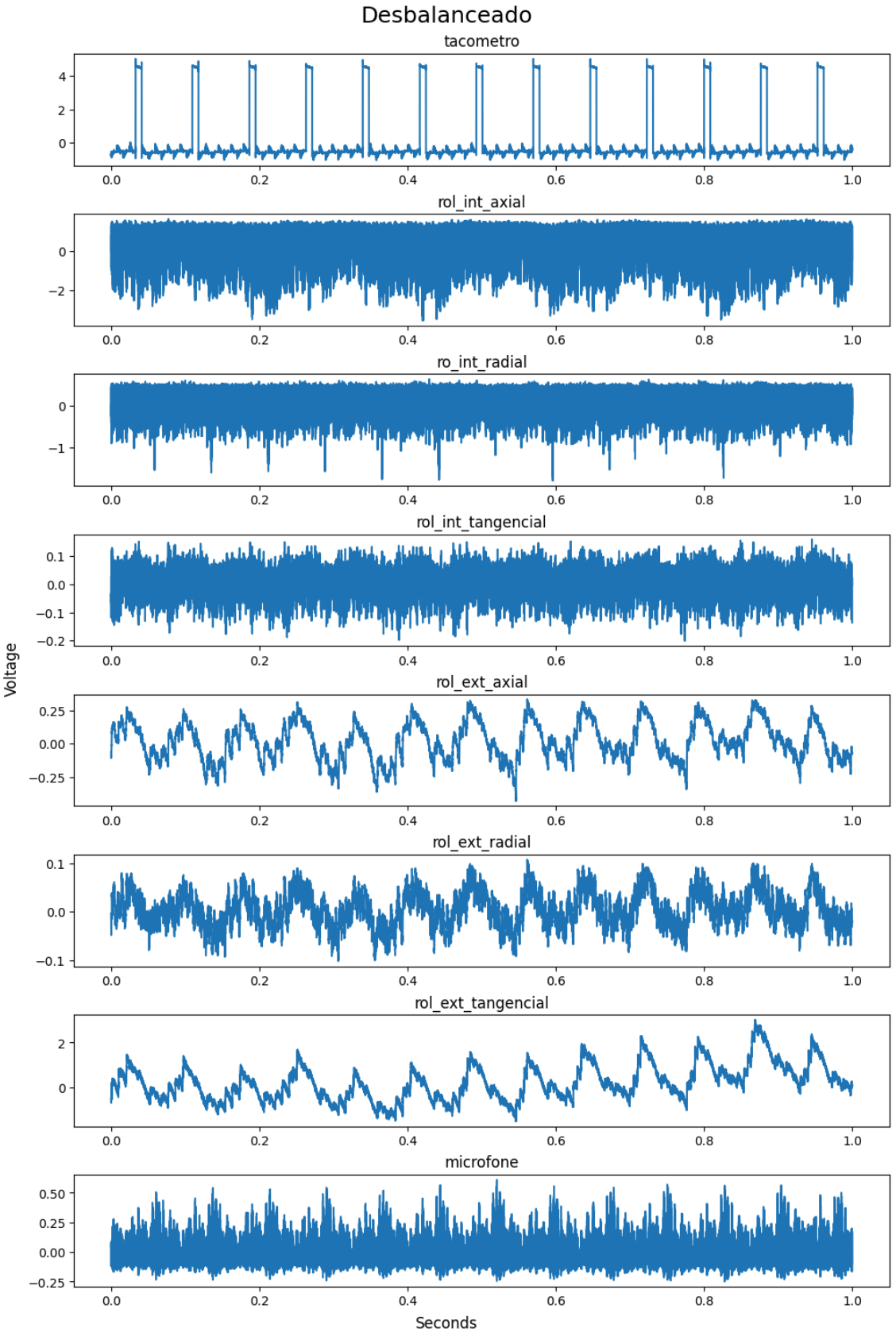


Figura 14: Sinal dos sensores em operação desbalanceada.
Fonte: Elaboração própria.

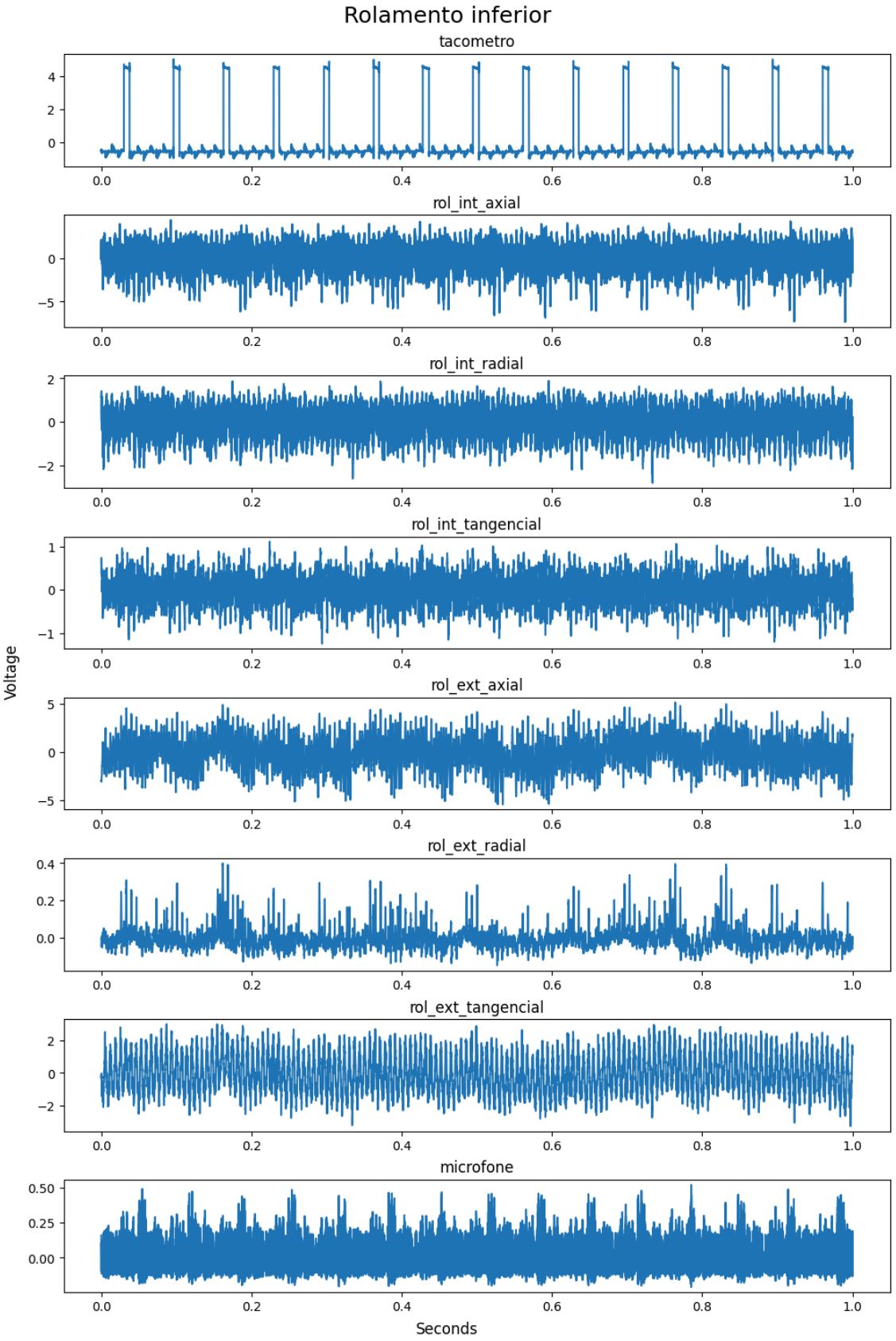


Figura 15: Sinal dos sensores em operação com falha no rolamento inferior.
Fonte: Elaboração própria.

Ao criar um mapa de calor de correlação entre as features, pôde-se verificar se havia alta correlação entre elas, apresentando suas redundâncias e permitindo remoção de alguma feature, evitando o overfitting e melhorando a interpretabilidade do modelo.

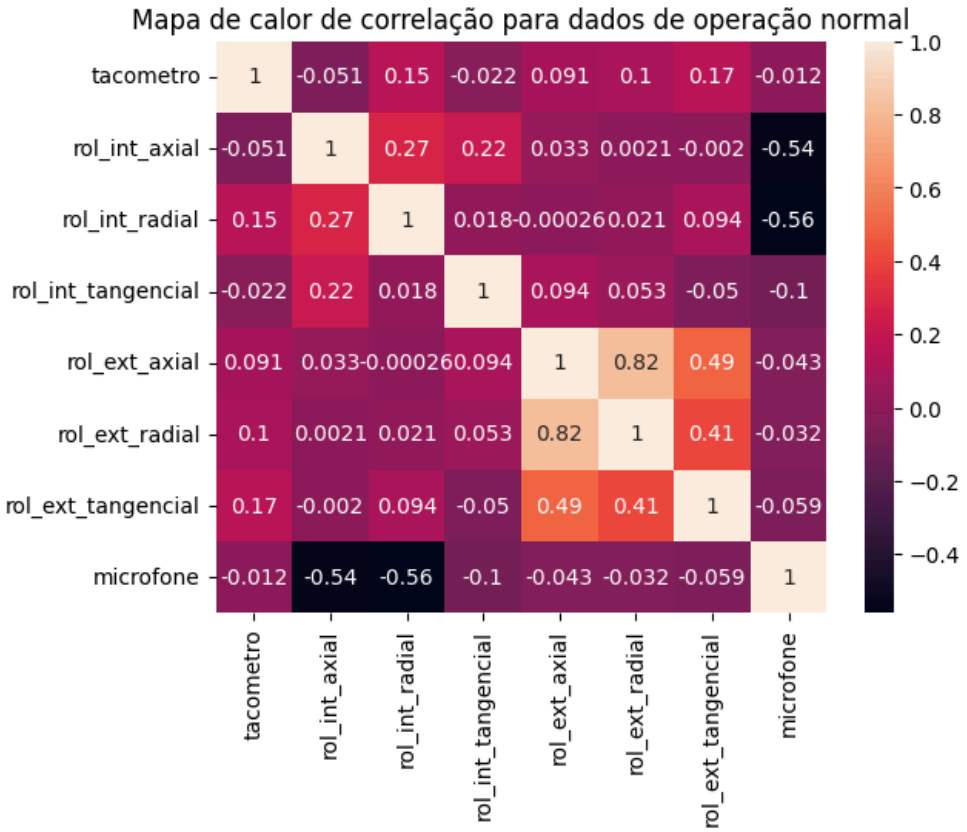


Figura 16: Mapa de calor de correlação para dados de operação normal.
Fonte: Elaboração própria.

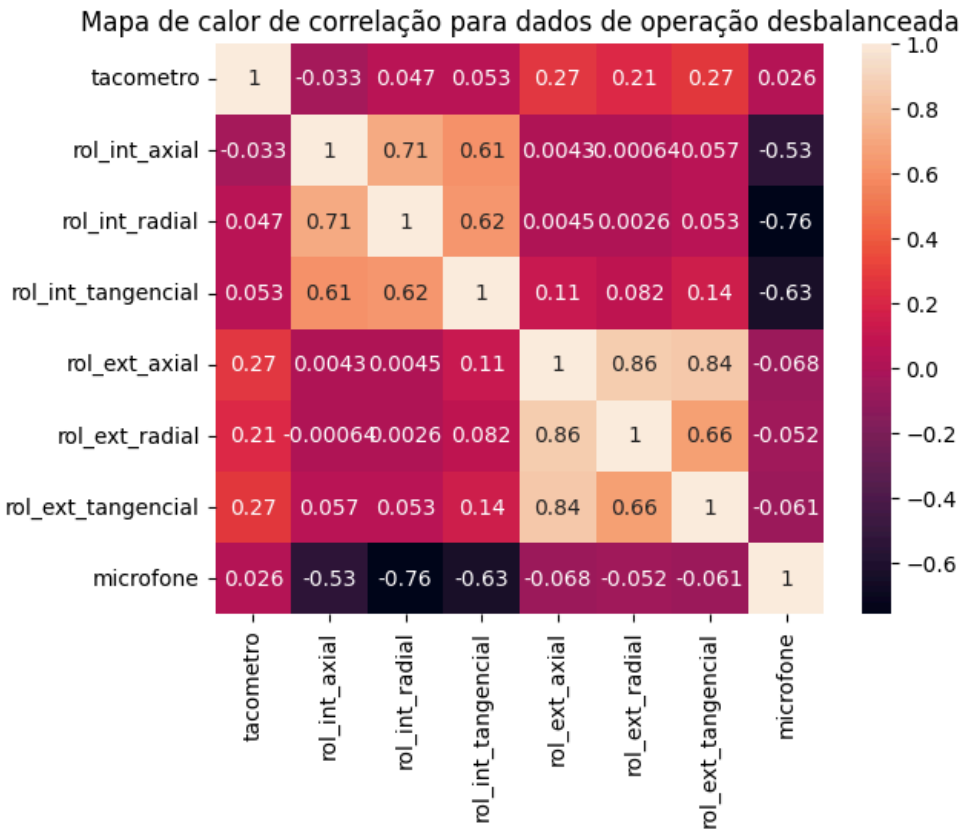


Figura 17: Mapa de calor de correlação para dados de operação desbalanceada.
Fonte: Elaboração própria.

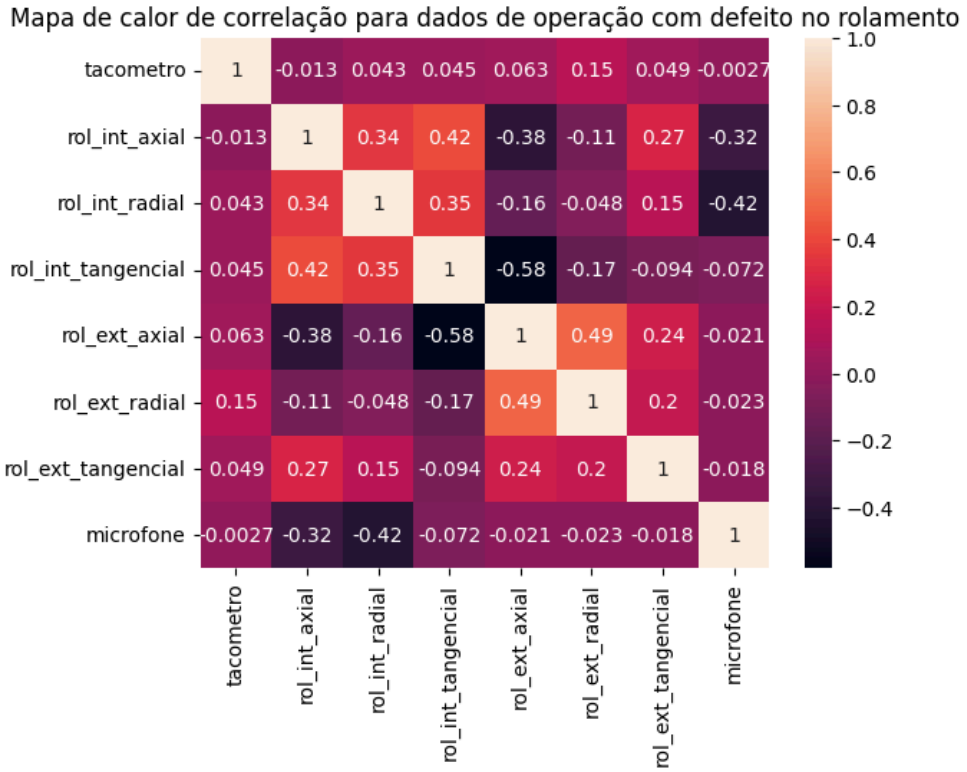


Figura 18: Mapa de calor de correlação para dados de operação com defeito no rolamento.
Fonte: Elaboração própria.

A correlação só é considerada forte para valores maiores que 0,7 ou menores que -0,7 (DA SILVA, Fernando, 2023). Usando este parâmetro, foi possível observar que só houve alta correlação em alguns casos, como por exemplo, entre os sensores do rolamento externo, sensores do rolamento interno em operação desbalanceada e acelerômetro radial de rolamento interno com o microfone em operação desbalanceada. Portanto, a correlação das features não é um fator que influenciará negativamente nos modelos que serão treinados.

Features com maior desvio padrão podem ser um indicativo de ser uma feature com mais informações relevantes. Uma técnica de seleção de features é a remoção das que não atingem um certo nível de variância (ARAUJO, Raquel, 2023).

	Normal	Desbalanceado	Rolamento inferior
tacometro	1.785603	1.585844	1.620201
rol_int_axial	0.627288	1.097578	1.538240
rol_int_radial	0.505619	0.344077	0.602366
rol_int_tangencial	0.295160	0.046907	0.356580
rol_ext_axial	0.164101	0.146979	1.990002
rol_ext_radial	0.036356	0.038289	0.059382
rol_ext_tangencial	0.593672	0.860979	1.193174
microfone	0.215691	0.124477	0.106651

Tabela 6: Desvio padrão das features de cada classe de operação.
Fonte: Elaboração própria.

A tabela acima exibe o desvio padrão das features para cada classe de operação, onde as cores mais escuras indicam valores mais altos. Como descrito anteriormente, os dados foram coletados com o motor em uma diversidade de rotações, o que explica o alto desvio padrão nos dados do tacômetro em todas as classificações de operação.

Algo interessante a ressaltar, assim como no gráfico da figura 15, é que o defeito no rolamento inferior causa muita influência nos sensores do rolamento exterior.

Em modo de operação normal, nenhuma feature, exceto o tacômetro, se destaca com um desvio padrão alto, que mostra que em modo normal de operação, a leitura dos sensores não sofre grandes alterações, funcionando de forma mais homogênea.

5. Processamento dos dados

Como destacado em tópicos anteriores, a base de dados MaFaulDa apresenta uma grande quantidade de dados, o que elevaria muito o tempo de treinamento dos modelos de classificação. Para resolver este problema, os dados foram reamostrados, reduzindo a frequência de amostragem.

Para realizar a reamostragem foi necessário definir qual seria a nova frequência de amostragem. Utilizar uma frequência alta pode garantir que não haverá perda de informações, porém não reduz significativamente a quantidade de dados para treinamento. Ao utilizar uma frequência muito baixa, a quantidade de dados será menor, porém perde-se informação relevante no processo. Para definir qual seria a frequência de amostragem adequada, utilizou-se o teorema de Nyquist, que diz que para não termos perdas de informação, a frequência de amostragem (f_s), chamada de frequência de Nyquist, deve ser o dobro da frequência do sinal (f_m): $f_s \geq 2f_m$ (NUNES, Tassia, 2023).

O sinal dos sensores é composto por frequências diferentes, e para identificar a frequência de Nyquist, foi necessário identificar qual a maior frequência que compunha este sinal. Com a Transformada Rápida de Fourier - FFT (HAYKIN, Simon; VAN VEEN, Barry, 2001), é possível decompor um sinal do domínio do tempo para o domínio de frequência, o que permite identificar a frequência máxima que compõe o sinal.

A função **apply_rfft** recebeu como parâmetros o sinal dos sensores e a frequência desse sinal, e aplicou a Transformada Rápida de Fourier com a função **rfft** disponível no pacote Numpy.

```python
def apply_rfft(signal, sampling_rate):
    rfft_result = np.abs(np.fft.rfft(signal, axis=0))[1:,:]
    rfft_freqs = np.fft.rfftfreq(len(signal), d=1/sampling_rate)[1:]

    return rfft_result, rfft_freqs
```

Figura 19: Função apply_rfft.
Fonte: Elaboração própria.

A função **rfft** retornou um array de números complexos com as componentes de amplitude e fase das frequências do sinal. Pôde-se aplicar o módulo neste array, pois para a análise proposta neste trabalho, apenas a amplitude é relevante, não sendo necessário considerar a parte imaginária com informação da fase. Também foi removida a primeira linha do array que contém informações da média do sinal. Por fim, com auxílio da função **rfftfreq,** gerou-se um array contendo as frequências referentes às amplitudes que foram calculadas pela função **rfft**. Esta função foi executada com dados de uma série temporal de cada uma das classificações, a fim de identificar a maior frequência, e, definir a frequência de Nyquist. Para apoiar nesse cálculo foi criada a constante **ORIGINAL_SAMPLE_RATE,** contendo a frequência original dos dados (50000 Hz), que foi usada com parâmetro da função **apply_fft**.

Nos gráficos da figura 20 estão exibidos os dados de séries com as 3 classificações, após realizar a Transformada Rápida de Fourier. O eixo X corresponde às frequências que compõem o sinal original, e no eixo Y a intensidade desta frequência.

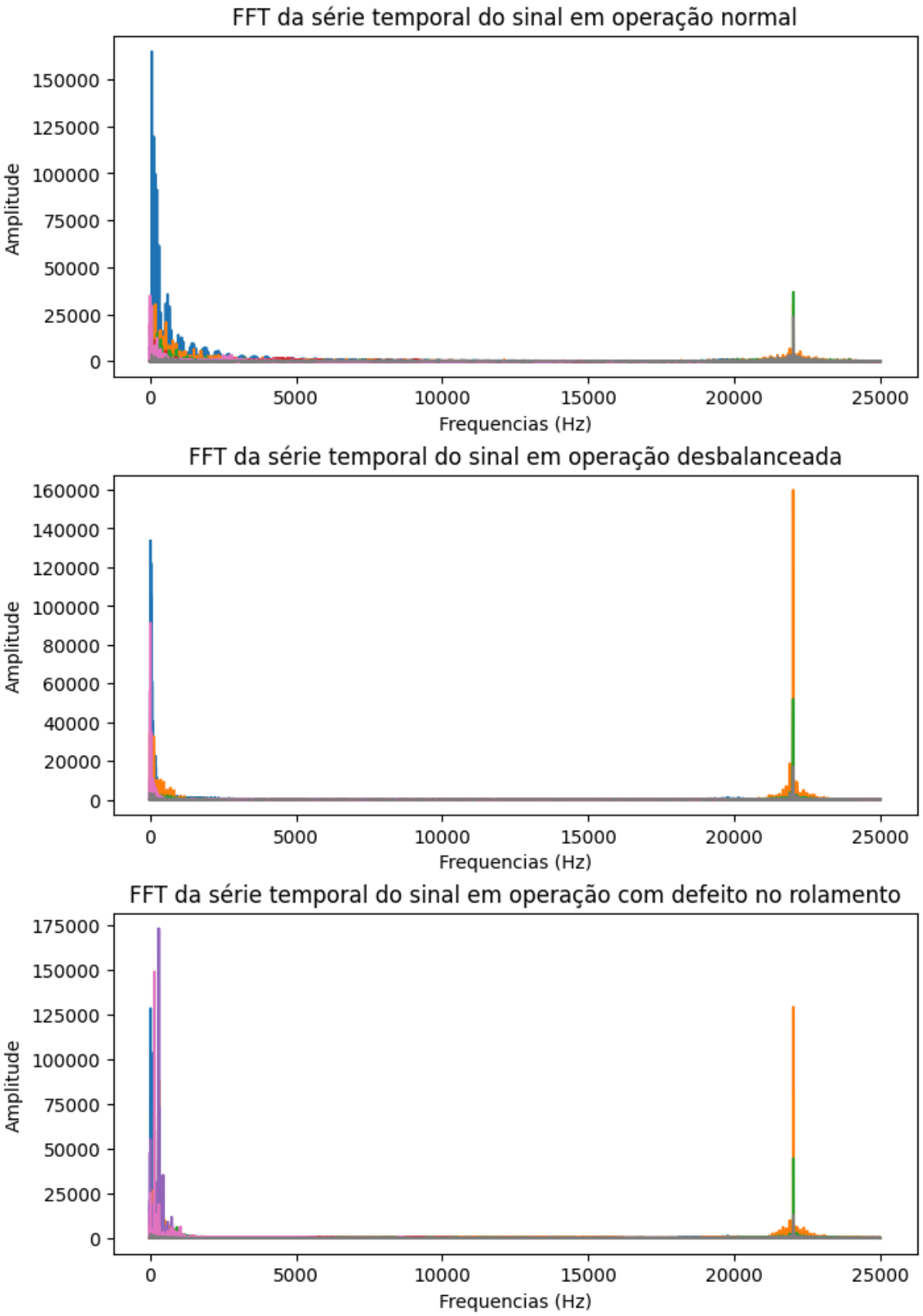


Figura 20: Séries após a FFT.
Fonte: Elaboração própria.

Foi possível observar que, nos 3 casos, há frequências com grande intensidade entre 0Hz e 1000Hz, e entre, aproximadamente, 21000Hz e 23000Hz. Porém entre 1000Hz e 21000Hz praticamente não há dados, ou seja, não há frequências que compõem o sinal original, ou, a intensidade destas frequências é tão baixa que podem ser desconsideradas.

Recorrendo ao teorema de Nyquist, a frequência de amostragem deve ser o dobro da maior frequência que compõe o sinal. Entretanto, utilizando uma frequência em torno de 23000 Hz, aproximadamente a frequência máxima, não tem-se uma redução considerável da quantidade de dados para o treinamento.

Na figura 21 observou-se que a maioria das frequências com maior intensidade estão até, aproximadamente, 350Hz. Ou seja, este valor foi considerado como a frequência relevante mais alta do sinal, o que remete à uma frequência de reamostragem, de acordo com o teorema de Nyquist, de 700 Hz.

Reamostrando o sinal em 700 Hz houveram perdas de informação no sinal, porém, após testes, verificou-se que os modelos treinados com este sinal reamostrado performaram de maneira satisfatória e com tempo reduzido de treinamento.

No código desenvolvido para este trabalho foi criada a constante **TARGET_RATE** para armazenar o valor adotado para a frequência de Nyquist (700Hz).

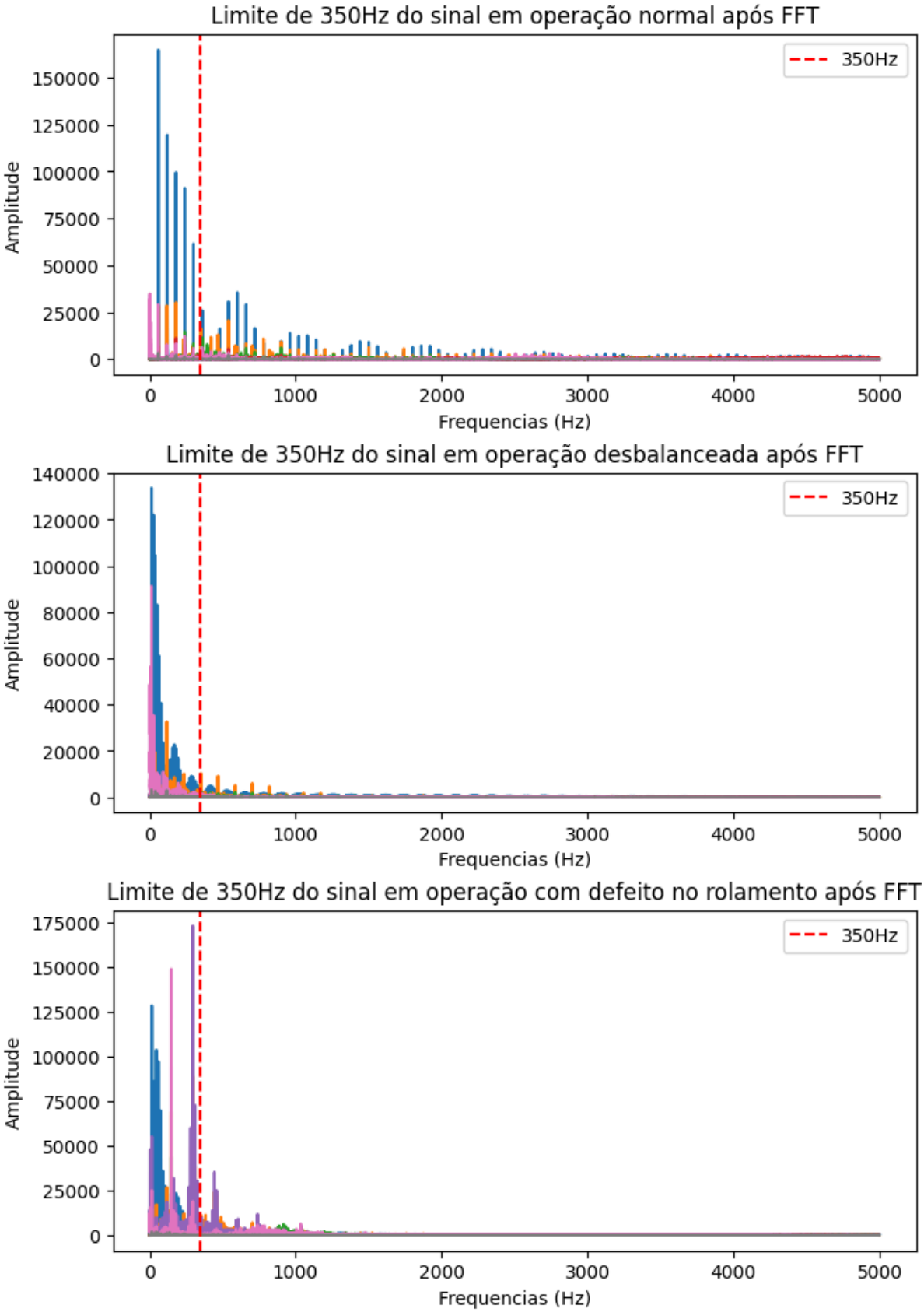


Figura 21: Séries após FFT, evidenciando a frequência de 350Hz.
Fonte: Elaboração própria.

Ao realizar um downsampling, ou seja, reamostrar um sinal reduzindo sua frequência, para uma frequência inferior à frequência de Nyquist, ocorreu um fenômeno chamado aliasing, em que as frequências mais altas são distorcidas, gerando frequências mais baixas no sinal reamostrado. Para evitar o aliasing, foi utilizado um filtro passa-baixa no sinal antes de realizar a reamostragem. Com isto foi possível garantir que a frequência mais alta no sinal seria 350 Hz, uma frequência adequada para a taxa de amostragem desejada de 700 Hz.

A função **resample_signal** é capaz de realizar o filtro passa-baixa na metade da frequência desejada, e realizar o downsampling dos dados. Este processo foi realizado em cada feature de cada série temporal.

```python
def resample_signal(signal_data, original_sr, target_sr):
  num_channels = signal_data.shape[1]
  num_samples = int(signal_data.shape[0] * target_sr / original_sr)
  resampled_data = np.zeros((num_samples, num_channels))

  for channel_index in range(num_channels):
    channel_signal = signal_data[:, channel_index]  # Obtém o sinal do canal atual

    # cria um filtro passa-baixa para evitar aliasing
    cutoff_freq = target_sr / 2  # Frequência de corte (metade da nova taxa de amostragem)
    b, a = signal.butter(2, cutoff_freq / (original_sr / 2), 'lowpass')  # Cria um filtro Butterworth

    # Aplica o filtro ao sinal
    channel_signal = signal.lfilter(b, a, channel_signal)

    # realiza a reamostragem do sinal
    resampled_signal = signal.resample(channel_signal, num_samples)
    resampled_data[:, channel_index] = resampled_signal

  return resampled_data
```

Figura 22: Função resample_signal.
Fonte: Elaboração própria.

No gráfico apresentado na figura 23 foi possível visualizar dados do tacômetro na versão do sinal original e do sinal reamostrado, tendo sido usada como base uma série temporal classificada como normal.

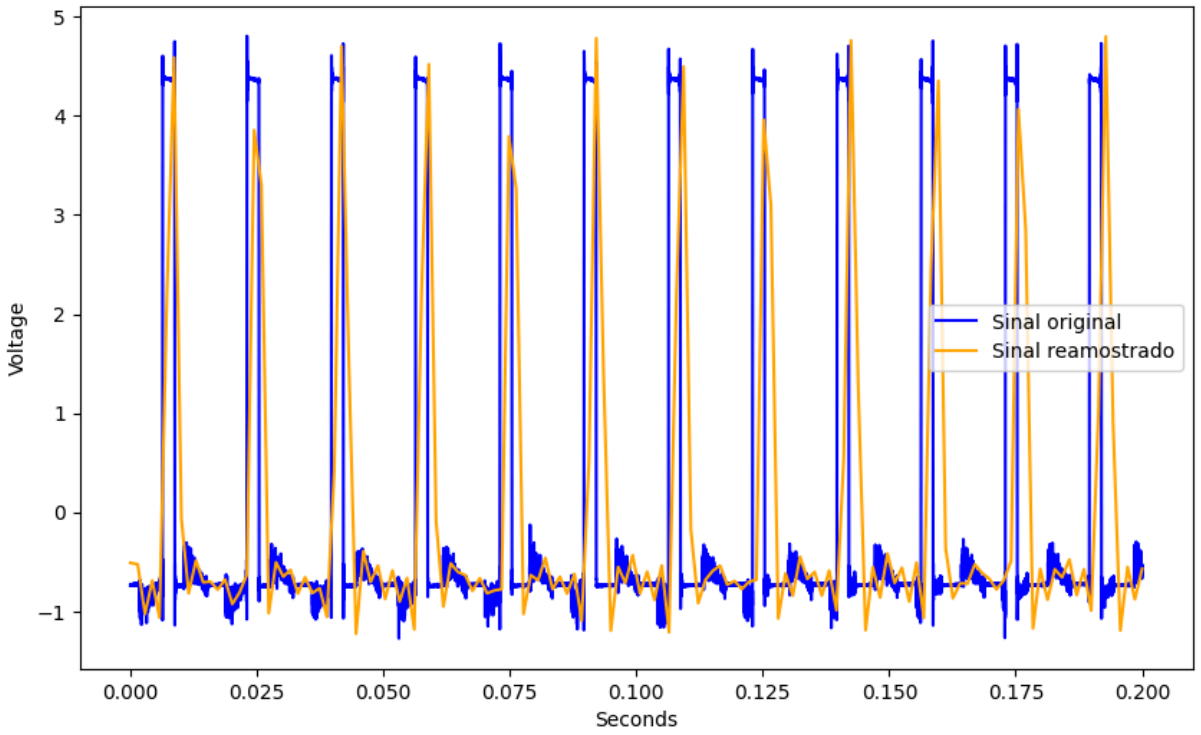


Figura 23: Sinal original x sinal reamostrado.
Fonte: Elaboração própria.

Notou-se que o sinal reamostrado não continha todos os detalhes do sinal original, mas que possuía os mesmos padrões, sendo o suficiente para os modelos de machine learning classificarem os dados.

Após essa reamostragem, cada série temporal que continha 250.000 registros, passou a ter 3500, o que reduziu significativamente o tempo de treinamento sem degradar de modo considerável o desempenho dos modelos.

As frequências que compõem o sinal, geralmente são mais relevantes que os dados do sinal bruto, pois destacam padrões e características importantes que podem ser difíceis de discernir no domínio do tempo. Assim, foi utilizado novamente a Transformada Rápida de Fourier para obter estas características em frequências. Estes são os dados analisados pelos modelos de classificação utilizados.

Para o processamento de todo o dataset foi criada a função **process_data**, que recebeu uma lista de arquivos CSV contendo os dados, fez a reamostragem e, então, aplicou a Transformada Rápida de Fourier. Ao final desta execução, obteve-se todo o dataset processado.

```python
def process_data(files_folder):
  processed_data = []

  for i in range(len(files_folder)):
    data = pd.read_csv(files_folder[i], header=None)
    data = data.to_numpy();
    signal_resampled = resample_signal(data, ORIGINAL_SAMPLE_RATE, TARGET_RATE)
    spectrum, xf = apply_rfft(signal_resampled, TARGET_RATE)

    processed_data.append(spectrum)

  return np.array(processed_data)
```

Figura 24: Função process_data.
Fonte: Elaboração própria.

Para que não houvesse necessidade de reprocessamento, os arquivos processados foram salvos com a biblioteca **pickle**.

```python
import pickle

data_normal = process_data( glob.glob(BASE_PATH + '/data/normal/*.csv'))
file_path = BASE_PATH + '/processed data/normal.pkl'
with open(file_path, 'wb') as f:
   pickle.dump(data_normal, f)

data_imbalance= process_data( glob.glob(BASE_PATH + '/data/imbalance/**/*.csv'))
file_path = BASE_PATH + '/processed data/imbalance.pkl'
with open(file_path, 'wb') as f:
   pickle.dump(data_imbalance, f)

data_underhang = process_data( glob.glob(BASE_PATH + '/data/underhang/**/*.csv', recursive=True))
file_path = BASE_PATH + '/processed data/underhang.pkl'
with open(file_path, 'wb') as f:
   pickle.dump(data_underhang, f)
```

Figura 25: Processamento e armazenamento dos dados processados.
Fonte: Elaboração própria.

Uma vez que os dados foram processados, eles foram divididos em dados de treino e de teste, sendo que a base de teste conteve 25% do total de dados, resultando em 705 séries para treino e 235 para teste.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

X = np.concatenate((
    data_normal,
    data_imbalance,
    data_underhang
    ), axis=0)

y_classes = np.concatenate((
    np.full(data_normal.shape[0], 'normal'),
    np.full(data_imbalance.shape[0], 'imbalance'),
    np.full(data_underhang.shape[0], 'underhang')
    ), axis=0)

label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y_classes)


X_train, X_test, y_train, y_test  = train_test_split(X, y, random_state=1)
```

Figura 26: Separação de dados de treino e de teste.
Fonte: Elaboração própria.

6. Criação de Modelos de Machine Learning

Para treinamento, foram escolhidos 3 modelos de machine learning com boas capacidades para classificação de séries temporais. O modelo LSTM é projetado para lidar com a natureza sequencial dos dados, utilizando mecanismos de memória, além de poder aprender padrões a longo prazo. O modelo CNN se destaca por conseguir capturar padrões locais das séries temporais por meio dos filtros convolucionais. E por fim o GRU, que possui um funcionamento semelhante ao LSTM porém numa arquitetura mais simplificada.

A fim de evitar o overfitting nos treinamentos, e, buscando as melhores combinações de hiperparâmetros, para todos os modelos foi utilizado o mecanismo de early stopping. O conceito de early stopping se baseia na utilização de uma métrica como referência, e, caso esta não apresente melhora após determinada quantidade de épocas, o treinamento é interrompido antes que ocorra o overfitting. Para o treinamento dos modelos utilizados neste trabalho, foi usada a métrica loss como referência, e o parâmetro patience = 3, que gera interrupção do treinamento após 3 épocas.

Como as classes estavam desbalanceadas, foi necessário aplicar alguma técnica para que isso não comprometesse a performance do modelo treinado. Foram consideradas duas possibilidades: redução da quantidade de dados com classes majoritárias, ou, ponderamento de classes atribuindo pesos diferentes para equilibrar discrepância de quantidade de dados. Em testes, verificou-se que o ponderamento de classes performa melhor.

Para utilização de ponderamento de classes foi necessário criar um dicionário contendo os pesos de cada classe, baseado na quantidade de dados da classe. Empregando a função **compute_class_weight** do pacote **sklearn**, foram criados os pesos para equilibrar as classes. Classes com menor quantidade de amostras, recebem um peso maior, compensando a escassez de dados para treinamento.

```
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_train),
    y=y_train
)
class_weights_dict = dict(enumerate(class_weights))
mapped_dict = {label_encoder.inverse_transform([k])[0]: v for k, v in class_weights_dict.items()}
print(mapped_dict)
```

Figura 27: Função compute_class_weight.
Fonte: Elaboração própria.

A distribuição dos pesos utilizados foi detalhada na tabela 6. Como a classe **normal** possuía uma quantidade de dados significativamente menor, seu peso foi maior em relação às outras classes.

Classe	Peso
normal	6.351351351351352
desbalanceada	0.9251968503937008
rolamento inferior	0.5676328502415459

Tabela 7: Pesos de cada tipo de classe.
Fonte: Elaboração própria.

Estes pesos foram utilizados no treinamento de todos os modelos de machine learning.

6.1. LSTM - Long Short-Term Memory

Dentro do conceito de Rede Neural Recorrente - RNN (CIABURRO, Giuseppe; VENKATESWARAN, Balaji, 2017), existe um tipo especial chamado Rede de Memória de Curto Prazo Longo - LSTM, que são projetadas para superar o problema do desaparecimento do gradiente que aflige as RNNs tradicionais em processamento de sequências longas. Ou seja, elas são capazes de aprender com dependências de longo prazo, o que a faz um ótimo modelo para séries temporais muito longas, como é o caso das utilizadas neste trabalho (Malhotra, Pankaj et al., 2015).

Apesar dos dados de treinamento deste trabalho terem passado por uma Transformada Rápida de Fourier, que removeu a dimensão temporal dos dados, o modelo LSTM ainda performou muito bem, aprendendo os padrões na distribuição das magnitudes das frequências.

A rede neural LSTM utilizada possui 3 camadas:

- Camada Input: Define a forma dos dados de entrada. Apesar de não realizar processamento, é considerada uma camada na biblioteca Keras (The Model class).
- Camada LSTM: Camada que aprende dependência de longo prazo com os dados.
- Camada Dense: Recebe dados da camada anterior, multiplica por pesos, soma vieses e aplica funções de ativação para realizar a classificação dos dad

Para ajuste de hiperparâmetros, foi utilizada a função **GridSearchCV** que testou cada uma das combinações de parâmetros disponíveis.

O modelo foi treinado utilizando validação cruzada com 3 folds: os dados de treinamento foram divididos em 3 partes, e realizados 3 treinamentos para cada combinação de hiperparâmetros. Ao final, o desempenho do modelo apresentou a média atingida utilizando cada fold. Isso reduziu o overfitting, pois o modelo foi avaliado com dados não vistos durante o treinamento.

```
%%time
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Input
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.callbacks import EarlyStopping
import pprint

n_samples, timesteps, features = X_train.shape

def create_lstm_model(lstm_units, optimizer, activation, dense_units):
    model = Sequential()
    model.add(Input(shape=(timesteps, features)))
    model.add(LSTM(units=lstm_units))
    model.add(Dense(units=dense_units,  activation=activation))
    model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model

early_stopping = EarlyStopping(monitor='loss', patience=3, restore_best_weights=True)
model = KerasClassifier(model=create_lstm_model, verbose=1, epochs=30, callbacks=[early_stopping])

param_grid = {
    'model__lstm_units': [50, 100, 200],
    'model__optimizer': ['adam', 'rmsprop'],
    'model__activation':['relu', 'softmax'],
    'model__dense_units':[12, 16, 20],
}

grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)

grid_result = grid.fit(X_train, y_train, class_weight=class_weights_dict)
LSTM_model = grid_result.best_estimator_.model_

print(f"Acurácia {grid_result.best_score_ * 100:.2f}% com dados de treino, usando parâmetros: \n\n{pprint.pformat(grid_result.best_params_)}")
```

Figura 28: Treinamento do modelo LSTM.
Fonte: Elaboração própria.

O modelo atingiu uma acurácia de 99,01% com dados de treino e 99,57% com dados de teste. Na figura 29 são apresentados os melhores hiperparâmetros utilizados pelo modelo. As métricas de avaliação do modelo são apresentadas na sequência.

```
Acurácia 99.01% com dados de treino, usando parâmetros:

{'model__activation': 'softmax',
 'model__dense_units': 16,
 'model__lstm_units': 100,
 'model__optimizer': 'rmsprop'}
```

Figura 29: Acurácia com dados de treino e parâmetros do modelo LSTM.
Fonte: Elaboração própria.

	precision	recall	f1-score	support
imbalance	1.0000	0.9873	0.9936	79
normal	0.9231	1.0000	0.9600	12
underhang	1.0000	1.0000	1.0000	144
accuracy			0.9957	235
macro avg	0.9744	0.9958	0.9845	235
weighted avg	0.9961	0.9957	0.9958	235

Figura 30: Métricas de avaliação do modelo LSTM.
Fonte: Elaboração própria.

6.2. CNN - Convolutional Neural Network

Redes Neurais Convolucionais (CNNs) são um tipo de modelo de aprendizado profundo amplamente utilizado em tarefas de visão computacional, e sua capacidade de extrair padrões locais e hierárquicos as tornam adequadas também para a classificação de dados de séries temporais (CIABURRO, Giuseppe; VENKATESWARAN, Balaji, 2017).

As CNNs operam aplicando filtros convolucionais aos dados de entrada para extrair features relevantes. Esses filtros deslizam pela entrada, realizando operações matemáticas para identificar padrões específicos. As camadas de pooling reduzem a dimensionalidade dos dados, selecionando as features mais importantes e tornando o modelo mais robusto a pequenas variações na entrada. As camadas densas, também chamadas de camadas totalmente conectadas, combinam as features extraídas para realizar a classificação final.

Os filtros convolucionais são ideais para identificar padrões locais no espectro de frequências gerado pela Transformada Rápida de Fourier. Isso permite que a CNN capture informações específicas sobre as frequências presentes nos dados dos sensores.

O modelo CNN foi criado com as seguintes camadas:

- Input: Define a camada de entrada do modelo, especificando a forma dos dados que serão recebidos
- Conv1D: Aplica filtros convolucionais 1D aos dados de entrada para extrair features.
- MaxPooling1D: Reduz a dimensionalidade dos dados, selecionando o valor máximo em cada janela de pooling.
- Flatten: Transforma a saída multidimensional da camada de pooling em um vetor unidimensional.
- Dense: Camada onde cada neurônio está conectado a todos neurônios da camada anterior Essa camada aprende relações complexas entre os dados.
- Dense: Outra camada densa para de saída do modelo, que produz as previsões finais.

```
%%time
from tensorflow.keras.layers import Input, Conv1D, MaxPooling1D, Flatten, Dense, Dropout
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Input
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.callbacks import EarlyStopping
import pprint

n_samples, timesteps, features = X_train.shape

def create_model(filters, kernel_size, pool_size, dense_units, activation_conv, activation_dense1, activation_dense2):
    model = Sequential()
    model.add(Input(shape=(timesteps, features)))
    model.add(Conv1D(filters=filters, kernel_size=kernel_size, activation=activation_conv))
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(Flatten())
    model.add(Dense(dense_units, activation=activation_dense1))
    model.add(Dense(3, activation=activation_dense2))
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

early_stopping = EarlyStopping(monitor='loss', patience=3, restore_best_weights=True)
model = KerasClassifier(model=create_model, epochs=30, batch_size=32, verbose=2, callbacks=[early_stopping])

param_grid = {
    'model__filters': [32, 64],
    'model__kernel_size': [3, 5],
    'model__pool_size': [2, 3],
    'model__dense_units': [64, 128],
    'model__activation_conv':['relu', 'softmax'],
    'model__activation_dense1':['relu', 'softmax'],
    'model__activation_dense2':['relu', 'softmax'],
}

grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3, return_train_score=True)

grid_result_CNN = grid.fit(X_train, y_train, class_weight=class_weights_dict)
CNN_model = grid_result_CNN.best_estimator_.model_

print(f"Acurácia {grid_result_CNN.best_score_ * 100:.2f}% com dados de treino, usando parâmetros:"
  f"\n\n{pprint.pformat(grid_result_CNN.best_params_)}")
```

Figura 31: Treinamento do modelo CNN.
Fonte: Elaboração própria.

O modelo atingiu uma acurácia de 99,01% com dados de treino e 99,57% com dados de teste. Na figura 32 foram apresentados os melhores hiperparâmetros utilizados pelo modelo. As métricas de avaliação do modelo são apresentadas na sequência.

```
Acurácia 99.01% com dados de treino, usando parâmetros:

{'model__activation_conv': 'softmax',
 'model__activation_dense1': 'relu',
 'model__activation_dense2': 'softmax',
 'model__dense_units': 64,
 'model__filters': 32,
 'model__kernel_size': 3,
 'model__pool_size': 2}
```

Figura 32: Acurácia com dados de treinamento e hiperparâmetros do modelo CNN.
Fonte: Elaboração própria.

	precision	recall	f1-score	support
imbalance	1.0000	1.0000	1.0000	79
normal	1.0000	0.9167	0.9565	12
underhang	0.9931	1.0000	0.9965	144
accuracy			0.9957	235
macro avg	0.9977	0.9722	0.9844	235
weighted avg	0.9958	0.9957	0.9957	235

Figura 33: Métricas de avaliação do modelo CNN.
Fonte: Elaboração própria.

6.3. GRU - Gated Recurrent Unit

Um Gated Recurrent Unit (GRU) é um tipo de rede neural recorrente (RNN) que é frequentemente usada para processar dados sequenciais, como texto ou séries temporais, buscando solucionar o problema de desaparecimento do gradiente, comum em RNNs tradicionais, sem adicionar tanta complexidade.

A GRU, assim como a LSTM, é uma arquitetura de RNN que incorpora "memória" para processar sequências de dados. Entretanto, a GRU simplifica o processo utilizando apenas duas portas: a porta de atualização e a porta de reset.

A rede GRU criada possui as seguintes camadas:

- Input: Camada de entrada que estabelece o formato dos dados para as próximas camadas.
- GRU: processa os dados aprendendo as dependências de longo prazo.
- Dense: Camada onde cada neurônio está conectado a todos os neurônios da camada anterior. Essa camada aprende relações complexas entre os dados.

Assim como nos modelos anteriores, para evitar o overfitting foi utilizado early stopping.

```
%%time

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import  Dense, Input, GRU
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.callbacks import EarlyStopping
import pprint

n_samples, timesteps, features = X_train.shape

# Definir a função para criar o modelo GRU
def create_gru_model(gru_units, dense_units, dense_activation, optimizer):
    model = Sequential()
    model.add(Input(shape=(timesteps, features)))
    model.add(GRU(units=gru_units))
    model.add(Dense(units=dense_units, activation=dense_activation))
    model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

early_stopping = EarlyStopping(monitor='loss', patience=3, restore_best_weights=True)
model = KerasClassifier(model=create_gru_model, verbose=1, epochs=20, callbacks=[early_stopping])

param_grid = {
    'model__gru_units': [128, 256],
    'model__dense_units': [4, 8],
    'model__dense_activation':['softmax', 'relu'],
    'model__optimizer': ['adam', 'rmsprop'],
}

# Criar o objeto GridSearchCV
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3, return_train_score=True)

# Ajustar o GridSearchCV aos dados de treinamento
grid_result_GRU = grid.fit(X_train, y_train,  class_weight=class_weights_dict, verbose=1)
GRU_model = grid_result_GRU.best_estimator_.model_

# Imprimir os melhores hiperparâmetros
print(f"Acurácia {grid_result_GRU.best_score_ * 100:.2f}% com dados de treino, usando parâmetros:"
  f"\n\n{pprint.pformat(grid_result_GRU.best_params_)}")
```

Figura 34: Treinamento do modelo GRU.
Fonte: Elaboração própria.

O modelo GRU obteve uma acurácia de 98,87% com dados de treinamento e 99,15% com dados de teste. Na figura 35 é apresentado os melhores hiperparâmetros utilizados pelo modelo. As métricas de avaliação do modelo são apresentadas na sequência.

```
Acurácia 98.87% com dados de treino, usando parâmetros:

{'model__dense_activation': 'softmax',
 'model__dense_units': 8,
 'model__gru_units': 256,
 'model__optimizer': 'adam'}
```

Figura 35: Acurácia com dados de treinamento e hiperparâmetros do modelo.
Fonte: Elaboração própria.

	precision	recall	f1-score	support
imbalance	0.9873	0.9873	0.9873	79
normal	1.0000	0.9167	0.9565	12
underhang	0.9931	1.0000	0.9965	144
accuracy			0.9915	235
macro avg	0.9935	0.9680	0.9801	235
weighted avg	0.9915	0.9915	0.9914	235

Figura 36: Métricas de avaliação do modelo GRU.
Fonte: Elaboração própria.

7. Interpretação dos Resultados

Para a interpretação dos modelos de classificação foram utilizadas as seguintes métricas:

- Acurácia: Proporção de previsões corretas em relação ao total de previsões.
- Precisão: Indica a proporção de verdadeiros positivos entre todas as amostras classificadas como positivas.
- Revocação: Indica a proporção de verdadeiros positivos entre todas as amostras que realmente são positivas.
- F1-score: É a média harmônica entre precisão e revocação, fornecendo um equilíbrio entre as duas métricas.

Na tabela 7 é possível comparar estas métricas para cada um dos modelos treinados.

LSTM				
Classes	Precisão	Revocação	F1-Score	Acurácia
Normal	92,31	100	96	99,57
Desbalanceado	100	98,73	99,36	
Rolamento inferior	100	100	100	

Média ponderada	99,61	99,57	99,58	
CNN				
Classes	Precisão	Revocação	F1-Score	Acurácia
Normal	100	91,67	95,65	99,57
Desbalanceado	100	100	100	
Rolamento inferior	99,31	100	99,65	
Média ponderada	99,58	99,57	99,57	
GRU				
Classes	Precisão	Revocação	F1-Score	Acurácia
Normal	100	91,67	95,65	99,15
Desbalanceado	98,73	98,73	98,73	
Rolamento inferior	99,31	100	99,65	
Média ponderada	99,15	99,15	99,14	

Tabela 8: Comparativo das métricas de avaliação.
Fonte: Elaboração própria.

Observou-se que todos os modelos apresentaram uma performance muito boa, com CNN e LSTM apresentando as melhores métricas, inclusive com a mesma acurácia.

Como as classes estavam desbalanceadas, a acurácia pode ser enganosa. Se o modelo classificar corretamente um grande número de classes majoritárias, ele poderá apresentar alta acurácia mesmo falhando em classificar as classes minoritárias.

Observando a média ponderada das métricas, o modelo LSTM teve o melhor desempenho, com uma métrica ligeiramente maior.

Nas figuras 37, 38 e 39, são apresentadas as matrizes de confusão dos modelos, exibindo um resumo do desempenho dos modelos de classificação, mostrando a relação entre as previsões do modelo e os valores reais.

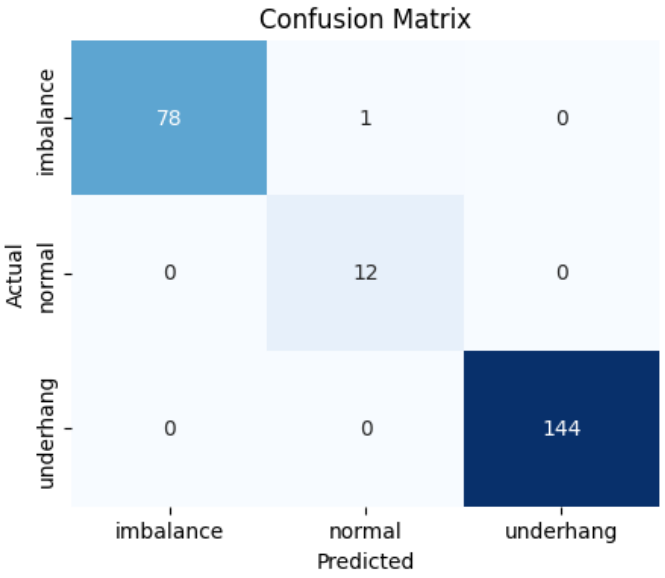


Figura 37: Matriz de confusão do modelo LSTM.
Fonte: Elaboração própria.

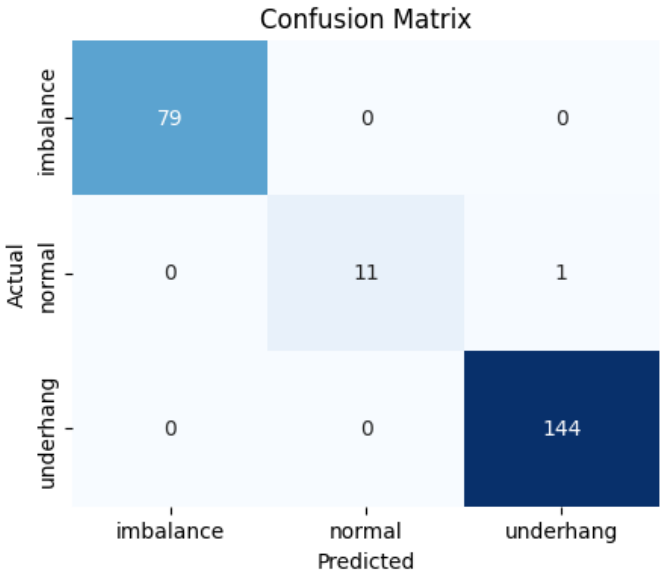


Figura 38: Matriz de confusão do modelo CNN.
Fonte: Elaboração própria.

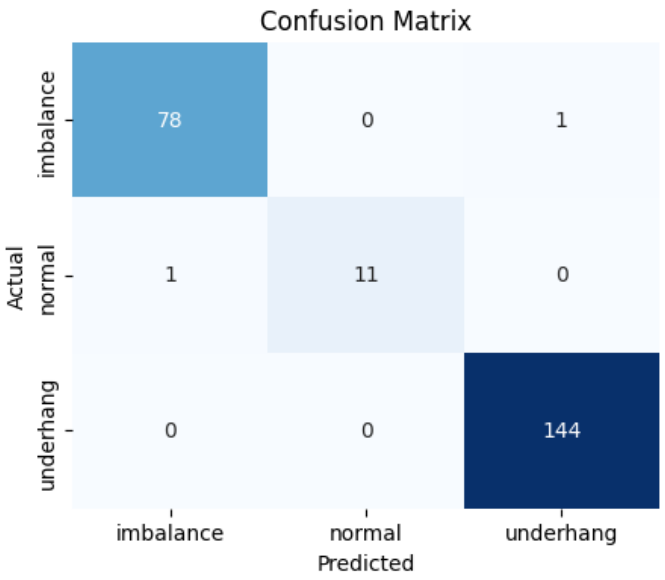
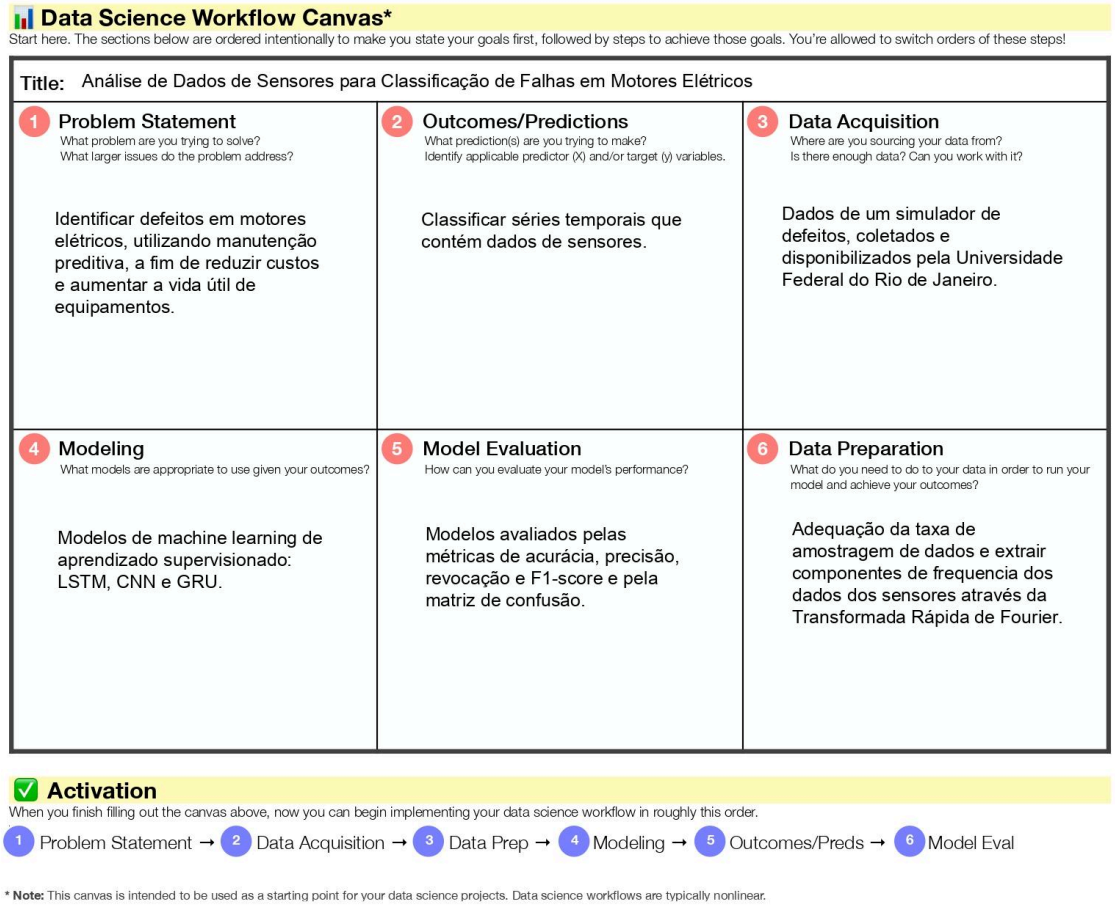


Figura 39: Matriz de confusão do modelo GRU.
Fonte: Elaboração própria.

Como o objetivo dessa classificação é a realização de manutenções preditivas, evitando parada de máquina, classificar um motor como normal quando ele apresenta algum defeito pode gerar maiores transtornos quando comparado ao cenário de classificar um motor como defeituoso mesmo o defeito não sendo real. Com isso, o modelo CNN é considerado o melhor dentre os 3 modelos treinados, por haver classificado incorretamente apenas uma série temporal como falha no rolamento, enquanto na realidade se tratava de uma série com funcionamento normal.

8. Apresentação dos Resultados

Para a execução desse trabalho, foi utilizado o modelo Canvas proposto por Vasandani (Vasandani, 2019), pois este auxiliou na identificação de cada etapa necessária para atender às especificidades de um projeto de machine learning.



Data Science Workflow Canvas*

Start here. The sections below are ordered intentionally to make you state your goals first, followed by steps to achieve those goals. You're allowed to switch orders of these steps!

Title: Análise de Dados de Sensores para Classificação de Falhas em Motores Elétricos

1 Problem Statement	2 Outcomes/Predictions	3 Data Acquisition
What problem are you trying to solve? What larger issues do the problem address?	What prediction(s) are you trying to make? Identify applicable predictor (X) and/or target (y) variables.	Where are you sourcing your data from? Is there enough data? Can you work with it?
Identificar defeitos em motores elétricos, utilizando manutenção preditiva, a fim de reduzir custos e aumentar a vida útil de equipamentos.	Classificar séries temporais que contém dados de sensores.	Dados de um simulador de defeitos, coletados e disponibilizados pela Universidade Federal do Rio de Janeiro.
4 Modeling	**5 Model Evaluation**	**6 Data Preparation**
What models are appropriate to use given your outcomes?	How can you evaluate your model's performance?	What do you need to do to your data in order to run your model and achieve your outcomes?
Modelos de machine learning de aprendizado supervisionado: LSTM, CNN e GRU.	Modelos avaliados pelas métricas de acurácia, precisão, revocação e F1-score e pela matriz de confusão.	Adequação da taxa de amostragem de dados e extrair componentes de frequencia dos dados dos sensores através da Transformada Rápida de Fourier.

Conceptualized by Jasmine Vasandani using notes from General Assembly's Data Science Immersive. Format inspired by Business Model Canvas.

Activation

When you finish filling out the canvas above, now you can begin implementing your data science workflow in roughly this order.

1 Problem Statement → 2 Data Acquisition → 3 Data Prep → 4 Modeling → 5 Outcomes/Preds → 6 Model Eval

* **Note:** This canvas is intended to be used as a starting point for your data science projects. Data science workflows are typically nonlinear.

Figura 40: Data science workflow Canvas.
Fonte: Elaboração própria.

9. Links

- Link para repositório do github: https://github.com/ricardo-faria/data-science-tcc
- Link para o vídeo: https://youtu.be/qx-t6a1pjZU

REFERÊNCIAS

1. SACOMANO, J. B. et al. Indústria 4.0: Conceitos e Fundamentos. São Paulo: Blücher, 2018
2. Tecnicon Sistemas Gerenciais. 4 exemplos práticos da adoção da Indústria 4.0 nas fábricas. 2022. Disponível em: <https://www.tecnicon.com.br/blog/476-4_exemplos_praticos_da_adocao_da_Industria_4_0_nas_fabricas>. Acesso em: 10 jul. 2024.
3. McKinsey & Company. Manufacturing: Analytics unleashes productivity and profitability. 2017. Disponível em: <https://www.mckinsey.com/capabilities/operations/our-insights/manufacturing-analytics-unleashes-productivity-and-profitability>. Acesso em: 10 jul. 2024.
4. Spectra Quest, Inc. Machinery Fault Simulator. 2015. Disponível em: <https://spectraquest.com/docview/?doc=brochure/MFS%20web%2016o.pdf>. Acesso em: 12/07/2024
5. Das, O., Bagci Das, D. Smart machine fault diagnostics based on fault specified discrete wavelet transform. J Braz. Soc. Mech. Sci. Eng. 45, 55 (2023). Disponivel em: <https://doi.org/10.1007/s40430-022-03975-0>. Acesso em: 20 jul. 2024
6. COPPE UFRJ. MaFaulDa - Machinery Fault Database. 2021. Disponível em: <https://www02.smt.ufrj.br/~offshore/mfs/index.html>. Acesso em: 15 jul. 2024
7. DA SILVA, F. Análise de Correlação em Estatística. 2023. Disponível em: <https://analisemacro.com.br/econometria-e-machine-learning/analise-de-correlacao-em-estatistica/>. Acesso em: 5 out. 2024.
8. ARAUJO, R. Seleção de Recursos em Ciência de Dados – (Feature Selection). 2023. Disponível em: <https://www.hashtagtreinamentos.com/selecao-de-recursos-em-ciencia-de-dados>. Acesso em: 5 out. 2024.
9. NUNES, T. Amostragem, teorema de Nyquist e aliasing - Ferramentas importantes durante processamento de sinal neural por EEG. 2023. Disponível em: <https://brainlatam.com/blog/amostragem-teorema-de-nyquist-e-aliasing-ferramentas-importantes-durante-processamento-de-sinal-neural-por-eeg-4233>. Acesso em: 5 out. 2024.

10. HAYKIN, S.; VAN VEEN, B. Sinais E Sistemas. Porto Alegre: Bookman, 2001.

11. CIABURRO, Giuseppe; VENKATESWARAN, Balaji. Neural Networks with R: Smart models using CNN, RNN, deep learning, and artificial intelligence principles. Birmingham: Packt Publishing, 2017

12. Malhotra, Pankaj et al. Long Short Term Memory Networks for Anomaly Detection in Time Series. European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning — ESANN 2015. Disponível em: <https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2015-56.pdf>. Acesso em: 5 out. 2024.
13. The Model class - Keras. 2024. Disponível em <https://keras.io/api/models/model/>. Acesso em: 13 out. 2024.
14. Vasandani , Jasmine. A Data Science Workflow Canvas to Kickstart Your Projects. Disponível em: <https://towardsdatascience.com/a-data-science-workflow-canvas-to-kickstart-your-projects-db62556be4d0>. Acesso em 15 out. 2024.