PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Davi Oliveira Francisco

Classificação de falhas em máquinas rotativas baseada em medições de vibração mecânica e métodos baseados em instância.

Projeto de Graduação

Projeto de Graduação apresentado ao Departamento de Engenharia Mecânica da PUC-Rio

Orientador: Helon Vicente Hultman Ayala, PhD
Coorientador: Mateus Gheorghe de C. Ribeiro, Msc

Rio de Janeiro

Novembro de 2021

Agradecimentos

Agradeço aos meus pais, Aloisio e Maria Daniella, pelo amor, apoio e confiança em toda minha jornada até este presente momento.

Agradeço à toda minha família, pelo apoio ao longo de toda minha formação, como profissional e ser humano.

Agradeço ao meu orientador e coorientador, Helon Vicente e Mateus Gheorghe, pelo apoio e confiança no desenvolvimento deste trabalho.

Agradeço aos inúmeros amigos da equipe Reptiles Baja, pelos bons momentos que proporcionaram ao longo do curso de engenharia mecânica. Amigos que levarei para toda minha vida. Tudo nosso!

Agradeço a Ana Luísa, minha companheira de todos os momentos, pelo amor e apoio incondicional em minha trajetória de formação e vida.

Resumo

Classificação de falhas em máquinas rotativas baseada em medições de vibração mecânica e métodos baseados em instância.

Este trabalho tem como objetivo desenvolver uma metodologia para classificação de máquinas rotativas com base em modelos baseados em instância. O desenvolvimento baseou-se em dados coletados utilizando uma bancada de testes que simulou quatro condições de operação. Primeiramente, foram escolhidos dois modelos, regressão linear e o *K-Nearest Neighbours* (KNN), para realizar a aplicação ao estudo de caso, onde a regressão linear foi escolhida para avaliar a dificuldade do problema. Além disso, o método de extração de características utilizado foi a análise de componentes principais (PCA). Após a escolha dos modelos foi utilizado um método de otimização de hiperparâmetros, com validação cruzada (*GridSearchCV*). Além disso, considerou-se diferentes porcentagens de variância explicadas para a escolha do número de componentes principais. Em um segundo momento, através dos resultados de otimização, realizou-se a comparação do modelo linear e KNN com os resultados da literatura. Por fim, obteve-se um novo modelo KNN otimizado e, utilizando métricas como, matriz de confusão e acurácia, seu desempenho resultante foi competitivo, considerando a eficiência e baixa complexidade. Os resultados demonstram que o KNN foi capaz de melhorar a acurácia em 68% quando comparado ao modelo linear.

Palavras chaves: Máquinas rotativas. Aprendizado de máquina. Regressão logística. KNN. *Principal Component Analysis*. Otimização de hiperparâmetros. Validação cruzada.

Abstract

Fault classification in rotating machines based on mechanical vibration measurements and instance-based methods.

This work aims to develop a methodology for rotating machinery classification based on instance-based models. The development was based on data collected using a test bench that simulated four operating conditions. First, two models, linear regression and K-Nearest Neighbours (KNN) were chosen to perform the application to the case study, where linear regression was chosen to evaluate the difficulty of the problem. In addition, the feature extraction method used was principal component analysis (PCA). After the models were chosen, a hyperparameter optimization method with cross-validation (GridSearchCV) was used. Besides this, different percentages of variance explained were considered for the choice of the number of principal components. In a second step, a comparison of the linear and KNN model with literature results was performed through the optimization results. Finally, a new optimized KNN model was obtained, and, using metrics such as confusion matrix and accuracy, its resulting performance was competitive, considering efficiency and low complexity. The results show that KNN improved accuracy by 68% compared to the linear model.

Keywords: Rotary machines. Machine Learning. Logistic Regression. KNN. Principal Component Analysis. Hyperparameter optimization. Cross-validation.

SUMÁRIO

Lista de figuras

Lista de tabelas

1 Introdução

Máquinas rotativas são equipamentos amplamente utilizados na indústria. Durante o tempo de operação as máquinas rotativas estão sujeitas a falhas que seguem uma lei conhecida como "*Baththub curve*" (LIU, XIAO, *et al.*, 2018). No início da operação a máquina pode falhar por motivos de erros de projeto ou de fabricação. A correção desses problemas leva a diminuição do número de falhas e no final do tempo de operação, com o desgaste sofrido pelo equipamento, os problemas tornam-se recorrentes.

Com o crescimento da indústria e da concorrência as empresas precisam encontrar meios de se manter competitivas. Uma solução possível é diminuir os gastos com manutenção, e ao mesmo tempo aumentar a confiabilidade, segurança e eficiência dos meios de produção da empresa. A implementação de um plano de manutenção e diagnósticos de falha são métodos eficientes para reduzir a ocorrência de paradas inesperadas para manutenção de equipamentos. O diagnóstico de falha é dividido em três principais etapas: determinar se o equipamento está funcionando normalmente, encontrar a falha incipiente e sua razão e prever a tendência de desenvolvimento da falha. Dessa forma, a determinação do estado de operação do equipamento é extremamente importante.

A inteligência artificial é um campo de estudo relativamente novo, que vem ganhando força nos últimos anos, e tem desenvolvido sistemas capazes de receber novas informações e tomar as melhores decisões considerando o cenário. Dessa forma, pode-se aplicar a inteligência artificial, já que se trata de uma ferramenta que possui grande capacidade para reconhecer padrões (LIU, YANG, *et al.*, 2018), ao problema de classificação, reduzindo os erros humanos.

1.1. Revisão de Literatura

O processo de classificação de falhas utilizando inteligência artificial envolve duas etapas principais: processamento dos dados (normalização, remoção de ruídos, extração de características etc.) e classificação das falhas (LIU, YANG, *et al.*, 2018). A etapa de extração de características, é uma etapa fundamental para a classificação. Ela tem como principal objetivo tratar os dados iniciais sem

perder informações importantes. Além disso, pode ser utilizada para facilitar a visualização e comparação das amostras.

A etapa de extração de características e classificação pode ser realizada de diversas maneiras. Liu, Zhihuai, et al. (2018) propõem a decomposição e tratamento do sinal de vibração e utiliza um método de descrição do sinal que gera uma sequência de números associando cada caso os aos tipos de falhas estudados. Zhao, Patel e Zuo (2011) propõem o estudo de vários sinais utilizando decomposição em modos empíricos (EMD) e a extração das funções de modo intrínsecos. Já Loutridis (2007) avalia o sinal de vibração utilizando técnicas estatísticas localmente tendo em vista que as falhas podem não aparecer ao analisar o sinal completo. Para equipamentos de grande porte, com um nível elevado de ruído, Liu, He, et al. (2016) sugerem a utilização de redução de ruído de segunda geração (SGWD) em conjunto com a decomposição média local (LMD). Os casos que obtiveram mais sucesso nos últimos anos foram propostos através de métodos do estado da arte, que utilizam aprendizado supervisionado. Em um estudo recente, Xue, Dou e Yang (2020) propõem, no campo do aprendizado profundo, a utilização de redes neurais convolucionais (DCNN) em conjunto com máquinas de vetores de suporte (SVM) para o problema de falhas múltiplas em máquinas rotativas. Para reduzir a dependência dos dados tratados e previamente classificados manualmente, Wu, Zhang, et al. (2021) utilizaram um método de classificação *autoencoder* híbrido.

1.2. Objetivos

Tendo em vista a importância da etapa de extração de características, o presente trabalho tem como objetivo desenvolver um modelo de aprendizado de máquina supervisionado, capaz de realizar a classificação automática da condição de operação de um rotor baseado apenas no sinal de vibração. Para isso, será utilizado um método de redução de dimensionalidade. Além disso, será realizada a otimização do algoritmo utilizando uma base de dados de vibração em 4 condições diferentes (LIU, XIAO, *et al.*, 2018).

Por fim, a performance do modelo será avaliada, utilizando diversos indicadores e métricas. Com isso, será estudada a capacidade de classificação do modelo para auxiliar na avaliação e diagnóstico de falha.

Em um primeiro momento, a fundamentação teórica do trabalho será descrita. Portanto, serão apresentados os conceitos de aprendizado de máquina e otimização de hiperparâmetros. Posteriormente, será discutido o estudo de caso nesse trabalho. Em seguida, será apresentada a metodologia utilizada para obter o modelo final otimizado. Por fim, será feita a discussão dos resultados seguida pela conclusão e sugestões de trabalhos futuros.

2 Fundamentação teórica

2.1. Máquinas Rotativas

Máquinas rotativas se caracterizam por possuírem partes móveis que são fundamentais para o seu funcionamento. Uma das aplicações mais comuns é a conversão de energia, seja de elétrica para mecânica com os motores, ou ainda, no sentido oposto como, por exemplo, com os geradores. Em um motor elétrico de corrente contínua o rotor é a parte móvel fundamental. A interação entre campos magnéticos e a corrente elétrica aplicada nos fios do rotor, faz com que um torque seja aplicado ao eixo do rotor. Por fim, o eixo de saída do motor é acoplado a outros equipamentos para transmitir a energia gerada (KIM, 2017).

2.2. Aprendizado de máquinas

Nos últimos anos a análise de dados se mostrou capaz de gerar informações valiosas para empresas. As análises estão se tornando mais complexas e precisas e por isso, mais empresas vem adotando esse estudo. O crescente valor econômico desse estudo é justificado pela capacidade de utilizar informações já disponíveis para otimizar a tomada de decisões futuras das empresas. Além de contribuir para o crescimento da margem de lucro, diminuir a possibilidade de acidentes e melhorar a performance dos equipamentos (LEI, HE e ZI, 2008).

O aprendizado de máquina envolve a criação de um modelo computacional capaz de detectar padrões, aprender com os próprios erros e fazer previsões baseadas em dados (LEMLEY, BAZRAFKAN e CORCORAN, 2017). Os dados utilizados em estudos que envolvem aprendizado de máquina geralmente apresentam a estrutura e nomenclatura dispostos na Figura 1.

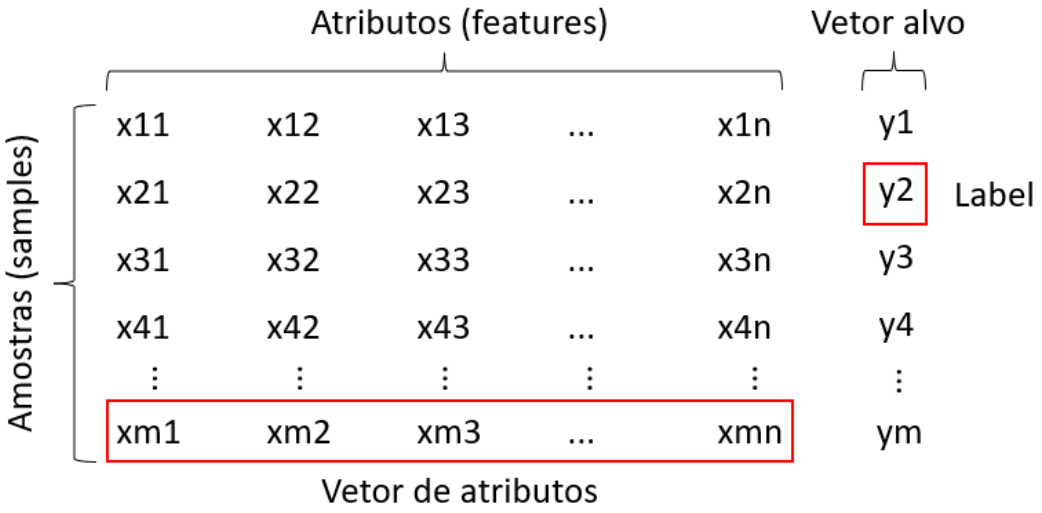

Figura 1 - Dados divididos em amostras com vetores de atributos associados a uma *Label*.

Existem diversos tipos de modelos de aprendizado de máquina e o mais apropriado para cada situação depende da natureza dos dados. Duas das principais abordagens de aprendizado de máquina são o aprendizado supervisionado e não-supervisionado.

O aprendizado não-supervisionado trata dos algoritmos que não recebem os dados com os rótulos (*Labels*) para o treinamento. Por isso, devem ter a capacidade de reconhecer os padrões apenas com os dados de entrada. Um exemplo comum de aprendizado não-supervisionado é o *clustering* (LEMLEY, BAZRAFKAN e CORCORAN, 2017). Nesse caso, o algoritmo recebe as informações e tenta agrupar os dados de acordo com as semelhanças encontradas entre os dados.

Neste trabalho os algoritmos em questão são baseados em aprendizado supervisionado. Neste caso, o algoritmo recebe os dados com os rótulos disponíveis. Baseado nessas informações desenvolve-se um modelo com a capacidade de realizar previsões utilizando dados não fornecidos previamente (RUSTAM, AHMAD RESHI, *et al.*, 2020). Os algoritmos de aprendizado supervisionado podem ser divididos em duas abordagens, classificação e regressão.

A Figura 2 ilustra um caso de algoritmo de aprendizado supervisionado, utilizado para dividir os dados em 4 classes.

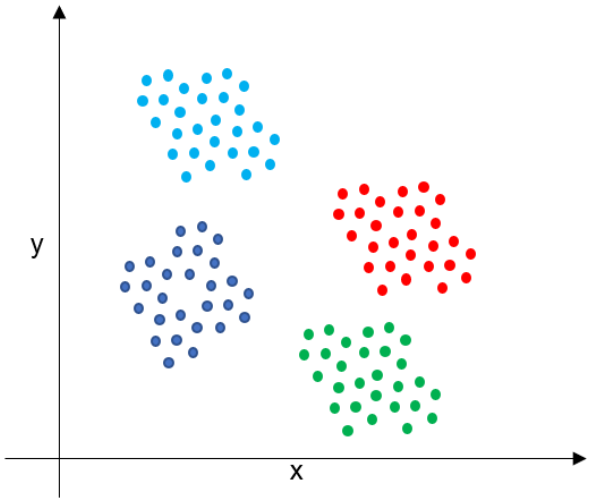

Figura 2 - Exemplo de algoritmo de aprendizado supervisionado que realizou a classificação dos dados fornecidos em 4 classes.

A Figura 3 ilustra a relação entre os valores reais e previstos por um modelo linear. O exemplo, se trata de um caso em que dados de treino foram fornecidos ao algoritmo e, baseado nessas informações, que possuem um atributo relacionado a um rótulo (valor real), foi a ajustada uma reta que melhor descreve essa relação. Utilizando esse modelo é possível realizar a previsão aproximada do rótulo de novos atributos.

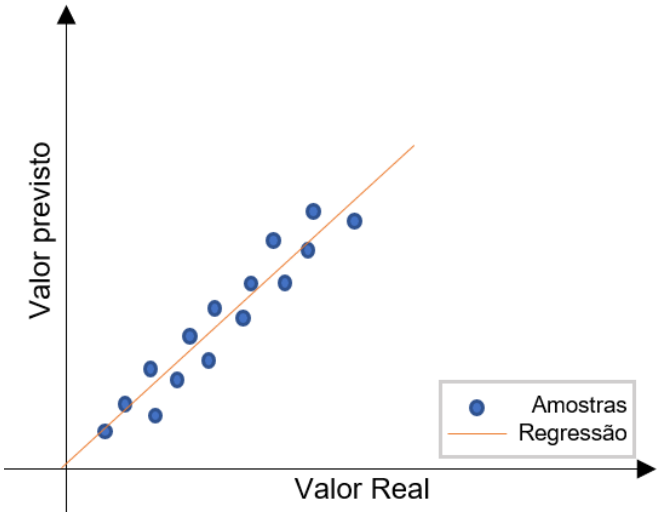

Figura 3 – Ilustração de um modelo de regressão linear ajustado aos dados de treino.

Ao criar um modelo para a utilização no aprendizado supervisionado um dos grandes problemas que deve ser considerado é o “*overfitting vs underfitting*”. A

utilização de um modelo complexo que descreve muito bem os dados de treino pode levar ao “*overfitting*”. Nessa situação o modelo é muito bem ajustado apenas para os dados de treino e não é capaz de generalizar para os dados de teste. Por outro lado, ao utilizar um modelo menos complexo, que não descreve bem os dados de treino, este pode não ser capaz de “aprender” a relação entre os dados de treino e suas respectivas saídas, ou seja, pode passar por “*underfitting*”. Sem ter sido propriamente treinado o modelo apresenta um erro elevado no treinamento e nos dados de teste (LEMLEY, BAZRAFKAN e CORCORAN, 2017).

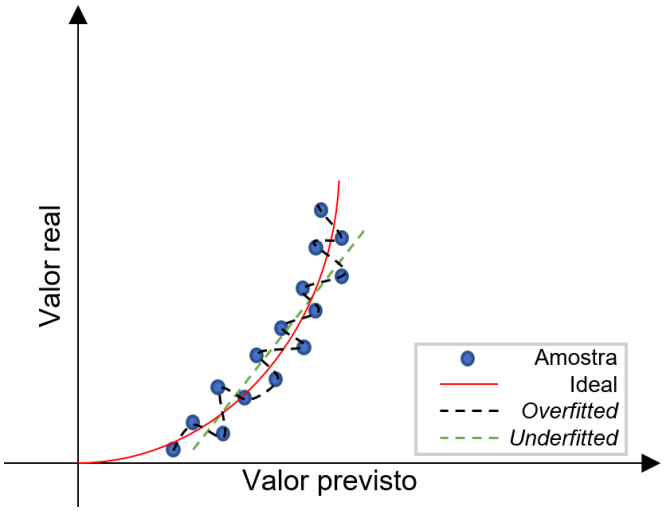

Figura 4 - Comparação *Overfitting* / *Underfitting* (LEMLEY, BAZRAFKAN e CORCORAN, 2017).

Neste trabalho, dois modelos de aprendizado de máquina, são considerados: regressão logística e *K-Nearest Neighbors* (KNN).

2.2.1. Regressão Logística

Este primeiro modelo, também conhecido como “Regressão logit” ou classificador “log-linear”, é um classificador linear utilizado em casos simples. Essa técnica tem como principal objetivo fornecer um modelo capaz de prever valores de uma variável dependente, que geralmente é binária, com base outras variáveis independentes. Com base nesse modelo, são obtidas as probabilidades de ocorrência de um evento, dadas as variáveis aleatórias (observações). Essa técnica é baseada no logaritmo natural da função “logit”

(Equação 1), que liga a combinação linear das variáveis a distribuição Bernoulli, da variável dependente.

$$Logit = ln\left(\frac{p}{1-p}\right) \quad (1)$$

A função inversa do “logit” (Figura 5), que assume a forma sigmóide (formato de “S”), recebe a combinação linear das variáveis independentes (Equação 2), t, e utiliza-a para calcular a probabilidade da variável dependente ser igual a 1, que é o evento de interesse.

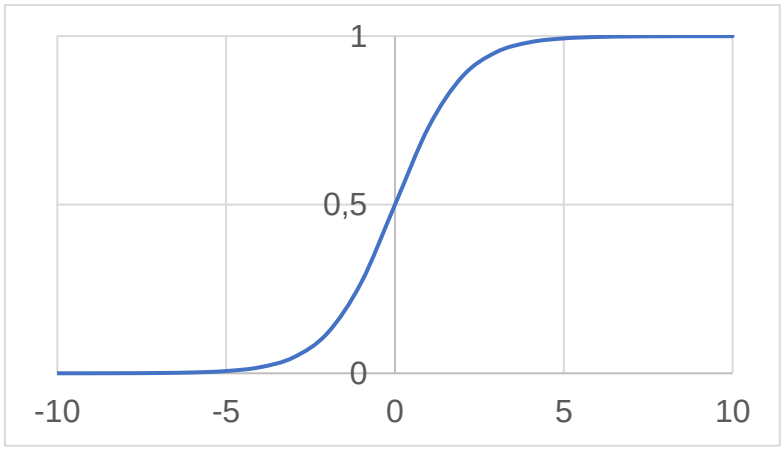

Figura 5 - Esboço do gráfico da inversa da função “logit”.

A combinação linear das variáveis independentes é representada por:

$$Combinação = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n \quad (2)$$

Onde β_n são os coeficientes e x_n são as variáveis independentes.

Considerando que o problema deste trabalho trata de mais de duas classes o modelo de regressão logística pode ser expandido e tratado em uma abordagem de um contra todos (SEIDEL, LEIBOLD, *et al.*, 2020). Nessa abordagem um novo modelo é treinado, para cada classe em questão, e prevê se o novo dado pertence ou não a sua classe. A classificação realizada pelos modelos é feita de maneira independente e ao final são unidas para definir a classe.

Os principais parâmetros considerados no modelo de regressão logística são (KSIAZEK, GANDOR e PAWEL, 2021):

“Penalty”: Utilizado para evitar o “*overfitting”*. Pode ser “L1”, “L2”, “Elasticnet” ou nenhum.

“Solver”: O algoritmo utilizado para encontrar os parâmetros de minimizam o erro do modelo. Por exemplo, “saga”, “lbfgs”, “liblinear”, “sag” e “newton cg”, entre outros.

“C”: É um parâmetro que também se relaciona com o problema de “*overfitting”* e o “*underfitting”*. Grandes valores de “C” determinam que o modelo descreva muito bem os dados de treino, dando menos importância a penalidade de complexidade. Já para valores pequenos de “C” o modelo leva mais em conta a penalidade por complexidade, deixando assim de descrever com excelência os dados de treino.

2.2.2. KNN

O KNN é um algoritmo utilizado em problemas mais complexos com um maior volume de dados, para classificação e regressão de dados (G. S. K., KUMAR VERMA e RADHIKA, 2019). No caso da classificação o algoritmo retorna a classe à qual a amostra pertence baseado nas classes dos dados mais próximos. Já no caso da regressão o objetivo do algoritmo é encontrar uma aproximação para o valor em questão baseado nas saídas das amostras com maior semelhança.

Neste trabalho o algoritmo é utilizado em uma abordagem de classificação. Primeiramente, os dados de treino são armazenados e qualquer nova informação, a ser classificada, inserida no modelo é posicionada no espaço n-dimensional criado pelos dados de treino. Para determinar sua classe é calculada a distância entre do novo ponto até os dados fornecidos no treinamento. A distância em questão pode ser calculada de diversas maneiras como: Euclidiana (Equação 3), Manhattan (Equação 4) e Minkowski (Equação 5).

Considerando um problema bidimensional a distância Euclidiana é dada por:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2} \quad (3)$$

Onde (p_1, p_2) e (q_1, q_2) são as coordenadas nos eixos de cada ponto e d é a distância calculada. Já a distância de Manhattan é definida como:

$$d(p, q) = (p_1 - q_1) - (p_2 - q_2) \quad (4)$$

Por fim, a distância de Minkowski, possui um parâmetro variável "j" e é definida como:

$$d(p, q) = \sum_{i=1}^{2} |p_i - q_i|^j \quad (5)$$

A amostra será classificada de acordo com a classe predominante entre os k elementos mais próximos, onde k é um valor a ser definido ao realizar o treinamento do modelo, e variações desse dado geram alterações de classificação. Outra característica importante desse algoritmo é a possibilidade de atribuir pesos as classes dos elementos próximos. Um exemplo é a utilização de pesos inversamente proporcionais à distância da nova amostra, que faz com que elementos mais próximos sejam mais influentes na classificação. Dessa forma, para o KNN os parâmetros em questão são:

K: número de vizinhos considerados na classificação do ponto em questão.

P: Determina o tipo de distância a ser calculada, como por exemplo a euclidiana.

"*Weight*" ou peso: Define a influência das classes dos dados próximos na classificação final do novo ponto. Se for "*uniform*" todos os dados têm a mesma influência. No caso "*distance*" a influência é inversamente proporcional à distância logo, pontos mais próximos são mais influentes.

Através da Figura 6 é possível observar um exemplo básico de classificação utilizando o KNN. Nesse caso, o ponto vermelho foi classificado como quadrado azul porque foi definido k = 4, antes de treinar o modelo. Dos quatro pontos mais próximos, considerados na avaliação da classe do novo ponto, três são quadrados azuis enquanto apenas um é um triângulo verde.

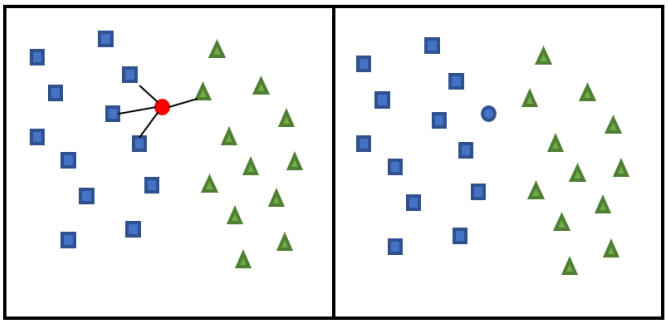

Figura 6 – Exemplo de classificação onde k é igual a 4, logo o ponto vermelho foi definido como azul (G. S. K., KUMAR VERMA e RADHIKA, 2019).

2.3. Análise de componentes principais (PCA)

Em problemas de aprendizado de máquina são comumente encontradas amostras com uma quantidade muito grande de atributos. Muitas vezes o grande volume de *features* é desnecessário e pode tornar a criação do modelo um processo muito caro computacionalmente e, por isso, seria melhor usar menos *features* sem perder muitas informações desses dados. Uma das técnicas mais conhecidas e aplicadas em problemas, para a redução da dimensionalidade é conhecida como *principal component analysis* (PCA).

O PCA é um método que reduz a dimensionalidade preservando grande parte das informações importantes dos dados. O exemplo 2D abaixo serve para ilustrar o método (HOLAND, 2019). Inicialmente, considera-se os dados do gráfico abaixo sobre largura e comprimento.

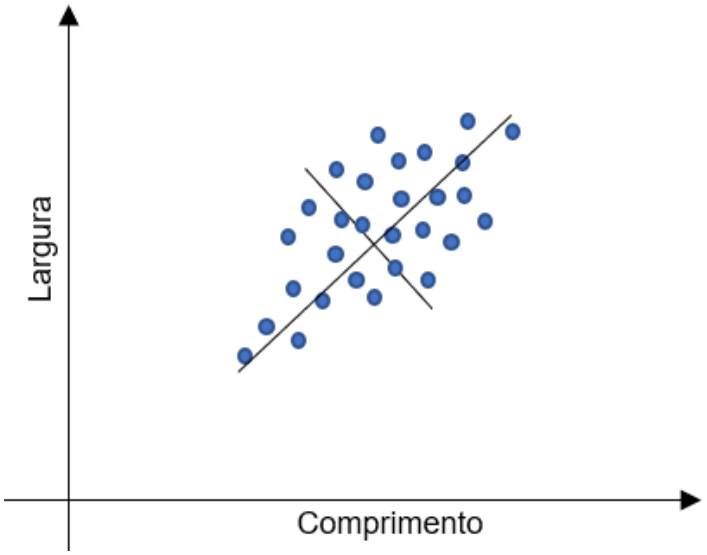

Figura 7 - Exemplo de conjunto de dados (HOLAND, 2019).

Com os dados acima, é possível encontrar o centro dos dados. Depois pode-se calcular o vetor que passa pelo centro e maximiza a soma das distancias até a origem dos dados projetados no vetor. Para determinar o segundo vetor, por se tratar do caso 2D, basta usar o vetor perpendicular ao primeiro e que passa pelo centro dos dados. Rotacionando os vetores encontrados obtemos o novo eixo de coordenadas (Figura 8).

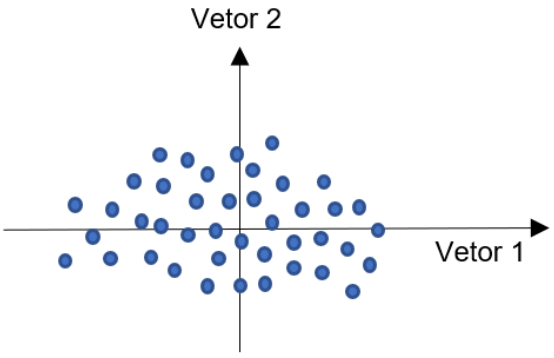

Figura 8 - Componentes principais (HOLAND, 2019).

Os eixos 1 e 2, Figura 8, são conhecidos como componentes principais e cada um representa um percentual da variância total dos dados.

Para casos com n-dimensões pode-se determinar as componentes principais e qual o percentual da variância representado por elas. Se for determinado, por exemplo, que apenas duas componentes representam um grande percentual da

variância explicada pode ser feito um gráfico considerando esses novos eixos para realizar a visualização dos dados.

Do ponto de vista matemático, esse procedimento inicia considerando os dados fornecidos em uma matriz m colunas e n linhas onde cada linha contém as medições de um experimento nas m colunas.

$$X = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{pmatrix} \quad (6)$$

Da matriz X será criada uma matriz, $\bar{X}$,com a média de cada linha (experimento).

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad (7)$$

$$\bar{X} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} * \bar{x}. \quad (8)$$

Posteriormente, essa matriz de médias é subtraída da original e é calculada a matriz de covariância da resultante.

$$B = X - \bar{X}, \quad (9)$$

$$C = B^T B. \quad (10)$$

Por fim, serão calculados os autovalores e autovetores da matriz de covariância e as componentes principais (T)

$$T = BV, \quad (11)$$

onde V é a matriz de autovetores. Vale ressaltar, que os autovalores calculados representam a variância dos dados originais explicada pelo autovetor (componente principal) correspondente (BRUNTON e KUTZ, 2017). Para saber, por exemplo, qual a variância explicada pelas primeiras r componentes

principais, basta calcular a razão entre a soma dos autovalores, associados as r componentes, e a soma de todos os n autovalores.

$$\%_{explicado} = \frac{\sum_{i=1}^{r} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \quad (12)$$

A capacidade de realizar análises em uma menor quantidade de atributos, considerando grande parte da informação inicial, é extremamente interessante e justifica a aplicação desse método em diversos trabalhos.

2.4. Métricas e avaliação

A avaliação da performance de um modelo é utilizada para fazer observações considerando a variação de parâmetros envolvidos no modelo e dos dados de treino e teste. Essa avaliação pode ser feita utilizando alguns das métricas mais comuns, como: matriz de confusão, acurácia, precisão, *recall* e *f1-score*.

2.4.1. Matriz de confusão

A matriz de confusão é amplamente utilizada em aprendizado de máquina para problemas de classificação. É representada por uma matriz quadrada na qual as linhas são as classes verdadeiras em cada caso e as colunas representam as classificações realizadas pelo modelo (HASNAIN, PASHA, *et al.*, 2020). Por isso, a diagonal da matriz contém os acertos do modelo e o restante das informações são os erros. Um exemplo de matriz de confusão para uma classificação binária é representado na Figura 9.

		Valores preditos	
		Negativo	Positivo
Valores reais	Negativo	VN	FP
	Positivo	FN	VP

Figura 9 – Ilustração de uma matriz de confusão, na qual os valores da diagonal representam os acertos de um modelo.

Na matriz de confusão, Verdadeiro Negativo (VN) e Verdadeiro Positivo (VP) são os valores que foram corretamente classificados como negativos e positivos, respectivamente. Os termos Falso Positivo (FP) e Falso Negativo (FN) representam os casos em que o modelo estimou a classe errada da amostra. Para casos de classificação com k classes, a matriz será sempre quadrada da forma k x k.

2.4.2. Acurácia

A acurácia (Acc) é um indicador simples que representa o número de previsões corretas do modelo. Esta métrica pode ser representada matematicamente por

$$Acc = \frac{VP + VN}{VP + FP + VN + FN}. \quad (13)$$

2.4.3. Precisão

A precisão, também conhecida como Valor Preditivo Positivo (VPP), de um modelo é definida como a proporção de dados classificados como positivos que realmente são positivos. Esta métrica pode ser representada por

$$VPP = \frac{VP}{VP + FP}. \quad (14)$$

2.4.4. Recall

Recall (Rec) é definido como a proporção dos dados positivos que foram classificados como positivos. A métrica mencionada pode ser representada matematicamente por

$$Rec = \frac{VP}{VP + FN}. \quad (15)$$

2.4.5. F1-score

Já *f1-score* é a média harmônica entre os dois indicadores anteriores, precisão e *recall*, que varia de 0 a 1. Quanto mais próximo de 1 melhor é a performance do modelo. Esta métrica é matematicamente definida como

$$F1 = 2 * \frac{1}{\frac{1}{VPP} + \frac{1}{Rec}}. \quad (16)$$

2.5. Calibração de hiperparâmetros

Cada algoritmo possui parâmetros que devem ser definidos antes de realizar a criação do modelo, essas informações são conhecidas como hiperparâmetros. Dependendo da combinação de hiperparâmetros utilizada a performance do modelo muda, por isso, é interessante ter a combinação de parâmetros que otimizam a performance. Para isso existem duas principais maneiras de testar diferentes combinações de parâmetros, e avaliar a performance utilizando validação cruzada (CV), os métodos *Grid Search CV* e *Random Search CV* (G. S. K., KUMAR VERMA e RADHIKA, 2019).

A validação cruzada é uma etapa fundamental dos métodos que avalia o desempenho em cada combinação de hiperparâmetros. Nessa etapa os dados disponíveis são divididos em n grupos e o modelo é treinado com n-1 grupos e testado com 1 grupo. Esse procedimento é repetido até que todos os dados tenham sido testados.

A primeira técnica de otimização, conhecida como *Grid Search CV*, propõe testar todas as combinações de um conjunto de hiperparâmetros selecionados. A combinações de todos os hiperparâmetro formam um espaço n-dimensional, e esse espaço, no método *Grid Search CV*, é totalmente testado. Um dos grandes problemas dessa técnica está relacionada ao seu grande esforço computacional para testar todas as possibilidades.

Já na segunda técnica, conhecida como *Random Search CV*, uma quantidade específica de combinações aleatórias são testadas em um universo de hiperparâmetros definido. Dessa forma, o esforço computacional pode ser controlado pelo número de amostras aleatórias utilizadas pelo método para otimizar o modelo.

Na Figura 10, pode-se observar as etapas do processo de otimização. Na etapa “a” é feita a escolha dos parâmetros para definir o modelo. Em seguida, etapa “b” é realizada a validação cruzada para obter a melhor performance desse

modelo. Essas duas etapas são repetidas para todas as combinações de parâmetros consideradas (etapa “c”). Por fim, é escolhida a combinação de hiperparâmetros que obteve o melhor desempenho.

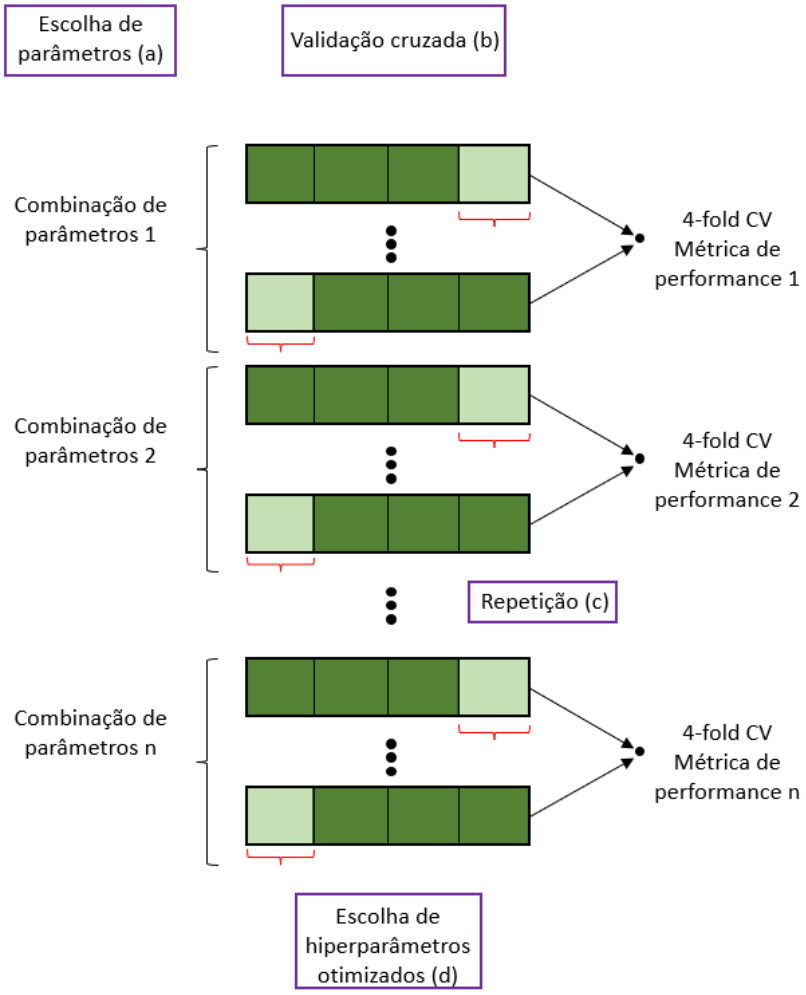

Figura 10 – Ilustração do funcionamento validação cruzada.

Um aspecto importante desses métodos de otimização é a escolha de um avaliador de performance. Para os casos de classificação um exemplo de parâmetro de performance utilizado é a acurácia, já nos casos de regressão um exemplo é o coeficiente de determinação.

3 Estudo de Caso

3.1. Obtenção dos dados

O desempenho de um modelo de aprendizado supervisionado depende em grande parte da qualidade dos dados utilizados no treinamento. Por isso, grande parte do esforço para o desenvolvimento de um modelo de aprendizado supervisionado está relacionado a obtenção e tratamento de dados (LIU, ZHIHUAI, *et al.*, 2018).

A fonte de informação em questão apresenta dados de vibração de um rotor coletada através de uma bancada de testes. A bancada em questão, Figura 11, foi desenvolvida para realizar testes sob diferentes condições de operação: normal, desbalanceado, desalinhado e com fricção.

A bancada é composta por um motor elétrico acoplado a um eixo, um controlador de velocidade, um processador de dados, um computador e três aparatos anexados ao eixo que servem para simular as condições de operação. O motor de corrente contínua (potência máxima de 148W) é acionado por um controlador de velocidade DH5600. O motor é conectado a dois eixos, unidos por um acoplador, que têm 10 mm de diâmetro, comprimento total de 850 mm e são suportados por dois mancais de rolamento. Além disso, foram adicionados dois discos com diâmetro de 75 mm.

O aparato que simula a condição de operação com fricção (1 na Figura 11), é um suporte responsável por sustentar um parafuso em contato com o eixo. Já para o desbalanceamento é realizado ao acoplar um bloco de 2 gramas, na borda do disco existente no eixo (2 na Figura 11), de maneira excêntrica causando vibrações acima do normal no eixo. Por fim, o desalinhamento é feito utilizando o acoplador para mudar a posição relativa entre as duas partes do eixo (3 na Figura 11).

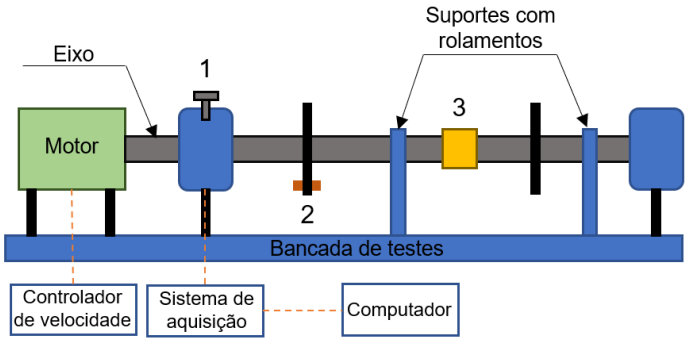

Figura 11 - Bancada de testes com os 3 aparatos anexos ao eixo do motor para simular as condições de operação. O aparato 1 representa o suporte para parafuso que simula a condição de atrito; o aparato 2 representa o suporte para a massa que emula o desbalanceamento do eixo; finalmente, o aparato 3 representa o acoplador para a simulação do desalinhamento.

Durante a aquisição dos sinais o motor de corrente contínua permanece em 1200 rotações por minuto e o sistema de aquisição de dados coleta o sinal de vibração a uma taxa de 2048 Hz durante um segundo para cada amostra. Os dados coletados são passados para o processador que amplifica, filtra e finalmente envia para o computador. Utilizando essa bancada foram obtidas 180 amostras de dados de vibração do rotor, 45 em cada condição de operação. É possível observar um exemplo uma amostra, aceleração (em g) por tempo (em segundos), coletada de cada condição de operação na Figura 12.

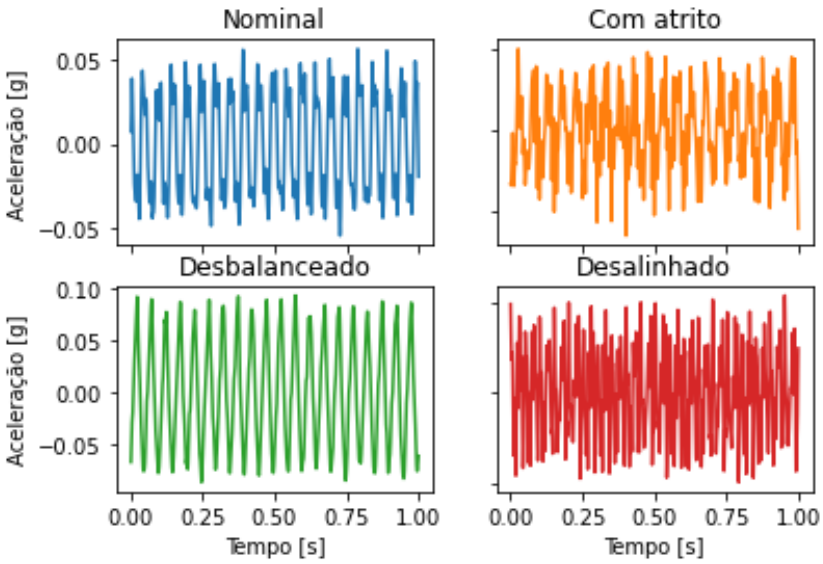

Figura 12 - Gráficos das amostras em diferentes condições de operação. É possível notar como é difícil diferenciar cada condição visualmente.

4 Metodologia

Para obter o modelo final otimizado foi desenvolvida uma metodologia, ilustrada na Figura 13, que é composta pelas etapas: obtenção dos dados, normalização e análise de componentes principais, otimização de hiperparâmetros e componentes principais, comparação de desempenho dos modelos e treinamento do modelo final otimizado.

Figura 13 - Ilustração dos passos utilizados na metodologia para obter o modelo otimizado.

Na primeira etapa, de obtenção de dados, as informações coletadas por Liu, Xiao, et al., (2018), utilizando a bancada de testes, foram extraídas do arquivo original e, utilizando a biblioteca Pandas, passadas para uma tabela. Como preparativo para as próximas etapas as condições de operação foram codificadas em classes de acordo com a Tabela 1.

Tabela 1 - Classes das condições de operação.

Condição de operação	Classe
Nominal	1
Com atrito	2
Desbalanceado	3
Desalinhado	0

Ao final da primeira etapa todas as informações obtidas foram dispostas na Tabela 2, onde cada linha representa uma sequência de 2048 medições realizadas dentro de um segundo para uma dada condição de operação. Dessa forma, a tabela obtida possui 180 linhas (45 de cada condição de operação) e 2049 colunas (2048 medidas e a identificação da condição de funcionamento).

Tabela 2 – Ilustração da tabela final com informações do problema.

Amostra	Medida 1	Medida 2	...	Medida 2048	Condição de operação
1	0,007	0,011	...	-0,019	1
2	-0,024	-0,026	...	-0,025	1
⋮	⋮	⋮	⋱	⋮	⋮
180	-0,000	-0,000	...	-0,027	0

Na etapa seguinte, é realizado o tratamento dos dados, utilizando a normalização, e a extração de características utilizando a análise de componentes principais. A normalização dos dados tem como objetivo facilitar a visualização, reduzir o impacto dos “outliers” no treinamento e aumentar a eficiência do modelo. Determinou-se para este trabalho que seria aplicada a normalização “MinMax” (LIU, LONG, *et al.*, 2021). Este procedimento de transforma os dados disponíveis passando os valores para o intervalo [0,1], através da seguinte fórmula.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}, \tag{17}$$

onde x' é o valor normalizado e x o original.

Com os dados normalizados, inicia-se a análise de componentes principais. Este procedimento tem como objetivo reduzir significativamente o volume de atributos (neste estudo os atributos são as medições) sem perder muitas informações. Um aspecto extremamente importante dessa etapa é a determinação de quantas componentes principais serão utilizadas para descrever os dados e, para determinar esse parâmetro, são utilizadas algumas quantidades diferentes de componentes. Através da comparação da performance do modelo para essas diferentes quantidades pode-se escolher a que otimiza o modelo. As quantidades avaliadas neste estudo foram: 71% da variância (3 componentes), 78% da variância (10 componentes), 90% (12 componentes), 95% (18 componentes) e 99% (30 componentes).

Na etapa de otimização dos hiperparâmetros busca-se obter a combinação desses parâmetros que otimizam o modelo, considerando que neste trabalho

são utilizados dois algoritmos, KNN e regressão logística, esse procedimento deve ser feito duas vezes. O método escolhido para otimizar o modelo foi o *GridSearchCV* já que o banco de dados utilizado é pequeno, por isso, mesmo com um número razoável de combinações de hiperparâmetros o custo computacional não é um problema.

Determinado o método o próximo passo é escolher quais parâmetros serão variados para obter o melhor desempenho, que será avaliado utilizando a acurácia como métrica para a validação cruzada. Considerando os parâmetros dos algoritmos expostos no capítulo de fundamentação teórica, foram escolhidos valores usuais para cada parâmetros, destacados na Tabela 3, para serem testados. Dessa forma, para o KNN foram consideradas 290 combinações, sendo os parâmetros: “K”, “P” e “Peso”, e para a regressão logística 96 combinações, sendo os parâmetros: “Solver”, “Penalidade” e “C”.

Tabela 3 - Intervalos de valores dos hiperparâmetros variados no processo de otimização.

Regressão Logística	
Hiperparâmetro	Intervalo de variação
Penalty	"None", "L1", "L2" e "Elasticnet"
Solver	"Newton-cg", "lbfgs" e "liblinear"
C	1E-5, 1E-4, 1E-3, 1E-2, 1E-1, 1, 10 e 100
KNN	
Hiperparâmetro	Intervalo de variação
K	1 a 29
P	1 a 5
Peso	"Uniform" ou "Distance"

Na etapa de validação cruzada do método de otimização *GridSearchCV* os dados foram divididos em 2 grupos. Dessa forma, o treinamento do modelo pode ser feito de maneira similar a proposta por Liu, Zhihuai, et al. (2018) com 80 amostras (20 de cada condição de operação) e 100 para o teste do modelo. Vale ressaltar, que a divisão dos grupos foi realizada de forma estratificada para respeitar a proporção entre as classes.

O processo de otimização de hiperparâmetros definido deve ser repetido para todas as quantidades de componentes principais determinadas, dessa forma, é possível analisar a performance para cada combinação.

Com esses resultados inicia-se a etapa de comparação dos modelos. Através da análise da performance dos modelos em cada combinação é possível, determinar qual configuração proporciona o melhor desempenho. Por fim, conhecendo a configuração otimizada é realizado o treinamento do modelo final otimizado.

5 Resultados

5.1. Análise exploratória dos dados

As amostras de sinal de vibração obtidas possuem comportamentos semelhantes como foi possível observar na Figura 12. Nesse sentido a classificação desses dados é muito complicada com apenas com as informações originais. Na Figura 14 pode-se observar uma representação do sinal de aceleração (g) por tempo (segundos) onde as informações se encontram sobrepostas dificultando a diferenciação das classes.

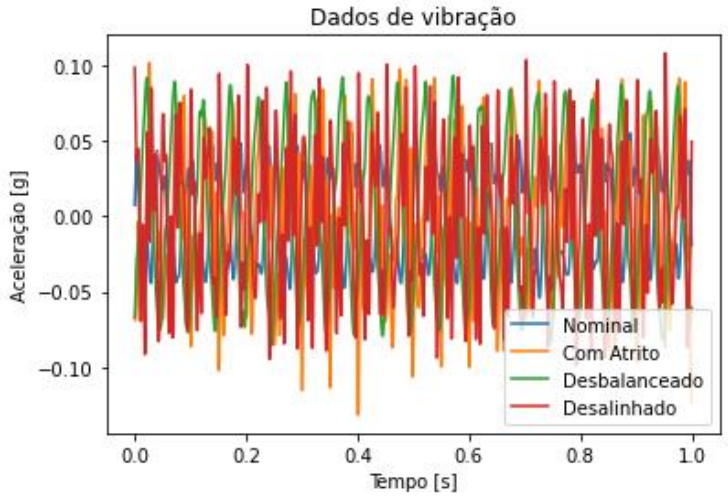

Figura 14 - Exemplo de sinal de aceleração por tempo em cada condição de operação obtido na bancada de testes. É praticamente impossível diferenciar os sinais e suas classes no sinal original.

Primeiramente, os dados são normalizados para levar os valores originais para o intervalo de [0;1]. Com os dados normalizados a análise de componentes principais é utilizada para reduzir o volume de dados, para diminuir o custo computacional do treinamento do modelo e extrair as informações contidas no sinal original para facilitar a classificação. É possível observar na Figura 15 que a três primeiras componentes principais representam a maior parte da variância dos dados originais enquanto as outras componentes não representam grandes percentuais.

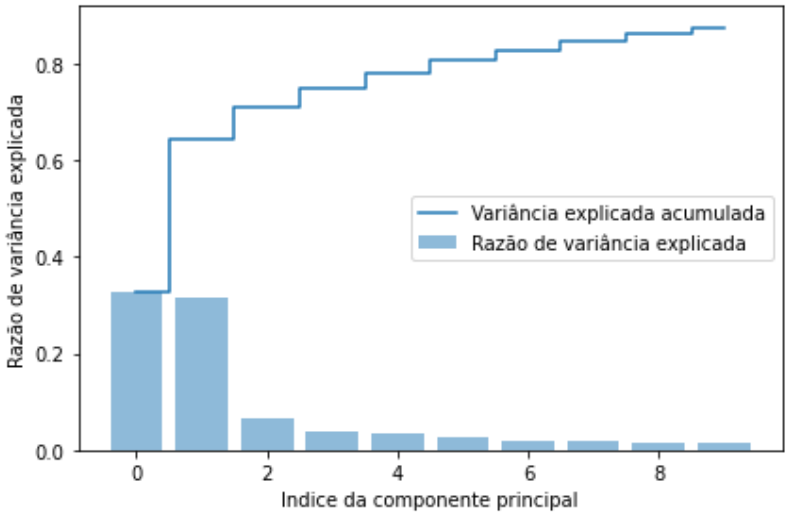

Figura 15 – Variância explicada pelas primeiras 10 componentes principais.

Nesse sentido é possível representar as amostras utilizando um espaço tridimensional onde os eixos seriam as três componentes. Na Figura 16 esse espaço está representado e estão posicionadas 80 amostras dividias por condição de operação. Vale ressaltar que, através dessa representação a diferenciação das classes fica mais clara quando comparada com a Figura 14, mesmo com apenas 71% da variância total dos dados representada. Os dados das classes “Desbalanceado”, “Com atrito” e “Nominal” estão dispostos de forma aproximadamente circular e com diâmetros diferentes, facilitando a diferenciação entre as classes. Já a classe “Desalinhado”, está disposta no eixo vertical passando pelo centro dos círculos formados pelos outros dados, ficando clara a demarcação dessa classe.

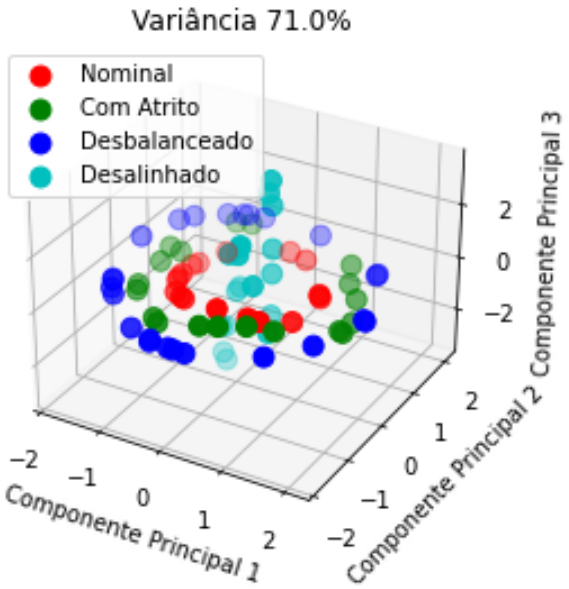

Figura 16 - Redução para 3 componentes principais permitiu a definição mais clara das classes.

5.2. Performance do KNN versus modelo linear

Para encontrar o modelo final otimizado foram executadas todas as etapas mencionadas na metodologia. Inicialmente os dados adquiridos na bancada de teste foram importados e as condições de operação codificadas. Em seguida, os dados passaram por um processo de normalização utilizando o método “MinMax” e foi aplicada a análise de componentes principais. Diferentes quantidades de componentes principais foram testadas com diversas combinações de hiperparâmetros dos dois modelos, na etapa de otimização. Dessa forma, com todas as informações será feita uma análise desses resultados observando a performance dos modelos em cada condição para, por fim, determinar as configurações do modelo final.

O modelo linear foi escolhido neste trabalho para ser um indicador de complexidade do problema de classificação em questão. Os resultados da Figura 17 mostram a variação de acurácia para diversas combinações de hiperparâmetros, considerando os diferentes percentuais de variância dos dados originais explicados pelas componentes principais. O modelo de regressão logística apresentou nos resultados da otimização valores de acurácia máxima perto de 30%. Além disso, fica claro, através da Figura 17, que o modelo não apresentou grandes melhorias no desempenho com a alteração do percentual

de variância representada. Dessa forma, pode-se afirmar que o problema em questão exige um classificador mais robusto.

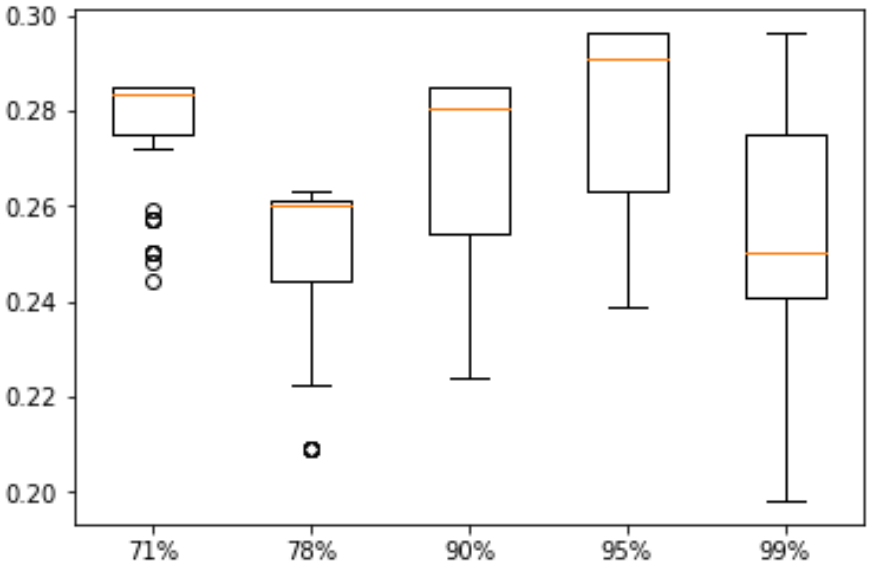

Figura 17 – Resultados de desempenho da regressão logística para vários percentuais da variância explicada pelo PCA. Vale ressaltar a baixar performance do modelo para todos os casos observados.

O modelo KNN abrange problemas mais complexos quando comparado ao modelo linear e, por isso, apresentou resultados superiores. Na Figura 18 pode-se observar que a performance do modelo atingiu valores mais altos para percentuais menores de variância explicada. Com apenas 71% da variância dos dados originais (3 componentes principais) o modelo chegou a valores de acurácia de aproximadamente 98%.

Vale ressaltar que o desempenho caiu significativamente para quantidades maiores de componentes principais, atingindo uma acurácia de aproximadamente 20% com 99% da variância explicada (Figura 18). A semelhança entre os sinais originais observada na Figura 14, dos dados originais, é um fator que possivelmente reduz o desempenho do modelo e, por isso, com menos componentes principais, como na Figura 16, as classes assumem características mais marcantes facilitando a classificação.

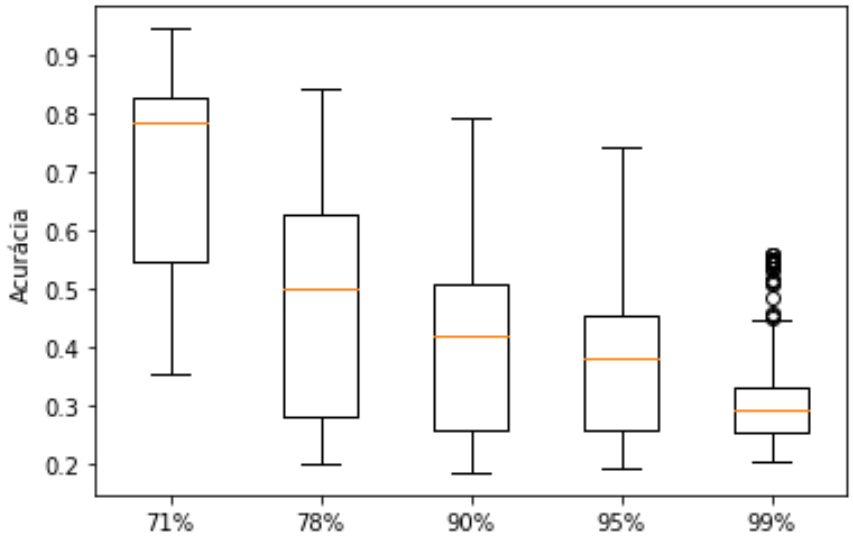

Figura 18 - Resultados de desempenho do KNN para vários percentuais da variância explicada pelo PCA. É possível observar a piora do modelo com o aumento da representação pelo PCA.

Analisando os resultados dos dois modelos na Figura 17 e na Figura 18 é possível afirmar que o modelo KNN apresentou uma performance muito superior e, por isso, será o algoritmo utilizado para o modelo final otimizado. Os resultados da otimização do modelo KNN estão dispostos na Tabela 3, onde acurácia é a métrica utilizada para performance, “k” é o número de vizinhos considerados para a classificação, “p” é o tipo de distância calculada e “Peso” define a influência da classe dos pontos próximos na classificação. Como foram consideradas 5 quantidades diferentes de componentes principais, existe uma configuração otimizada para cada quantidade.

Levando em conta a coluna de acurácia na Tabela 4, pode-se definir que a configuração otimizada é utilizando 3 componentes principais no PCA, “k” igual a 1, “p” igual a 2 e “Peso” uniforme.

Tabela 4 - Configurações otimizadas para diferentes percentuais de variância explicada.

Melhor combinação por PCA				
Variância explicada pelo PCA	Média da acurácia ± desvio padrão	k	p	Peso
71%	95% ± 0,060	1	2	Uniforme
78%	84% ± 0,061	1	2	Uniforme
90%	79% ± 0,057	1	2	Uniforme

95%	74% ± 0,056	1	2	Uniforme
99%	56% ± 0,057	1	2	Uniforme

Utilizando as configurações otimizadas foi treinado outro modelo KNN e nessas condições o modelo leva em conta apenas o ponto mais próximo, calculado utilizando a distância euclidiana, no espaço n-dimensional para a classificação.

Um modelo de aprendizado supervisionado como o KNN pode obter diferentes acurácias dependendo dos dados utilizados para o treinamento. Dessa forma, para avaliar a performance o modelo foi treinado, considerando 100 combinações diferentes de dados de treino e teste, e a acurácia atingiu aproximadamente 98% no melhor caso e pouco menos de 80% no pior (Figura 19).

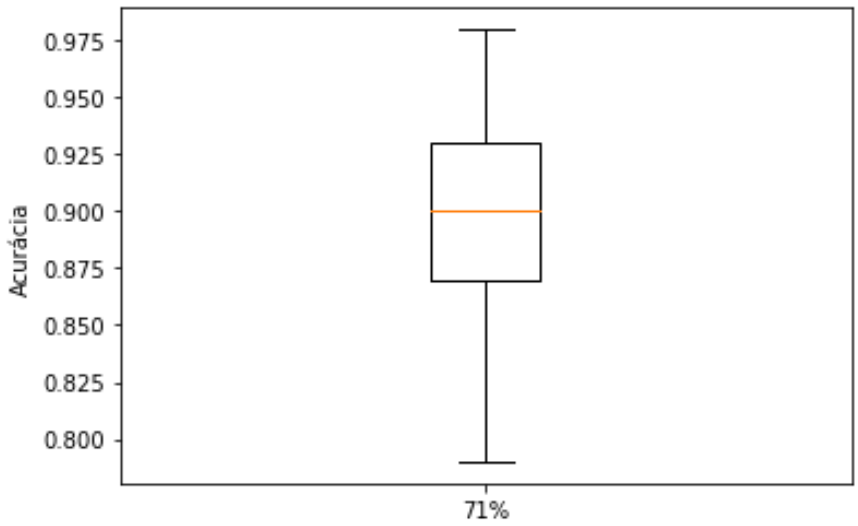

Figura 19 - Desempenho modelo otimizado alto quando comparado com os casos observados no processo de otimização.

A performance do melhor modelo da distribuição da Figura 19 pode ser avaliada utilizando os parâmetros mais comuns, como acurácia, matriz de confusão e precisão. A matriz de confusão resultante, Figura 20, mostra que todas as amostras das classes “Nominal” e “Desalinhado” foram classificadas corretamente. Porém, no caso das classes “Com atrito” e “Desbalanceado”, 2 amostras foram classificadas incorretamente como “Com atrito”, dessa forma, o modelo atingiu 98% de acurácia.

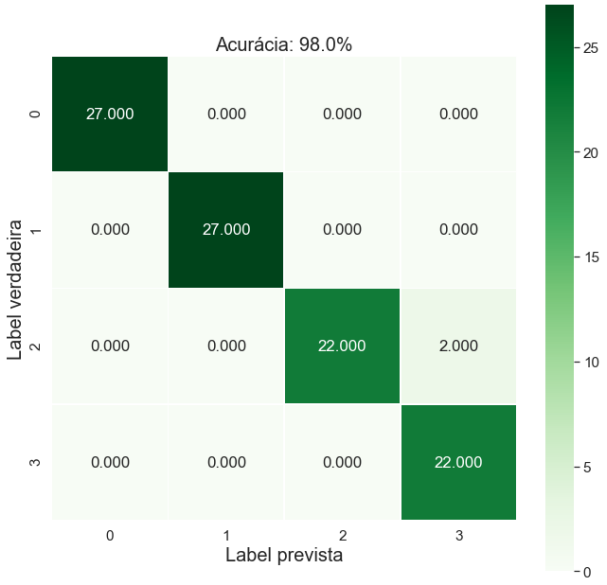

Figura 20 - Matriz de confusão revela a dificuldade de classificar elementos da classe 2.

As principais métricas podem ser observadas na Tabela 4. Como já foi exposto na matriz de confusão para as classes “Nominal” e “Com atrito” todas as amostras foram classificadas corretamente e, portanto, a precisão, o *Recall* e o *F1-score* são iguais a 1. Já para o “Desbalanceado” a precisão é igual a 1 porque todos os dados classificados como “Desbalanceado” realmente eram dessa classe. O *Recall* por outro lado ficou em 0,92 já que 2 amostras não foram classificadas corretamente. Com o *Recall* diferente de 1 o *F1-score* também não será 1 visto que depende desta métrica e da precisão. Por fim, para a condição de desalinhamento 2 amostras previstas como desalinhadas na verdade eram desbalanceadas e, por isso, a precisão não atingiu 100% e o *Recall* chegou a 1. Assim como na classe de desbalanceamento o *F1-score* atingiu 0,96. Nesse estudo de caso pode-se afirmar que a precisão é a métrica a ser observada, visto que indica o percentual dos valores classificados que estão corretos.

Tabela 5 – Principais métricas. O modelo obtido apresentou uma excelente performance.

Métricas				
Classe	Precisão	*Recall*	*F1-score*	Amostras por classe
Nominal	1	1	1	27
Com atrito	1	1	1	27

Desbalanceado	1	0,92	0,96	24
Desalinhado	0,92	1	0,96	22
Média	0,98	0,98	0,98	
Acurácia	0,98			

Os resultados obtidos mostram que o modelo apresenta uma performance excelente considerando sua simplicidade. Os modelos considerados por Liu, Zhihuai, et al. (2018), são gerados por um algoritmo de aprendizado não-supervisionado mais complexo. No entanto, utiliza métodos de extração de características consideravelmente complexos, realizando a decomposição, transformação e codificação de sinais. O modelo desenvolvido nesse trabalho, apesar de mais simples, obteve acurácia de 98% enquanto o melhor modelo avaliado por Liu, Zhihuai, et al. (2018) obteve acurácia em torno de 99,29%. Vale ressaltar, que o modelo, mesmo na melhor configuração obtida, apresentou problemas para diferenciar as classes 3 e 2.

De modo geral, os resultados das otimizações foram fundamentais para determinar as configurações ideais, além de mostras características contraintuitivas do comportamento dos modelos, como a queda de performance do KNN com o aumento do número de componentes. Com esses resultados foi possível observar também a incapacidade de aplicar o algoritmo de regressão logística para esse problema, e a importância de modelos mais complexos como o KNN, que aumentou a acurácia em aproximadamente 68% em relação ao modelo linear. Através das análises realizadas considerando as otimizações foi possível obter o melhor modelo possível considerando os algoritmos e dados utilizados.

6 Conclusão

Este trabalho teve como objetivo desenvolver um modelo de aprendizado de máquina supervisionado baseado em instância capaz de classificar diferentes condições em uma máquina rotativa. Com isso, demonstrou ser uma solução simples e eficaz para o problema de classificação.

O estudo baseou-se nos dados de vibração obtidos em uma bancada de testes construída especificamente para simular as condições de operação propostas (LIU, ZHIHUAI, *et al.*, 2018). Foram considerados dois modelos possíveis para a solução: regressão logística e KNN. Realizou-se a validação cruzada em conjunto com a calibração de hiperparâmetros, para os dois casos, com o objetivo de otimizar a solução dos modelos treinados.

Através dos resultados obtidos foi observado que o modelo KNN teve o melhor desempenho. Esse resultado já era esperado, considerando que a regressão logística é um modelo linear utilizado para casos mais simples. Utilizando os resultados do procedimento de otimização obteve-se um novo modelo KNN. Este apresentou 98% de acurácia e apenas 2 amostras não foram classificadas corretamente. Um aspecto interessante dos resultados da otimização, com relação ao KNN, é o modelo apresentar uma performance superior com menos componentes principais, ou seja, um percentual menor da variância explicada. Esse fato pode ser explicado pela semelhança dos sinais quando observados em sua forma original (Figura 14), que dificulta a classificação. Com menos componentes os dados ficam separados de maneira mais clara.

O desempenho obtido do modelo otimizado é competitivo, que conta com procedimentos amplamente difundidos na literatura, quando comparado com os resultados apresentados por LIU, ZHIHUAI, et al., (2018), que utiliza um modelo não-supervisionado (*k-means*) juntamente com uma extração de características consideravelmente mais complexa. Considerando a baixa complexidade da metodologia utilizada e a capacidade de classificar as condições de operação, pode-se concluir que o modelo obtido atende o objetivo de ser uma boa solução para o problema de classificação.

6.1. Trabalhos futuros

Esta última seção tem como objetivo propor possíveis melhorias e complementos ao presente trabalho. Algumas sugestões são:

- A utilização de outros bancos de dados que abordam problemas de máquinas rotativas em diferentes perspectivas como outras condições de operação ou métodos de aquisição de dados. Um exemplo seria o *Machinery Fault Database* (MaFaulDa) já utilizado para classificação de falhas (MARINS, RIBEIRO, *et al.*, 2018).
- Investigação de outros métodos de extração de características como a utilização da teoria de grafos para melhorar a performance da extração de características (YANG, ZHOU e LIU, 2021).
- Experimentar métodos mais robustos que obtém resultados no estado da arte como redes neurais convolutivas (YANG, ZHOU e LIU, 2021) e redes neurais recorrentes. Liu, Wang, et al. (2017), por exemplo, utilizou imagens infravermelho em conjunto com uma rede neural convolucional para o problema de classificação de falhas.

Referências bibliográficas

BRUNTON, S. L.; KUTZ, J. N. **Data Driven Science & Engineering**. [S.l.]: [s.n.], 2017.

G. S. K., R.; KUMAR VERMA, A.; RADHIKA, S. K-Nearest Neighbors and Grid Search CV Based Real Time Fault Monitoring System for Industries. **2019 IEEE 5th International Conference for Convergence in Technology (I2CT)**, p. 1-5, Março 2019.

HASNAIN, M. et al. Evaluating Trust Prediction and Confusion Matrix Measures for Web Services Ranking. **IEEE Access**, p. 90847-90861, Maio 2020.

HOLAND, S. M. **PRINCIPAL COMPONENTS ANALYSIS (PCA)**. University of Georgia. [S.l.], p. 1-12. 2019.

KIM, S.-H. **Electric Motor Control:** DC, AC and BLDC Motors. [S.l.]: Elsevier, 2017.

KSIAZEK, W.; GANDOR, M.; PAWEL, P. Comparison of various approaches to combine logistic regression with genetic algorithms in survival prediction of hepatocellular carcinoma. **Computers in Biology and Medicine**, p. 1-13, 2021.

LEI, Y.; HE, Z.; ZI, Y. A new approach to intelligent fault diagnosis of rotating machinery. **Expert Systems with Applications**, p. 1-8, 2008.

LEMLEY, J.; BAZRAFKAN, S.; CORCORAN, P. Deep Learning for Consumer Devices and Services: Pushing the limits for machine learning, artificial intelligence, and computer vision. **IEEE Consumer Electronics Magazine**, v. 6, p. 48-56, Abril 2017.

LIU, D. et al. Feature Extraction of Rotor Fault Based on EEMD and Curve Code. **Measurement**, 2018.

LIU, D. et al. Feature extraction of rotor fault based on EEMD and curve code. **Measurement**, v. 135, p. 1-13, Dezembro 2018.

LIU, R. et al. Artificial intelligence for fault diagnosis of rotating machinery. **Mechanical Systems and Signal Processing**, p. 1-15, Fevereiro 2018.

LIU, X. et al. Prediction of glass forming ability in amorphous alloys based on different. **Journal of Non-Crystalline Solids**, Junho 2021.

LIU, Z. et al. A hybrid fault diagnosis method based on second generation wavelet de-noising and local mean decomposition for rotating machinery. **ISA Transactions**, p. 211-220, Março 2016.

LIU, Z. et al. Infrared Image Combined with CNN Based Fault Diagnosis for Rotating Machinery. **IEEE**, p. 137-142, August 2017.

LOUTRIDIS, S. J. Gear failure prediction using multiscale local statistics. **Engineering Structures**, v. 30, p. 10, Agosto 2007.

MARINS, M. A. et al. Improved similarity-based modeling for the classification of rotating-machine failures. **Journal of the Franklin Institute**, Julho 2018.

RUSTAM, F. et al. COVID-19 Future Forecasting Using Supervised Machine Learning Models. **IEEE Access**, p. 1-11, Junho 2020.

SEIDEL, R. et al. Prediction of the Solder Rise in Selective Wave Soldering Comparing Decision Tree and Logistic Regression. **2020 43rd International Spring Seminar on Electronics Technology (ISSE)**, p. 1-7, 2020.

SHLENS, J. A Tutorial on Principal Component Analisys. **Google Research**, Abril 2014.

SINAGA, K. P.; YANG, M.-S. Unsupervised K-Means Clustering Algorithm. **IEEE Access**, p. 1-12, Maio 2020.

WU, X. et al. A hybrid classification autoencoder for semi-supervised fault diagnosis in rotating machinery. **Mechanical Systems and Signal Processing**, p. 16, Fevereiro 2021.

XUE, Y.; DOU, D.; YANG, J. Multi-fault diagnosis of rotating machinery based on deep convolution. **Measurement**, p. 7, Fevereiro 2020.

YANG, C.; ZHOU, K.; LIU, J. SuperGraph: Spatial-temporal graph-based feature extraction for rotating machinery diagnosis. **IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS**, 2021.

ZHAO, X.; PATEL, T. H.; ZUO, M. J. Multivariate EMD and full spectrum based condition monitoring for rotating machinery. **Mechanical SystemsandSignalProcessing**, p. 17, Agosto 2011.

ANEXO 1 – Código em Python de otimização de hiperparâmetros do modelo KNN.

```
#Importando bibliotecas
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy import io
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn import neighbors
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold

def tic():
    import time
    global startTime_for_tictoc
    startTime_for_tictoc = time.time()

def toc():
    import time
    if 'startTime_for_tictoc' in globals():
        print ('\nElapsed time is ')
        print (str(time.time() - startTime_for_tictoc))
        print('seconds.\n')
    else:
        print ('\nToc: start time not set\n')

#Importando dados do arquivo original
#Fonte: https://md-datasets-cache-zipfiles-prod.s3.eu-west-
1.amazonaws.com/p9bsmj4xwg-1.zip
dataGear = io.loadmat('180data_new_select_denoised.mat')
X = dataGear['Y_wavedeno']
labels = 45 * ['nominal'] + 45 * ['rubbing'] + 45 * ['unbalance'] + 45
* ['misalignment']
tic()

#Montando a tabela com os dados
data = pd.DataFrame();

for i in range(0,len(X)):
    teste = X[i];
    data["Amostra"+str(i)] = teste;

data = data.T;
features = list(data.columns);
x = data.loc[:, features].values
y = labels;
data["Condição"] = labels;

#Codificando as condições de operação
le = LabelEncoder()
y = le.fit_transform(y)
n_classes = np.unique(y).shape[0]
```

```
#Normalizando os dados
normalizer = MinMaxScaler()
X = normalizer.fit_transform(x)

#Definição das componentes consideradas para a otimização
list_pca = [3,10,.9,.95,.99]

#Divisão dos grupos para validação cruzada
cv = RepeatedStratifiedKFold(n_splits=2, n_repeats=10, random_state=2)

#Inicialização dos dados
cont = 0
dfr = pd.DataFrame(columns=["3 Componentes","10","90%","95%","99%"])
dfd =
pd.DataFrame(columns=["3","10","90%","95%","99%"],index=("Média","Desv
. Padrão","k","p","Peso"))
dfd = dfd.T

#Loop para combinações de componentes
for npca in list_pca:

    #PCA dos dados
    pca = PCA(n_components=npca,whiten=True)
    X_pca = pca.fit_transform(X)
    # Definição do modelo
    model = neighbors.KNeighborsClassifier()
    # Definição do espaço de busca
    k_range = list(range(1,30));
    p_range = list(range(1,6));
    wt = ["uniform","distance"];
    space = dict(n_neighbors=k_range, p=p_range, weights = wt)
    # Configuração da busca
    search = GridSearchCV(model, space, scoring='accuracy', n_jobs=-1,
cv=cv);
    # Executa a busca
    result = search.fit(X_pca, y);

    #Armazenando os resultados
    df = pd.DataFrame(result.cv_results_)
    dfr[dfr.columns[cont]] = df["mean_test_score"]
    dfd["Média"][cont] = result.best_score_
    dfd["Desv. Padrão"][cont] = df["std_test_score"].max()
    dfd["k"][cont] = result.best_params_["n_neighbors"]
    dfd["p"][cont] = result.best_params_["p"]
    dfd["Peso"][cont] = result.best_params_["weights"]
    cont = cont + 1

#Visualização dos dados da otimização
plt.boxplot(dfr,labels=["71%","78%","90%","95%","99%"])
plt.ylabel("Acurácia")
plt.show()
#Visualização das configurações otimizadas
print(dfd)
```

ANEXO 2 – Código em Python de otimização de hiperparâmetros do modelo de regressão logística.

```
#Importando bibliotecas
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy import io
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold

def tic():
    import time
    global startTime_for_tictoc
    startTime_for_tictoc = time.time()

def toc():
    import time
    if 'startTime_for_tictoc' in globals():
        print ('\nElapsed time is ')
        print (str(time.time() - startTime_for_tictoc))
        print('seconds.\n')
    else:
        print ('\nToc: start time not set\n')

#Importando dados do arquivo original
dataGear = io.loadmat('180data_new_select_denoised.mat')
X = dataGear['Y_wavedeno']
labels = 45 * ['nominal'] + 45 * ['rubbing'] + 45 * ['unbalance'] + 45
* ['misalignment']
tic()

#Montando a tabela com os dados
data = pd.DataFrame();

for i in range(0,len(X)):
    teste = X[i];
    data["Amostra"+str(i)] = teste;

data = data.T;
features = list(data.columns);
x = data.loc[:, features].values
y = labels;
data["Condição"] = labels;

#Codificando as condições de operação
le = LabelEncoder()
y = le.fit_transform(y)
n_classes = np.unique(y).shape[0]

#Normalizando os dados
normalizer = MinMaxScaler()
```

```
X = normalizer.fit_transform(x)

#Divisão dos grupos para validação cruzada
cv = RepeatedStratifiedKFold(n_splits=2, n_repeats=10, random_state=2)

#Definição das componentes consideradas para a otimização
list_pca = [3,10,.9,.95,.99]

#Inicialização dos dados
cont = 0
dfr = pd.DataFrame(columns=["3 Componentes","10","90%","95%","99%"])

#Loop para combinações de componentes
for npca in list_pca:

    #PCA dos dados
    pca = PCA(n_components=npca,whiten=True)
    print(X.shape)
    X_pca = pca.fit_transform(X)

    # Definição do modelo
    model = LogisticRegression(multi_class='ovr')
    # Definição do espaço de busca
    space = dict()
    space['solver'] = ['newton-cg', 'lbfgs', 'liblinear']
    space['penalty'] = ['none', 'l1', 'l2', 'elasticnet']
    space['C'] = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100]
    #https://machinelearningmastery.com/hyperparameter-optimization-
with-random-search-and-grid-search/
    # Configuração da busca
    search = GridSearchCV(model, space, scoring='accuracy', n_jobs=-1,
cv=cv);
    # Executa a busca
    result = search.fit(X_pca, y);

    #Armazenando os resultados
    df = pd.DataFrame(result.cv_results_)
    dfr[dfr.columns[cont]] = df["mean_test_score"]
    cont = cont + 1

#Visualização dos dados da otimização
dfr.dropna(inplace=True)
plt.boxplot(dfr,labels=["71%","78%","90%","95%","99%"])
plt.show()
```

ANEXO 3 – Código em Python de visualização dos dados e treinamento do modelo final otimizado.

```
#Importando bibliotecas
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy import io
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn import neighbors
from sklearn import metrics
import seaborn as sns

def tic():
    import time
    global startTime_for_tictoc
    startTime_for_tictoc = time.time()

def toc():
    import time
    if 'startTime_for_tictoc' in globals():
        print ('\nElapsed time is ')
        print (str(time.time() - startTime_for_tictoc))
        print('seconds.\n')
    else:
        print ('\nToc: start time not set\n')

#Importando dados
dataGear = io.loadmat('180data_new_select_denoised.mat')
X = dataGear['Y_wavedeno']
labels = 45 * ['nominal'] + 45 * ['rubbing'] + 45 * ['unbalance'] + 45
* ['misalignment']
t = np.linspace(0,1,2048);

#Visualizando dados originais
fig,axs = plt.subplots(2,2)
axs[0, 0].plot(t, X[0])
axs[0, 0].set_title('Nominal')
axs[0, 1].plot(t, X[46], 'tab:orange')
axs[0, 1].set_title('Com atrito')
axs[1, 0].plot(t, X[92], 'tab:green')
axs[1, 0].set_title('Desbalanceado')
axs[1, 1].plot(t, X[170], 'tab:red')
axs[1, 1].set_title('Desalinhado')

for ax in axs.flat:
    ax.set(xlabel='Tempo [s]', ylabel='Aceleração [g]')
for ax in axs.flat:
    ax.label_outer()

tic()
#Montando a tabela com os dados
data = pd.DataFrame();
```

```
for i in range(0,len(X)):
    teste = X[i];
    data["Amostra"+str(i)] = teste;

data = data.T;
features = list(data.columns);
x = data.loc[:, features].values
y = labels;
data["Condição"] = labels;

#Codificando as condições de operação
le = LabelEncoder()
y = le.fit_transform(y)
n_classes = np.unique(y).shape[0]

#Iniciando variável de armazenamento dos resultados
dfr = pd.DataFrame(np.zeros((100, 1)),columns = ["Acc"]);

#Iniciando loop de treinamento
for k in range(0,100):

    x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.55, random_state=k)

    #Normalizando os dados
    normalizer = MinMaxScaler()
    x_train = normalizer.fit_transform(x_train)
    x_test = normalizer.transform(x_test)

    #PCA dos dados
    pca = PCA(n_components=3,whiten=True)
    x_train_pcs = pca.fit_transform(x_train)
    x_test_pcs = pca.transform(x_test)

    #Criando o modelo KNN
    knn_model = neighbors.KNeighborsClassifier(n_neighbors=1, p=2)

    #Treinando o modelo
    knn_model.fit(x_train_pcs, y_train)

    #Avaliando acurácia
    score = knn_model.score(x_test_pcs, y_test)

    #Coletando dados
    dfr["Acc"][k] = score;

#Armazenando dados
dadoss = dfr["Acc"]

#Visualizando os resultados
plt.boxplot(dadoss, labels = ["71%"])
plt.ylabel("Acurácia")

#Separando dados para treinamento otimizado
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.55, random_state=19)
```

```
#Normalizando
normalizer = MinMaxScaler()
x_train = normalizer.fit_transform(x_train)
x_test = normalizer.transform(x_test)

#PCA dos dados
pca = PCA(n_components=3,whiten=True)

x_train_pcs = pca.fit_transform(x_train)
x_test_pcs = pca.transform(x_test)

#Criando o novo modelo KNN
knn_model = neighbors.KNeighborsClassifier(n_neighbors=1, p=2)

#Treinando o modelo
knn_model.fit(x_train_pcs, y_train)

#Testando com dados de teste
predictions = knn_model.predict(x_test_pcs)

#Avaliando acurácia
score = knn_model.score(x_test_pcs, y_test)

#Gerando e visualizando matriz de confusão
cm = metrics.confusion_matrix(y_test, predictions)

print(cm)
plt.figure(figsize=(11,11))
res = sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square =
True, cmap = 'Greens');
plt.ylabel('Label verdadeira',size = 20);
plt.xlabel('Label prevista',size = 20);
all_sample_title = 'Acurácia: {0}%'.format(100*round(score,2))
plt.title(all_sample_title, size = 20);
plt.show()
print(metrics.classification_report(y_test, predictions))

#Preparando dados para representação 3D das amostras

dfpcs = pd.DataFrame(x_train_pcs,columns = ['principal component
1','principal component 2','principal component 3'])
dfpcs["Condição"] = y_train
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

targets = [1, 2, 3,0]
colors = ['r', 'g', 'b','c']
for target, color in zip(targets,colors):
    indicesToKeep = dfpcs['Condição'] == target
    ax.scatter(dfpcs.loc[indicesToKeep, 'principal component 1']
               , dfpcs.loc[indicesToKeep, 'principal component 2']
               , dfpcs.loc[indicesToKeep, 'principal component 3']
               , c = color
               , s = 80)
ax.legend(["nominal",'rubbing','unbalance','misalignment'])
```

```
ax.set_xlabel('Componente Principal 1')
ax.set_ylabel('Componente Principal 2')
ax.set_zlabel('Componente Principal 3')
ax.set_title("% variance " +
str(round(sum(pca.explained_variance_ratio_),2)))
plt.show()

toc()
```

ANEXO 4 – Código em Python para estudo de componentes principais.

```
#Importando bibliotecas

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy import io
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split


def tic():
    import time
    global startTime_for_tictoc
    startTime_for_tictoc = time.time()

def toc():
    import time
    if 'startTime_for_tictoc' in globals():
        print ('\nElapsed time is ')
        print (str(time.time() - startTime_for_tictoc))
        print('seconds.\n')
    else:
        print ('\nToc: start time not set\n')

#Importando dados
dataGear = io.loadmat('180data_new_select_denoised.mat')

#Montando a tabela com os dados
X = dataGear['Y_wavedeno']
labels = 45 * ['nominal'] + 45 * ['rubbing'] + 45 * ['unbalance'] + 45
* ['misalignment']
t = np.linspace(0,1,2048);
tic()
data = pd.DataFrame();

for i in range(0,len(X)):
    teste = X[i];
    data["Amostra"+str(i)] = teste;

data = data.T;
features = list(data.columns);
x = data.loc[:, features].values
y = labels;
data["Condição"] = labels;

#Codificando as condições de operação
le = LabelEncoder()
y = le.fit_transform(y)
n_classes = np.unique(y).shape[0]

#Inicializando variáveis de armazenamento
dfr = pd.DataFrame(np.zeros((100, 2)),columns = ["RS","Acc"]);
```

```
#Separando dados de treino e teste
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.55, random_state=19)

#Normalizando os dados
normalizer = MinMaxScaler()
x_train = normalizer.fit_transform(x_train)
x_test = normalizer.transform(x_test)

#PCA dos dados
pca = PCA(10)

x_train_pcs = pca.fit_transform(x_train)
x_test_pcs = pca.transform(x_test)

#Calculando a variância explicada por cada componente principal
exp_var_pca = pca.explained_variance_ratio_

#Calculando a variância explicada acumulada para cada componente
principal
cum_sum_eigenvalues = np.cumsum(exp_var_pca)

#Visualizando os dados calculados
plt.bar(range(0,len(exp_var_pca)), exp_var_pca, alpha=0.5,
align='center', label='Razão de variância explicada')
plt.step(range(0,len(cum_sum_eigenvalues)), cum_sum_eigenvalues,
where='mid',label='Variância explicada acumulada')
plt.ylabel('Razão de variância explicada')
plt.xlabel('Indice da componente principal')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```