Antonio Luiz da Silva Loca

Uma metodologia experimental para avaliar abordagens de aprendizado de máquina para diagnóstico de falhas com base em sinais de vibração

Vitória, ES

2020

Antonio Luiz da Silva Loca

Uma metodologia experimental para avaliar abordagens de aprendizado de máquina para diagnóstico de falhas com base em sinais de vibração

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Programa de Pós-Graduação em Informática

Orientador: Prof. Dr. Thomas Walter Rauber

Vitória, ES

2020

Ficha catalográfica disponibilizada pelo Sistema Integrado de Bibliotecas - SIBI/UFES e elaborada pelo autor

---

Loca, Antonio Luiz da Silva, 1986-
L811m Uma metodologia experimental para avaliar abordagens de aprendizado de máquina para diagnóstico de falhas com base em sinais de vibração / Antonio Luiz da Silva Loca. - 2020.
126 f. : il.

Orientador: Thomas Walter Rauber.
Dissertação (Mestrado em Informática) - Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Algorítmos computacionais. 2. Inteligência computacional. 3. Aprendizado do computador. 4. Sistemas de reconhecimento de padrões. 5. Classificação. 6. Localização de falhas. I. Rauber, Thomas Walter. II. Universidade Federal do Espírito Santo. Centro Tecnológico. III. Título.

CDU: 004

Antonio Luiz da Silva Loca

Uma metodologia experimental para avaliar abordagens de aprendizado de máquina para diagnóstico de falhas com base em sinais de vibração

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Trabalho aprovado. Vitória, ES, 04 de junho de 2020:

Prof. Dr. Thomas Walter Rauber
Orientador

Flávio Miguel Varejão
Membro interno

Francisco de Assis Boldt
Membro externo

Vitória, ES
2020

Ao meu eterno amado pai.

Agradecimentos

Agradeço a Deus, o grande arquiteto do universo, pelo dom da vida e pelo desejo constante de busca pelo conhecimento.

Ao meu orientador Dr. Thomas Walter Rauber pelos seus ensinamentos, colaborações e conselhos valiosos e essenciais para este trabalho, além de todo o incentivo e motivação. Aos professores Dr. Flávio Miguel Varejão e Dr. Francisco de Assis Boldt pela ajuda imprescindível para a concretização deste trabalho.

À minha amada Gessica por todo o seu amor e carinho durante esta fase tão importante da minha vida.

“ Is there anyone so wise as to learn by the experience of others? ”
(Voltaire)

Resumo

Este trabalho apresenta um procedimento sistemático para comparar de maneira justa os valores de desempenho experimental para abordagens de aprendizado de máquina para diagnóstico de falhas com base em sinais de vibração. Na vasta maioria das publicações científicas relacionadas, a estimativa de acurácia e critérios de desempenho semelhantes são os únicos parâmetros de qualidade apresentados. A metodologia que foi usada para gerar os resultados dessas publicações é predominantemente tendenciosa, com base em métodos de validação demasiadamente simples. Além disso, todos os métodos, em geral, reciclam padrões idênticos para estimar os melhores hiperparâmetros, introduzindo super ajuste adicional nos resultados. Para reparar esse problema, foram criticamente analisadas as condições usadas na divisão de treinamento, validação e teste e foi proposto um algoritmo que permite uma comparação bem definida dos resultados experimentais. Para ilustrar as ideias do trabalho, o *benchmark Case Western Reserve University Bearing Data* é usado como um caso de estudo. Quatro classificadores distintos são comparados experimentalmente, sob tarefas de generalização mais difíceis usando a estrutura de avaliação proposta: K-vizinhos mais próximos, Máquina de Vetor de Suporte, Floresta Aleatória e Rede Neural Convolucional unidimensional. Uma extensa revisão da literatura sugere que a maioria dos trabalhos de pesquisa da *Case Western Reserve University Bearing Data* usa padrões semelhantes para treinamento e teste, tornando sua classificação uma tarefa fácil.

Palavras-chaves: Detecção de falhas, Dados de falha de rolamentos CWRU, Critérios de desempenho, Classificação, Reconhecimento de padrões, Aprendizado de máquina.

Abstract

This work presents a systematic procedure to fairly compare experimental performance values for machine learning approaches for fault diagnosis based on vibration signals. In the vast majority of related scientific publications, the estimate of accuracy and similar performance criteria are the only quality parameters presented. The methodology that was used to generate the results of these publications is predominantly biased, based on validation methods that are too simple. In addition, all methods, in general, recycle identical patterns to estimate the best hyperparameters, introducing additional overfitting in the results. To repair this problem, the conditions used in the training, validation and test division were critically analyzed and an algorithm was proposed that allows a well-defined comparison of the experimental results. To illustrate the work's ideas, the Case Western Reserve University Bearing Data benchmark is used as a case study. Four distinct classifiers are compared experimentally, under more difficult generalization tasks using the proposed evaluation structure: K-Nearest Neighbors, Support Vector Machine, Random Forest and One-dimensional Convolutional Neural Network. An extensive review of the literature suggests that most research work at Case Western Reserve University Bearing Data uses similar standards for training and testing, making classification an easy task.

Keywords: Fault detection, CWRU bearing fault database, performance criteria, classification, pattern recognition, machine learning.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CV	*Cross Validation*
DE	*Drive End*
FE	*Fan End*
ML	*Machine Learning*
TE	*Tennessee Eastman*
CWRU	*Case Western Reserve University*
CNN	*Convolutional Neural Network*
CPU	*Central Process Unit*
GPU	*Graphics Processing Unit*
MLP	*Multilayer Perceptron*
SVM	*Suport Vector Machine*
RF	*Random Forest*
KNN	*K-nearest neighbors*
RNA	Redes Neurais Artificiais
ANN	*Artificial Neural Network*
MATLAB	*MATrix LABoratory*
AEM	*Abnormal Events Management*
AF	*Activation Function*
REB	*Rolling Element Bearings*

Sumário

1 Introdução

O controle de processos complexos dependeu unicamente de operadores humanos durante muito tempo, tarefas regulatórias como gerenciar a atividade de equipamentos, ativar ou desativar válvulas, entre outros, sempre estiveram confiadas a mãos humanas até a grande difusão dos sistemas computadorizados de controle. A dependência completa de operadores humanos para lidar com eventos anormais e emergências na indústria torna-se cada vez mais rara na atualidade devido a fatores ligados à segurança do pessoal bem como da necessidade da tomada de ações se processar no tempo mais curto possível, em alguns casos, inferior ao tempo de reação do ser humano.

A tarefa de identificar falhas em equipamentos se apresenta verdadeiramente difícil para um operador devido ao amplo escopo da atividade de diagnóstico que engloba uma variedade de defeitos, como as falhas nas máquinas, degradação dos equipamentos, desvios de parâmetros e assim por diante. Tudo tende a ser ainda mais complicado devido ao tamanho e complexidade das modernas plantas industriais. Em uma dessas grandes plantas, pode haver numerosas variáveis de processo sendo observadas a cada poucos segundos levando à sobrecarga de informações. Além disso, muitas vezes a maior necessidade reside no rápido diagnóstico da falha o que impõe restrições e exigências significativas para o diagnóstico humano. Ainda assim, a tarefa de diagnóstico de falha pode ser dificultada pelo fato de as medições do processo serem insuficientes, incompletas e/ou não confiável devido a uma variedade de causas, como distorções ou falhas do sensor (YÉLAMOS et al., 2009).

O desenvolvimento de sistemas automáticos de controle de processos nas últimas décadas até a atualidade reduziu significativamente os custos operacionais da indústria. Os benefícios gerados por tais sistemas foram enormes para vários segmentos industriais, contudo, tarefas muito importantes para o gerenciamento dos processos ainda permaneceram em sua maioria, como atividades manuais, realizada pelos operadores. Essas práticas adotadas do ponto de vista econômico podem resultar em prejuízos financeiros além de, negligenciando a segurança do processo, pôr em risco a integridade de toda a planta e dos operadores. O cenário atual para a supervisão de processos em instalações modernas é altamente complexo, onde centenas de variáveis de processo podem interagir entre si.

O diagnóstico de falhas tornou-se uma tarefa muito difícil, exigindo sistemas sofisticados de suporte para auxiliar os operadores diante de eventos anormais da planta. Isso envolve a detecção oportuna de um evento anormal, diagnosticando suas origens causais e, em seguida, tomada de decisões e ações apropriadas de controle de supervisão para trazer o processo de volta a um estado operacional normal e seguro, consistindo a tarefa em

responder a eventos anormais que surgem durante um processo (VENKATASUBRAMANIAN et al., 2003). A importância associada a esta tarefa motivou o desenvolvimento de inúmeras técnicas buscando o melhor método automático para diagnóstico de falhas. Nesse contexto, o gerenciamento de eventos anormais (AEM) tornou-se essencial em pesquisas de operações de processos relacionados à segurança (VENKATASUBRAMANIAN, 2003).

Um evento anormal pode ser definido como qualquer desvio de um processo de sua faixa aceitável de operação. O AEM lida com essas situações através de detecção, diagnóstico e planejamento de contramedidas durante a operação dos equipamentos que mantém o processo, sendo as partes mais importantes a segurança e a otimização (DASH; VENKATASUBRAMANIAN, 2003). Essa abordagem representa uma segurança *on-line* da linha de produção tendo por responsabilidade diagnosticar desvios do processo e, apoiado em informações conhecidas do sistema, tomar decisões corretivas adequadas e a tempo. Para isso, o AEM depende de sistemas adequados de aquisição e processamento de dados proveniente dos equipamentos, bem como de ferramentas eficazes de apoio à tomada de decisão que também são fundamentais para alcançar um bom desempenho geral.

1.1 Motivação

Um diagnóstico precoce, quando a planta ainda não está fora de controle, é sempre crucial para evitar consequências mais perigosas. Muito foi feito na literatura relatada a respeito da busca por altos valores de acurácia no diagnóstico de falhas com dados amostrais de operação de diferentes modelos, mas poucos trabalhos trataram de garantir o isolamento do conjunto de teste de modo a não permitir que o sistema aprenda e teste com os mesmos dados, ou que as fontes de aquisição para treino e teste sejam diferentes, o se traduz na confiabilidade dos resultados. Outro ponto sensível para a aplicabilidade do método de aprendizado é a possibilidade de garantir os mesmos valores de precisão alcançados, ou próximos, para uma diversidade de situações operacionais dos equipamentos considerando o viés de semelhança existente entre dados de treino e teste.

1.2 Hipóteses

- É desejável realizar o diagnóstico de falhas em um conjunto de dados com base em vibração seguindo uma metodologia que, dentro dos recursos disponíveis, aproxime ao máximo os resultados de desempenho de situações reais de avaliação.

- É desejável a obtenção de valores mais justos para a classificação quando se utiliza automatização para escolha dos melhores conjuntos de hiperparâmetros para cada classificador, ao invés de, à exaustão, testar diferentes combinações para somente relatar resultados obtidos com os melhores valores encontrados nessa busca.

- É desejável utilizar uma base pública de dados de sinais de vibração para fornecer aquisição de informações para diagnóstico de falhas evitando o viés de semelhança para produzir resultados com confiabilidade.

1.3 Objetivos

Este trabalho tem como principais objetivos discutir e avaliar estratégias de uso de aprendizado de máquina no contexto de diagnóstico de falhas e propor uma nova metodologia para utilização de conjuntos de dados de sinais baseados em vibração.

Utilizar uma distribuição amostral dos dados com base na metodologia experimental proposta e avaliar os valores de exatidão para os diferentes classificadores em uma situação de maior semelhança possível com condições práticas diversas de operação dos equipamentos.

Para concretizar tais objetivos, tem-se os seguintes objetivos específicos:

- Utilizar a base de dados de rolamentos pública CWRU (*Case Western University*) como estudo de caso;

- Determinar o estado da arte das técnicas de avaliação de desempenho que utilizem CWRU para comparação de resultados;

- Avaliar quais métodos foram mais utilizados com a base CWRU e quais os seus valores médios de desempenho;

- Realizar os experimentos de avaliação nos modos de generalização propostos neste trabalho utilizando CWRU.

1.4 Metodologia

- Realização de uma extensa revisão bibliográfica sobre o uso de aprendizado de máquina para diagnóstico de falhas e de métodos de experimentação de classificadores;

- Identificação de falhas e incompletudes nessas métodos;

- Proposta de método abrangente que cubra os problemas;

- Realização de pesquisa bibliográfica de métodos de experimentação usados com CWRU;

- Implementação de método proposto para CWRU;

- Realização de experimentação em CWRU usando abordagens alternativas e gradualmente mais difíceis de generalização com o uso, em cada método, de quatro classificadores: *K-Nearest Neighbors*, *Support Vector Machine*, *Random Forest* e *Convolutional Neural*

Network.

1.5 Estrutura da Dissertação

Esta dissertação está organizada em 6 capítulos. O primeiro capítulo aborda a introdução onde é realizada uma breve descrição do problema tratado, a motivação para o desenvolvimento da pesquisa, os principais objetivos do trabalho e a metodologia utilizada. No segundo capítulo são listados alguns conceitos de base, necessários para o entendimento do estudo. O terceiro capítulo exibe a revisão da literatura relacionada e detalhes sobre a origem e formato dos dados utilizados. No quarto capítulo são mostrados detalhes mais técnicos, referentes a implementação, arquitetura, plataforma e outros aspectos sobre o trabalho além dos resultados de comparações entre diferentes métodos de avaliação. No quinto capítulo são apresentadas as contribuições, a análise estatística dos dados e os resultados obtidos com a execução da metodologia proposta. Por fim, no sexto capítulo são expostas as considerações finais sobre o trabalho e o futuro da pesquisa. Anexo como apêndice, o código do algoritmo que foi utilizado para o trabalho bem como os resultado da execução diretamente da execução do algoritmo.

2 Aprendizado de Máquina

No campo de Inteligência Artificial, *Machine Learning* ocupa uma posição de destaque. Seu uso representa uma ruptura de paradigma na concepção de como as máquinas podem ser utilizadas para manipular informações previamente aprendidas. O conceito é que ao ter acesso a um grande volume de dados, os algoritmos podem aprender sozinhos a extrair informações importantes dos dados, que serão utilizadas em momento oportuno para controle dos processos dos quais foram originadas, sem que tenham sido programados com as instruções.

Os processos de aprendizado de máquina são normalmente classificados em duas categorias amplas, de acordo com a forma em que o retorno de aprendizado é disponibilizado para o sistema: Aprendizado supervisionado e não supervisionado.

2.1 Aprendizado supervisionado

São apresentados ao algoritmo vários exemplos de entradas e saídas desejadas, a tarefa necessita que haja o papel de um "professor" que forneça os dados ao sistema com seu respectivo significado, por exemplo, um operador que precisa rotular os dados obtidos de um equipamento em condições de correto funcionamento e com algum tipo de falha.

O objetivo esperado do classificador é que ele seja capaz de aprender uma regra geral que mapeia as entradas para as saídas podendo essa regra ser utilizada posteriormente para novas predições em um outro conjunto de dados que não foi visto anteriormente. Desta forma, ele consegue realizar a classificação dos dados em diferentes categorias de acordo com o aprendizado. Outro problema supervisionado possível é a regressão onde as saídas esperadas do algoritmo de aprendizado são contínuas, em vez de discretas.

Outra forma de Aprendizado supervisionado é o aprendizado por reforço, onde o algoritmo mantém uma interação constante com um ambiente dinâmico, onde ele deve desempenhar um determinado objetivo como por exemplo, conduzir um veículo. Ao algoritmo é fornecido um *feedback* da sua atuação como forma de premiações e punições, na medida em que é navegado o espaço do problema. Também como exemplo de aprendizado por reforço, podemos citar uma tarefa que consiste no programa aprender a jogar um certo tipo de jogo apenas jogando contra um oponente.

2.2 Aprendizado não supervisionado

Neste modo, nenhum tipo de rótulo é fornecido ao algoritmo de aprendizado, deixando-o encontrar sozinho estruturas de agrupamento nos dados fornecidos. A tarefa mais básica do aprendizado não supervisionado é possibilitar a descoberta de padrões latentes nos dados os agrupando por padrões similares que não eram conhecidos *a priori*. Na técnica de *clustering*, um conjunto de dados de entrada após ser fornecidos ao algoritmo, é dividido em diferentes grupos na saída após o processamento do aprendizado através de análise de padrões estatísticos. Diferentemente da classificação, os grupos gerados não são conhecidos previamente, tornando o *clustering* uma tarefa caracteristicamente não supervisionada.

2.3 Reconhecimento de Padrões

Esta é uma área da ciência de rápido desenvolvimento, que sustenta os avanços tecnológicos em vários campos relacionados, como processamento de imagens, visão computacional, análise de texto e documento e Redes Neurais. De acordo com cada aplicação, os objetos de estudo podem variar entre imagens, sinais em forma de ondas como voz, texto, luz, rádio ou qualquer outro tipo de medida que necessite ser classificada. Um sistema para reconhecimento de padrões engloba três grandes etapas: representação dos dados de entrada e sua mensuração, extração das características e finalmente identificação e classificação do objeto em estudo.

O avanço do estudo do reconhecimento de padrões em inteligência artificial trouxe a possibilidade de os computadores aprenderem a executar tarefas de classificação de dados sem serem categoricamente programados e inúmeros algoritmos foram criados para dar suporte ao aprendizado de máquina. Uma grande parte do mecanismo de aprendizado de tais algoritmos está diretamente ligada a estatística. Alguns classificadores são totalmente construídos seguindo análise estatística, uns não a usam, enquanto um certo número a usam de forma combinada com diferentes abordagens, que se utilizam de outras técnicas como meios para a aprendizagem. Para que o processo de aprendizado seja possível, é preciso construir um modelo a partir de entradas amostrais fornecidas ao algoritmo que vão direcionar o processo.

2.3.1 Classificadores

No contexto de aprendizado de máquina, a classificação pode ser definida como parte do processo de aprendizado que visa identificar atributos de um conjunto de dados de entrada e atribuir rótulos às saídas desses dentro de um cenário previamente analisado. Como exemplo, poderíamos ter a situação onde um arquivo contendo inúmeras medições de sensores de um maquinário seria a entrada para o classificador e esse deveria rotular

cada medição ou conjunto de medições em um estado normal de operação ou de um estado de falha da máquina.

A classificação é usada normalmente quando as previsões tem natureza distinta, ou seja, as saídas podem ser simples "sim ou não" nesse caso, as classificações são do tipo binárias, quando há apenas duas respostas possíveis. A classificação também pode ser do tipo multiclasse, que são aquelas que possuem várias características como diferentes falhas no exemplo anterior: falha A, falha B, etc.

A figura 1 exibe uma simples hierarquia de diferentes classificadores que possuem uma grande utilização. Pode parecer, a princípio, que os diferentes algoritmos de classificação existentes possuem grandes semelhanças, mas estes têm objetivos e funções distintas, e dependendo da situação, pode-se escolher entre dois diferentes tipos de abordagem: Estatística ou Estrutural.

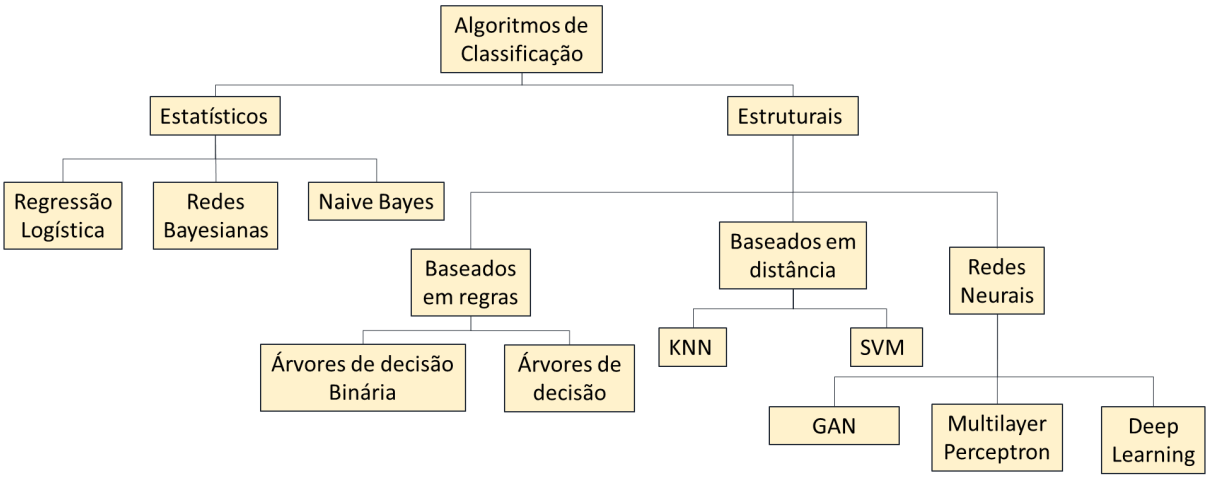


Figura 1 – Exemplo de algoritmos classificadores de *Machine Learning*

2.3.1.1 *K-nearest neighbor* (KNN)

O aprendizado baseado em distância dá origem a algoritmos de aprendizado de máquina que, em vez de executar generalização explícita, compara novas instâncias de problemas com instâncias vistas em treinamento, que foram armazenadas na memória. É chamado também de aprendizado baseado em instância porque constrói hipóteses diretamente a partir das próprias instâncias de treinamento. Isso significa que a complexidade computacional do algoritmo aumenta com o tamanho do conjunto de dados e no pior caso, para conjuntos de treinamento muito grandes, o KNN pode ser estocástico tomando uma amostra do conjunto de dados de treinamento para a partir da qual calcular as instâncias mais semelhantes (SCIENCE, 2019). Uma vantagem que o aprendizado baseado em instância tem sobre outros métodos de aprendizado de máquina é sua capacidade de adaptar seu modelo a dados não vistos anteriormente assim como a simplicidade em armazenar uma nova instância ou jogar fora uma instância antiga.

O algoritmo *K-nearest neighbor* (KNN) é um método típico de aprendizado baseado em instâncias e é amplamente usado na classificação de texto, análise de clientes e assim por diante.(ZHAO; TANG; DAI, 2012). A ideia básica é que um objeto seja classificado de acordo com a voto majoritário de seus vizinhos, com o objeto sendo atribuído à classe que pertence a maioria de seus k vizinhos mais próximos. Para o KNN encontrar os k vizinhos mais próximos é necessário calcular a distância da amostra de teste a todas as amostras de treinamento. Quando os dados da amostra de treinamento são muito grandes, eles produzem uma alta sobrecarga computacional, resultando em um declínio na velocidade de classificação.

Dado um conjunto de treinamento, o algoritmo calcula primeiramente a distância entre a amostra x e o conjunto de treinamento e depois encontra as k amostras de treinamento mais próximas. Assim, podemos atribuir x à classe em que a maioria das k amostras de treinamento é classificada como pertencentes. A fórmula que permite classificar uma amostra em uma determinada classe no algoritmo KNN pode ser descrita como abaixo.

$$C_k = min_1^k(D(x, c_{ij}))$$

onde $D(x, c_{ij})$ é a distância entre x e c_{ij}, i representa o número de classes, j representa o número de amostras no conjunto de treinamento. Se a maioria das amostras de treinamento k satisfizer a condição $max(C_k) \in C_n$, a amostra de teste x pertence a classe C_n.

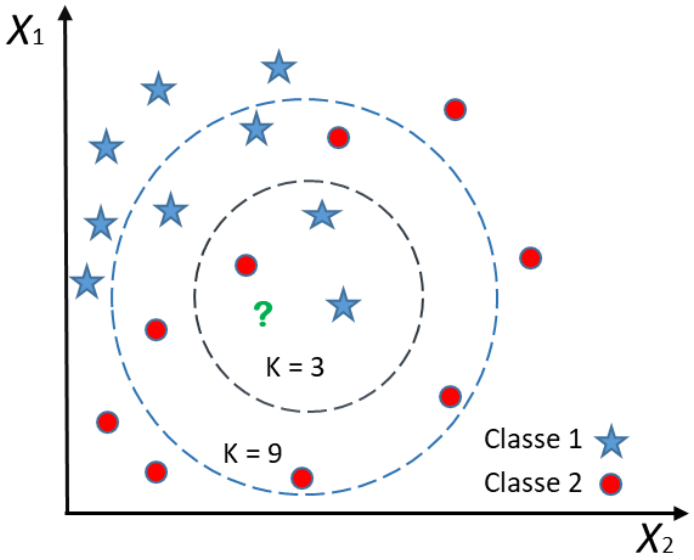


Figura 2 – Classificador *K-nearest neighbor*.

2.3.1.2 *Random forest* (RF)

O classificador *Random forest* (RF) é uma das técnicas de aprendizado mais bem-sucedidas, é comprovadamente uma das técnicas mais populares e poderosas no reconhecimento de padrões e aprendizado de máquina para classificação de alta dimensão e problemas com alta variação nos dados (BREIMAN, 2001b). As florestas aleatórias são uma combinação de árvores preditoras, de modo que a decisão de cada árvore depende dos valores de um vetor aleatório amostrado independentemente e com a mesma distribuição para todas as árvores da floresta. O erro de generalização de uma floresta de classificadores de árvores depende da força das árvores individuais na floresta e da correlação entre elas. Melhorias na precisão da classificação resultam do crescimento de um conjunto de árvores e do seu voto na classe mais popular. Para classificar um novo vetor de entrada, cada árvore vota em uma classe e a floresta escolhe a classe com a maioria de votos sobre todas as árvores na floresta. a Figura 3 exemplifica o processo de decisão desse classificador.

O algoritmo *Random Forest* (RF) tem recebido crescente interesse por causa de sua boa precisão e robustez ao ruído em comparação com os classificadores únicos. Ele é executado eficientemente em grandes bancos de dados e pode processar milhares de variáveis de entrada sem exclusão. Além disso, é computacionalmente mais leve que outros métodos de conjunto de árvores

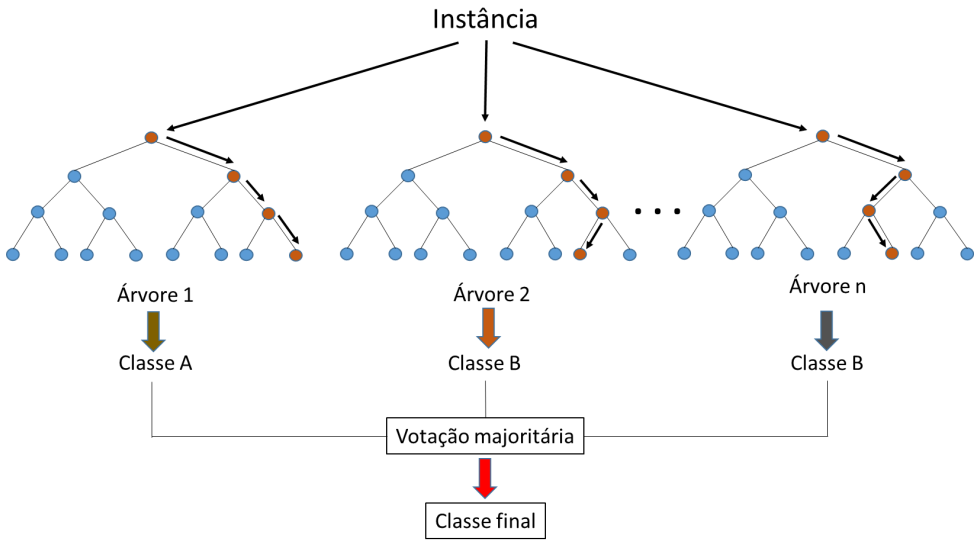


Figura 3 – Classificador *Random Forest*

2.3.1.3 *Support Vector Machine* (SVM)

As máquinas de vetores de suporte são classificadores supervisionados multivariados (que analisam o comportamento de três ou mais variáveis simultaneamente). Analogamente à regressão logística, o SVM foi inicialmente concebido para classificação em duas classes. Mais tarde, essa abordagem foi estendida para tratar de resultados e classificações contínuas

com mais de duas classes. A Figura 4 mostra um exemplo de uso de SVM para classificação com duas variáveis. O resultado da categorização é a associação a uma das classes (bola ou estrela) da figura 4, que é indicado pelo símbolo gráfico. Podemos desenhar muitas linhas que separam completamente as duas classes, duas das quais são mostradas na figura 4 (A e B). Para responder à pergunta de qual é a melhor linha que separa as classes, uma escolha sensata seria escolher uma linha de separação de modo que a margem (a distância mínima entre a linha e o ponto mais próximo) seja maximizada. É justamente essa a característica do SVM, garantir a separação máxima das classes através dos vetores de suporte. O algoritmo trabalha encontrando um hiperplano que melhor separa os dois grupos de dados. O SVM é treinado com dados em um espaço de treinamento n dimensional, após isso, as amostras de teste são classificadas de acordo com sua posição no espaço de recursos n dimensional. O SVM classifica as amostras em diferentes classes construindo um hiperplano no espaço de recursos de alta dimensão que pode ser usado para classificação. O hiperplano pode ser representado pela equação

$$w.x + b = 0$$

onde w é o vetor de peso normal ao hiperplano e b o viés ou limiar. Como classificador, o SVM pode ser usado, por exemplo, para classificar os recursos extraídos de imagens em duas classes ou mais, como para imagens médicas classificando-as entre imagens normais e anormais (SELVATHI; EMALA, 2016).

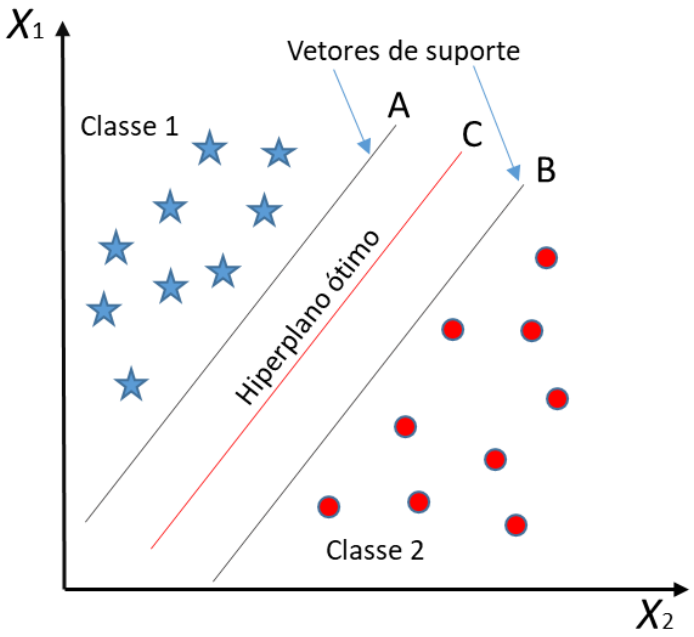


Figura 4 – Classificador *Support Vector Machine*

2.3.2 Redes Neurais Artificiais (ANN)

As ANN são sistemas computacionais baseados em uma rede de nós compostos por uma determinada quantidade de entradas e unidades de processamento, denominados por neurônios artificiais, os quais são ligados através de pesos sinápticos, ver figura 5, que executam funções lógico matemáticas inspirados por estruturas de armazenamento e processamento de origem biológica. Essa rede de neurônios compõem um modelo de aprendizagem que é utilizado para aproximar funções que dependem de um grande número de entradas. O processamento dos dados se inicia com uma fase de aprendizado, na qual um conjunto de dados para os quais já se conhecem os rótulos é apresentado ao programa, fazendo com que as forças das conexões da rede se alterem de modo a gerar um resultado que seja o mais próximo possível daquele observado nos dados de treinamento. As entradas são propagadas através da topologia da RNA, sendo transformadas pelos pesos sinápticos e pela função de ativação (AF) dos neurônios. Espera-se, posteriormente, que a rede adquira aptidão para a generalização, ou seja, tenha capacidade de fornecer rótulos para dados não conhecidos previamente (SIQUEIRA-BATISTA et al., 2014).

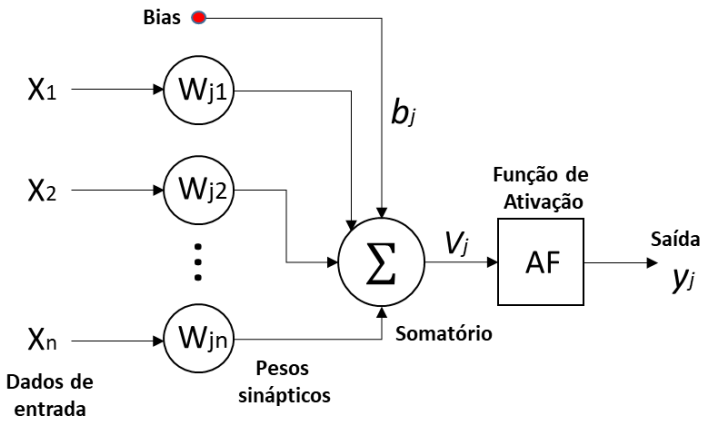


Figura 5 – Modelo de Neurônio Artificial de McCuloch e Pitts

Combinando os neurônios artificias em uma ou mais camadas conforme figura 6, que podem conter um ou mais neurônios e interligando estes neurônios através das sinapses, pode-se formar uma RNA (Rede Neural Artificial) possibilitando a utilização de processamento paralelo no aprendizado. Com mais de uma camada de neurônios em alimentação direta pode-se construir redes denominadas *Perceptron* Multicamadas (MLP) onde, através de um algoritmo de retro-propagação do erro, a rede pode aprender funções mais complexas. Dentre as vantagens das redes neurais, estão sua adaptação por experiência, tolerância a falhas, aplicações em tempo real, capacidade de aprendizado e de resolver problemas práticos sem a necessidade da definição de lista de regras (MACHADO; JUNIOR, 2013).

Recebendo entradas de n neurônios (y_i), o neurônio k calcula a sua saída através de:

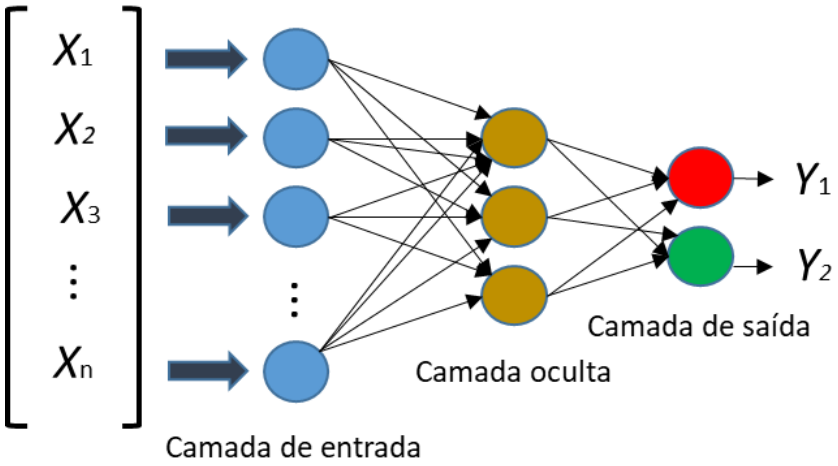


Figura 6 – *Multilayer perceptron*

$$y_k = AF\left(\sum_{i=1}^{n}(y_i w_{ki}) + b_k\right)$$

onde y_i é a saída calculada pelo neurônio i, w_{ki} representa o peso sináptico entre o neurônio i e o neurônio k e b_k é o peso entre um valor constante e diferente de zero ao neurônio k, conhecido como *bias*. Se o neurônio estiver ligado às entradas, o termo y_i é substituído pela entrada correspondente. Verifica-se que é necessário determinar os pesos sinápticos e bias para se utilizar uma RNA. A estimação de tais parâmetros, denominada de treinamento, se constitui de um processo iterativo onde os parâmetros iniciais são atualizados até a convergência do processo. Considerando a iteração j, o peso w_{ki} é atualizado através de:

$$w(j)_{ki} = w(j-1)_{ki} + \Delta w(j)$$

sendo o vetor de correção do parâmetro definido na iteração j. Funções de ativação não lineares do tipo sigmoidal antissimétrica são indicadas na solução de problemas de natureza não linear. Uma função que se enquadra nesta categoria é a denominada logística, a qual é dada por:

$$AF(x) = \frac{1}{1+e^{-x}}$$

onde x é calculado por:

$$x = \left(\sum_{i=1}^{n}(y_i w_{ki}) + b_k\right)$$

Os neurônios em redes MLP diferem-se do modelo de *McCuloch e Pitts*, por permitirem a saída de um valor qualquer ao invés de ter o somatório ponderado das

entradas comparado a um limiar gerando um sinal de saída binário (0 ou 1). Para isso, desenvolveram-se novos modelos de neurônios artificiais aos quais se aplicam funções de ativação aos valores ponderados da entrada. Essas funções são geralmente não lineares, para que possam representar problemas não lineares e diferenciáveis (FIORIN et al., 2011). O neurônio apresentado na figura 5 pode ser visto como um neurônio genérico dentro da estrutura de uma rede MLP.

Para (PAOLETTI et al., 2018), Redes Neurais Artificiais (RNA) têm sido amplamente utilizadas para classificação de dados, no entanto sua aplicação ainda enfrenta alguns desafios, especialmente relacionados ao processamento de informações de volume muito elevados, pois tal cenário resulta em um aumento significativo no tempo de computação necessário para o aprendizado, além de provocar a saturação dos parâmetros da rede (*overfitting*).

2.3.2.1 Aprendizagem Profunda (*Deep Learning*)

O tema tem sido muito discutido atualmente. A aprendizagem profunda se resume basicamente a uma categoria de algoritmos de *Machine Learning*, que na grande maioria das vezes, usam Redes Neurais Artificiais para gerar modelos. Trata-se de uma técnica avançada de aprendizado que busca otimizar o processo objetivando melhores resultados de classificação. É relatado com frequência na literatura que técnicas de *Deep Learning* foram muito bem sucedidas na função de resolver problemas relacionados com o reconhecimento de imagens devido à sua capacidade de selecionar os melhores atributos, utilizando uma representação em camadas para as propriedades das características.

2.3.2.2 Redes Neurais Convolucionais (CNN)

As CNN são predominantes no campo da visão computacional devido às suas vantagens por possuir elevada capacidade de generalização. CNNs 1D (Redes Neurais Convolucionais de uma Dimensão) também são ferramentas eficazes para sequenciamento de tempo (LIU et al., 2018). A CNN e suas variantes são amplamente implementadas no diagnóstico de falhas devido à sua superior capacidade de extração de recursos e mecanismo exclusivo de compartilhamento de peso. Outra característica da CNN é a capacidade de processar dados 1D e 2D, o que fornece mais flexibilidade para a implementação da CNN. Seu uso, diferente de abordagens tradicionais que utilizam pré-processamento dos sinais de forma estatística para posterior uso no aprendizado de uma RNA, possibilita que os dados sejam utilizados em sua forma bruta, dispensando a necessidade de qualquer processamento computacional que não seja o dispensado com a própria aprendizagem. A ideia por trás da aprendizagem profunda é descobrir múltiplos níveis de representação com a expectativa de que recursos de alto nível representem uma semântica mais abstrata dos dados (GUO et al., 2017). A aplicação de camadas de convolução em *data sets* pode ser

utilizada como extratora de características implícitas nos dados.

A convolução em dois sinais relacionados com o tempo e de função f e g respectivamente, é definida por:

$$(f * g)(t) - h(t) - \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau) d_{\tau}$$

Uma CNN é uma arquitetura composta de três camadas distintas: uma camada de entrada, uma camada convolucional e uma camada de *pooling*. O parâmetro k especifica o número de mapas de recursos n da camada convolucional.

$$(f * g)[n] \equiv \sum_{m=-\infty}^{\infty} f[n - m] g[m]$$

onde f é a camada de entrada e g é um dos k filtros que a CNN irá otimizar para uma função objetiva durante o processo de aprendizagem.

2.4 Diagnóstico de falhas em processos industriais

Após o desenvolvimento da computação, houve uma crescente demanda por aplicações com capacidade de reconhecer padrões que por sua vez trouxeram novas demandas por desenvolvimento teórico para criar classificadores melhores. Na fase pós-industrial da sociedade, a produção em larga escala cada vez mais automatizada, trouxe uma crescente necessidade de sistemas capazes de lidar com recuperação de informações de modelos de processos.

O reconhecimento de padrões tornou-se uma ferramenta fundamental para sistemas de tomada de decisão que apoiam e controlam complexos processos industriais. Um sistema de reconhecimento de padrões pode ser descrito em três partes constituintes: um sensor que faz a aquisição das amostragens que devem ser classificadas ou especificadas, um mecanismo de extração de características que faz análise e seleção das informações simbólicas ou numéricas das amostragens e de um algoritmo que realiza a classificação das amostras baseando-se no processamento de observações rotuladas apresentadas anteriormente no conjunto de treinamento (CORREIA; FERREIRA, 2007). Esta forma de classificação é caracterizada como um tipo de aprendizado supervisionado. Várias aplicações se beneficiam do reconhecimento de padrões tais como: classificação de documentos em diferentes categorias, reconhecimento de linguagem natural, reconhecimento facial, reconhecimento de escrita, detecção de falhas em equipamentos entre outros, seja por processamento de imagens ou de sinais de uma dimensão.

2.4.1 Aprendizado de máquina para diagnóstico de falhas

A dependência crescente da sociedade por tecnologia e produtos de fabricação industrial impulsionou o avanço da complexidade do parque industrial de diferentes setores fabris. A manutenção de tais sistemas tornou-se objeto de extrema importância econômica, o que explica o estudo de falhas em sistemas industriais ser tão antigo quanto os próprios sistemas em si (NANDI; TOLIYAT; LI, 2005).

Grande parte dos equipamentos responsáveis por gerar força motriz para manter os mais diversos processos em uma planta industrial são motores de indução, dentre as máquinas elétricas, os motores de indução possuem posição de destaque com uso amplo devido a sua confiabilidade, relativo baixo custo, robustez e eficiência energética (GRITLI et al., 2017). É fundamental manter o funcionamento destes equipamentos dentro do regime de trabalho que otimize o processo no qual fazem parte e que diminua ao máximo a necessidade de sua manutenção, pois isso além dos custos de reparo a situação ainda poderia levar a prejuízos causados pelo tempo que o processo ficou parado à espera do conserto da máquina e finalmente, reduzir o consumo de energia. Um motor elétrico com peças danificadas ou desgastadas tenderá a consumir mais eletricidade que a quantidade necessária para manter o seu correto funcionamento, levando em conta que dois terços de toda energia consumida pela indústria mundialmente é devido motores elétricos (KOTAK; JAIWAL; PATEL, 2016), o custo por não evitar que as falhas nos motores sejam identificadas prematuramente e, a tempo, contingenciadas pode se tornar extremamente oneroso.

Os rolamentos de elementos rolantes ou *Rolling-Element Bearing* (REB) são encontrados em quase todas as aplicações de engenharia e desempenham papel significativo para o funcionamento dos equipamentos que necessitam de movimentação de estruturas e engrenagens possibilitando a diminuição do atrito entre as peças girantes. Os REB estão entre os componentes mais precisos em conjuntos mecânicos, são produzidos com tolerâncias muito restritas e normalmente são encontrados na maioria dos equipamentos rotacionais (GEBRAEEL et al., 2004). Estes componentes também são responsáveis pela maioria dos defeitos nos equipamentos e causadores de interrupção do processo para manutenção do motor devido a serem as partes que mais sofrem desgaste pela elevada carga de trabalho a qual são submetidos durante o funcionamento da máquina.

É vasta a literatura acerca de trabalhos que propõe o uso das mais diversas técnicas para o diagnóstico de falhas em dados de rolamentos. Devido à importância do tema, a busca por melhores resultados na identificação de falhas nos REB evoluiu juntamente com o uso de técnicas de classificação. Um grande número de algoritmos tem sido aplicado ao problema sendo tratado como aprendizado supervisionado. O mais comumente encontrado é o uso de classificadores para diagnosticar falhas nos dados de rolamentos. O diferencial dos algoritmos classificadores em comparação com os demais é que esses em vez de

seguir condições previamente conhecidas para tomada de decisão, devem primordialmente aprender com seus erros para chegar a um nível de confiabilidade sobre suas futuras previsões. O nível dessa incerteza é algo crucial de ser conhecido para validar o uso do classificador em situações onde seu correto prognóstico pode significar grande economia de recursos ou sua falha pode conduzir a uma situação de falsa segurança que pode ser altamente perigosa.

O diagnóstico e o gerenciamento do estado do sistema são cruciais para máquinas cujos rolamentos são os principais componentes (ZHANG et al., 2017). Um sistema de diagnóstico de falhas eficiente deve possuir boa capacidade de generalização, sendo capaz de reconhecer o maior número possível de falhas, mesmo quando certas condições de falha não estavam presentes nos dados de treinamento. Para ser reconhecido como um método válido e efetivo de classificação, o procedimento dever seguir uma metodologia bem arquitetada caso contrário, cai-se na situação em que os resultados obtidos são super otimistas e não representam de forma confiável a realidade do diagnóstico como ele deveria se proceder na prática.

2.4.2 Métodos de diagnóstico

Softwares para diagnóstico automáticos de falhas são essenciais para dar garantia de segurança e manutenção de processos dinâmicos (GAO; CECATI; DING, 2015; CHIANG; BRAATZ; RUSSELL, 2001). Uma diferenciação essencial que pode ser feita nos processos utilizados para tal é: métodos de diagnóstico baseados em modelo (VARGA, 2017; GERTLER, 2017; DING, 2012) e de diagnóstico não baseado em modelo (DING, 2016; MCMILLAN; VEGAS, 2019). O acesso a dados confiáveis de equipamentos reais em diferentes situações de operação que possam ser utilizados como referência para diagnóstico é escasso.

Muitas técnicas foram propostas para avaliação do desempenho dos classificadores com modelos de diferentes graus de complexidade e confiabilidade. De todos os métodos, é esperado que deem confiabilidade ao processo para que este possa ser concretamente utilizado como ferramenta em situações onde o sistema seria capaz de funcionar diretamente no ambiente industrial, tratando de diagnosticar a situação *online* e abranger a diversidade de diferentes falhas possíveis.

Grande parte dos trabalhos definem as classes de falhas a partir de fragmentos retirados de uma única aquisição de sinal, gerando um único conjunto de dados que representará uma classe. Dessa forma, a classificação pode representar um cenário demasiadamente simples pois não há uma melhor representação amostral de uma determinada falha feita a partir de várias aquisições de diferente situações do equipamento, o que não condiz com os problemas de diagnóstico de falhas encontrados em situações reais. Para que o processo seja confiável e sólido, é imprescindível que o sistema seja capaz

de generalizar bem em condições de uso normal do equipamento, sendo treinado com diferentes representações de uma mesma falha.

2.5 Métodos de Avaliação

Para situações onde a quantidade de amostras disponíveis é limitada, os dados devem ser separados e não sobrepostos. Em um grande número de trabalhos, frequentemente se verifica que os mesmos dados são utilizados duas ou mais vezes para treinar e avaliar o sistema de classificação de falhas e encontrar as melhores hiperparâmetros que irão funcionar para esse sistema. O pesquisador geralmente repete os mesmos ciclos de teste de validação do treinamento, variando os hiperparâmetros envolvidos, até obter o melhor resultados. Como por exemplo, uma rede neural onde a sua arquitetura é alterada em cada rodada de classificação até que a melhor configuração seja alcançada para o mesmo conjunto de dados, e posteriormente, o melhor de todos os resultados é considerado. Esta é uma forma injusta de classificar os dados.

Para se obter uma forma mais justa de alcançar os resultados de desempenho é necessário que se isole completamente um conjunto para teste. Com o restante dos dados pode ser feito o treinamento e o ajuste dos hiperparâmetros, somente após o fim desta etapa o conjunto de teste pode ser utilizado para estimar o valor final de precisão do modelo. Contanto que o conjunto de teste esteja sempre separado dos demais, o procedimento de treino e validação pode ser repetido várias vezes para um melhor resultado, constituindo assim um *loop* interno e um externo.

2.5.1 Resubstituição

Este método é normalmente usado quando se tem poucos dados para treinar e testar um modelo, mas também é a forma de avaliação que possui a maior taxa de erro e os resultados menos confiáveis, pois todo o conjunto de dados é usado para treinar o sistema e, posteriormente, classificar os mesmos dados novamente como ilustrado na figura 7. Todos os dados são utilizados para treino e também para teste, os resultados alcançados dessa forma são super otimistas e os valores de acerto não condizem com a realidade de um ambiente de produção.

2.5.2 Avaliação de divisão única - *Hold-out*

Diferentemente da Resubstituição é uma forma um pouco menos deturpada de abordar o problema é fazer uma divisão única dos dados em duas partes distinta: conjuntos de treinamento e teste. Mesmo assim, os resultados obtidos com o método são insuficientemente confiáveis. Repetidamente se vê situações em que um conjunto de teste não exibe erro algum de classificação embora isso seja probabilisticamente impossível de

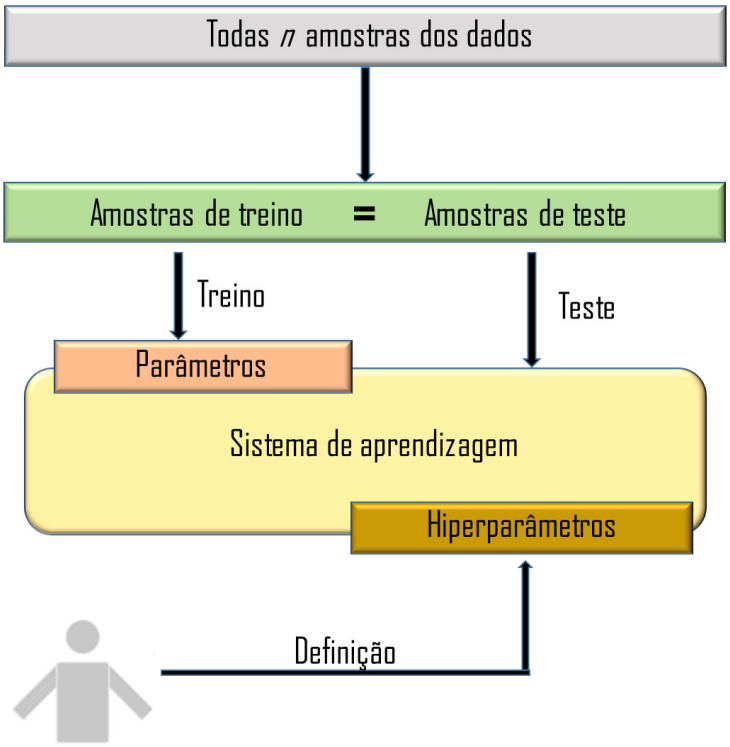


Figura 7 – Método sem divisão de treino e teste para um sistema de diagnóstico de falhas baseado em aprendizado de máquina.

ocorrer. Esse modelo de avaliação é também é denominado de divisão simples de treino e teste e é ilustrado na figura 8.

2.5.3 Amostragem aleatória

Na amostragem aleatória, são criadas K partições do conjunto de todos os documentos. Cada partição é criada selecionando de forma aleatória e sem reposição um número fixo de documentos pra treinamento. São realizados K experimentos e a medida de eficiência é a média das medidas de eficiência obtidas em cada experimento. A amostragem aleatória pode produzir melhores estimativas de erro que o método *holdout*.

2.5.4 *Bootstrap*

O método *Bootstrap* serve para combinar diferentes classificações a partir do treino no mesmo conjunto de dados de entrada, o conjunto de treinamento possui o mesmo tamanho do conjunto de todos os dados e é constituído de dados selecionados aleatoriamente com reposição a partir de tal conjunto. Desta forma, para um mesmo conjunto de treinamento, alguns dados podem não estar incluídos, enquanto outros podem aparecer mais de uma vez. Os dados que não aparecem no conjunto de treinamento são usados como conjunto de teste. Desta forma, são gerados vários subconjuntos diferentes de treinamento a partir de uma amostragem aleatória do conjunto original de dados,

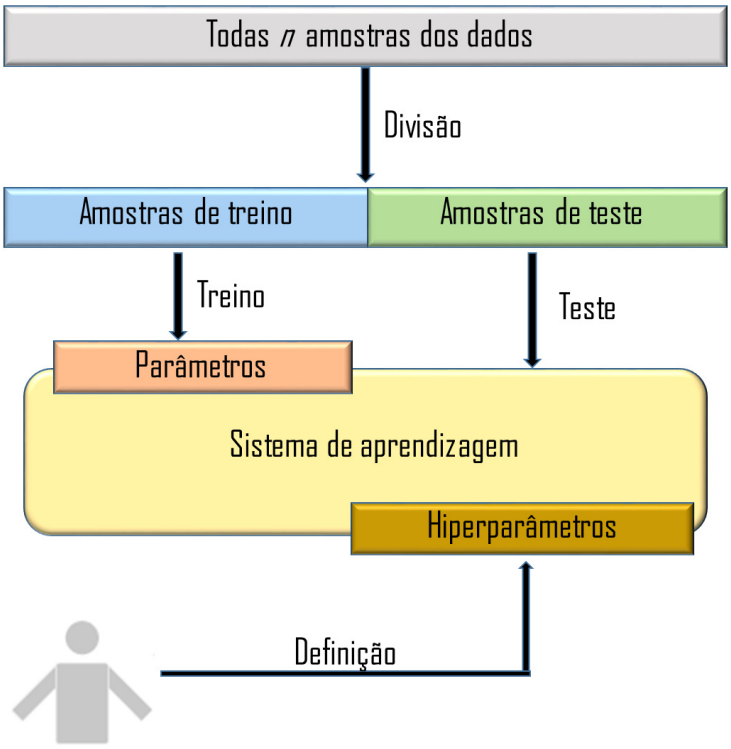


Figura 8 – Divisão simples de treino e teste para um sistema de diagnóstico de falhas baseado em aprendizado de máquina.

com reposição. Então, são induzidos diferentes processos de classificação a partir de cada um destes subconjuntos de treinamento, como representado na Figura 9. Geralmente, o processo de *Bootstrap* é repetido inúmeras vezes, sendo que a medida de eficiência estimada é a média das medidas de eficiência obtidas em cada experimento.

2.5.5 *K-Fold Cross Validation*

Uma técnica muito popular de classificação é o *K-Fold Cross Validation*. Nele, todo o conjunto de dados é dividido em K partes sem que ocorra sobreposição. O ciclo de treinamento e teste é realizado K vezes, cada uma das K partes é deixada de fora do treinamento uma vez, e as partes restantes $(K-1)$ são usadas para treinar o sistema. A parte retida é empregada como o conjunto de teste, e nos resultados dos K ciclos, de avaliação é retirada a média para se obter o resultado final estimativa. A figura 10 apresenta o procedimento para o exemplo de $K = 4$. Se, na avaliação cruzada, o número de divisões for igual ao número de amostras disponíveis, ou seja, $K = n$, o metodo é denominado como validação cruzada *Leave-One-Out*.

Na geração de exemplos, quando um sinal de aquisição é dividido em várias amostras, uma amostra carrega informações da aquisição consigo, não apenas informações de falha. Assim, em um simples *K-fold* a validação cruzada também pode produzir resultados super otimistas pois os classificadores são treinados com praticamente os mesmos exemplos em

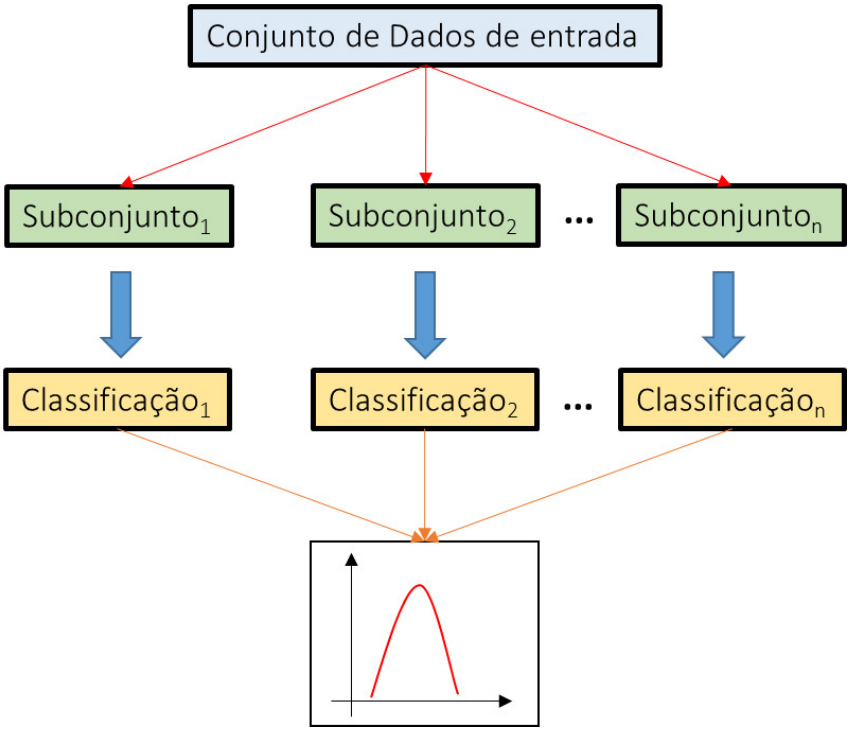


Figura 9 – Método *Bootstrap*.

cada rodada. Uma maneira de obter resultados mais confiáveis é não colocar amostras que se originaram da mesma aquisição no mesmo *fold* . Portanto, os *folds* podem ser divididos por carga para ter um *fold* de validação cruzada onde as amostras de alguma carga são usadas como *fold* de teste enquanto os outros se juntam aos *folds* de treinamento. Essa abordagem de avaliação reduz a influência da aquisição nas amostras. Também possibilita ao sistema uma maior capacidade de adaptação ao domínio do método de aprendizagem, porque as condições das amostras de teste não estão presentes no conjunto de dados de treinamento. Além disso, em processos industriais reais não há garantia de que o conjunto de dados de treinamento possua todas as condições de falhas para o equipamento, assim essa avaliação produziria um *feedback* mais realista do sistema de diagnóstico.

2.5.6 Validação Cruzada Aninhada - *Nested Crosss Validation*

A validação cruzada aninhada possui um CV de *loop* interno aninhado em um CV externo. O *loop* interno é responsável pela seleção do modelo, ou seja, ajuste dos hiperparâmetros (conjunto de validação), enquanto o *loop* externo é utilizado para estimativa de erro (conjunto de teste) conforme figura 11. A validação cruzada aninhada é interessante quando o objetivo da pesquisa é selecionar um entre diferentes algoritmos de *Machine Learning*, resultando em uma boa estimativa do resultado esperado, em termos de erro de predição, para a otimização dos hiperparâmetros de um algoritmo, seguido de sua aplicação em novos conjuntos de dados (RASCHKA, 2015).

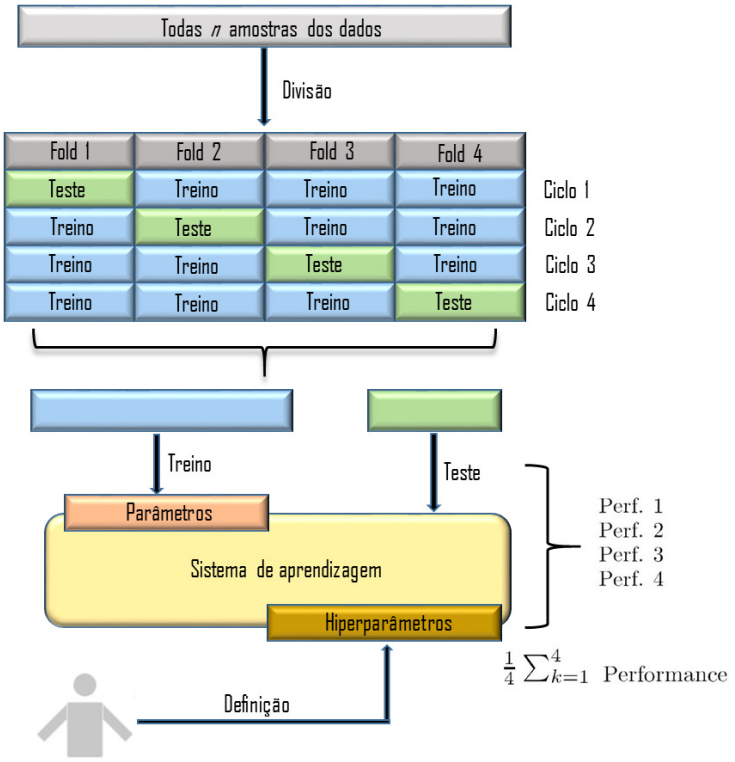


Figura 10 – Validação cruzada *K-fold*.

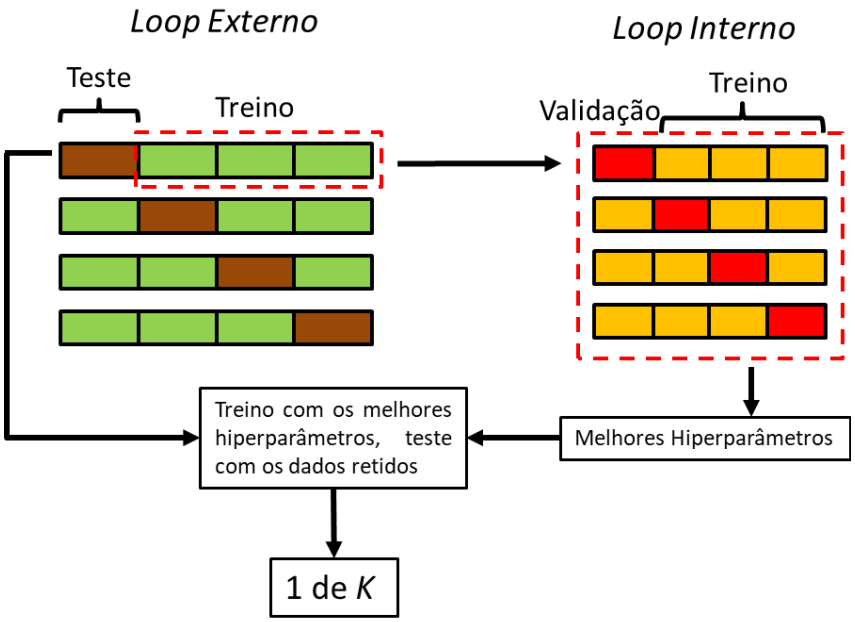


Figura 11 – Validação cruzada aninhada.

2.5.7 Ajuste de hiperparâmetros

Os hiperparâmetros são um ponto crítico para qualquer classificador, eles geralmente são definidos manualmente uma vez e permanecem inalterados durante o procedimento de avaliação. Com isso, o pesquisador geralmente inclina o resultado ao otimismo, experimentando diferentes valores de hiperparâmetro até que melhores valores não sejam visíveis nos

resultados. A partir da variação desses valores, resultados progressivamente mais altos são alcançados e esses números serão relatados como resultado final. Os índices de classificação das amostras não poderiam ser sustentados em situações ondes os dados não fossem previamente classificados a exaustão por um pesquisador. O perigo do ajuste manual de hiperparâmetros é especialmente alto, se apenas poucas amostras estão disponíveis e quando não existe separação de conjuntos de treino, validação e teste. Uma melhor avaliação e desempenho é possível de ser alcançada com o ajuste automático dos hiperparâmetros (CAWLEY; TALBOT, 2010). É importante salientar que, para a obtenção de valores de classificação justos, não existe um melhor conjunto de hiperparâmetros candidatos com poucos valores. Este conjunto só estaria disponível com um número infinitamente grande de elementos de valores candidatos para serem testados o que, consequentemente, demandaria um alto poder computacional.

Na tabela 12 é feito um comparativo dos métodos de avaliação encontrados na revisão de trabalhos correlatos bem como as particularidades e limitações de cada um. Na última linha da tabela é visto as características do método presente neste trabalho que possui correspondência com o método *Nested cross-validation*, reservadas as peculiaridades da proposta do trabalho.

Método/Condições	Divisão do conjunto de dados em treino e teste	Dados de teste não são utilizados no treinamento	Balanceamento entre amostras de treino e teste	Isolamento de conjunto de validação
Resubstituição	✘	✘	✘	✘
Divisão única de Treino e Teste	✓	✓	✘	✘
Validação Cruzada	✓	✓	✓	✘
Validação cruzada *Leave-One-Out*	✓	✓	✓	✘
Validação Cruzada Aninhada	✓	✓	✓	✓
Metodologia do trabalho	✓	✓	✓	✓

Figura 12 – Quadro comparativo entre métodos de validação.

2.6 Bases de dados de sinais de vibração

Diversas bases de dados públicas disponibilizam informações operacionais de equipamentos em condições industriais e servem como fontes para a pesquisa sobre diagnóstico de falhas. Diferentes tipos de equipamentos e arquiteturas são utilizadas para amostragem, a base 3W (VARGAS et al., 2019) é um conjunto de dados com amostras de oito tipos de

eventos anormais no processo de operação de bombas submersas de petróleo, caracterizados por oito variáveis de processo.

O conjunto de dados de rolamentos da *Paderborn University* (BEARING..., 2020) disponibiliza dados de medição síncrona dos sinais de vibração de um motor, permitindo assim a verificação de modelos que incluem múltiplas escalas inerentes e a fusão de sensores de diferentes sinais para aumentar a precisão da detecção de falhas nos rolamentos. Os sinais de vibração do estator são medidos com alta resolução e alta taxa de amostragem, e os dados são obtidos de 26 rolamentos danificados e 6 não danificados (saudáveis).

Já as informações do *dataset* AMES fornecido pela NASA *National Aeronautics and Space Administration* (LEE et al., 2007), consistem de dados de três conjuntos, cada um composto por quatro mancais, nos quais são coletadas informações de vibração que indicam as falha destes em condições de operação. Os dados de vibração são coletados regularmente servindo como indicador da condição do rolamento.

2.6.1 Estudo de caso - Base de dados CWRU

A base de dados é composta por sinais vibratórios gerados a partir de rolamentos em condições normais e defeituosas, extraídos de um motor elétrico de 2 *hp*. As falhas são introduzidas em uma posição específica nas partes que compõe o rolamento, usando usinagem com descarga eletrostática com diâmetros de falha de 0,007, 0,014, 0,021 e 0,028 polegadas. O rolamento consiste em quatro componentes: pista externa (*outer race*), pista interna (*inner race*), e esferas (*ball*) figura 14. Um dinamômetro induz cargas de 0, 1, 2 e 3 *hp*, alterando a velocidade de rotação do eixo do motor de 1797 para 1720 rpm. O rolamento modelo (6205-2RS JEM SKF) é utilizado na extremidade que está voltada para o dinamômetro e o de modelo (6203-2RS JEM SKF) é usado na extremidade do motor voltada para a ventoinha. Três acelerômetros são colocados na extremidade do dinamômetro, na extremidade do ventilador e na base do motor desta forma, são coletados os sinais de vibração do sistema conforme mostrado na figura 13. Os dados coletados estão disponíveis na forma de arquivos Matlab (*.mat).

2.6.2 Descrição dos dados CWRU

Na literatura, encontra-se uma vasta aplicação de técnicas de ciência de dados em processos de diagnóstico sem modelo e muitos resultados de pesquisas realizadas para classificar diferentes estados nos dados disponíveis. O *Case Western Reserve University* (CWRU) é uma base de dados que contém informações de sinais que representam diferentes fontes de vibração provenientes de um sistema mecânico composto por um motor e um dinamômetro e que serve de modelo para diagnóstico (CASE..., 2014). Os dados obtidos são divididos por diferentes cargas aplicadas ao motor, em diferentes falhas aplicadas

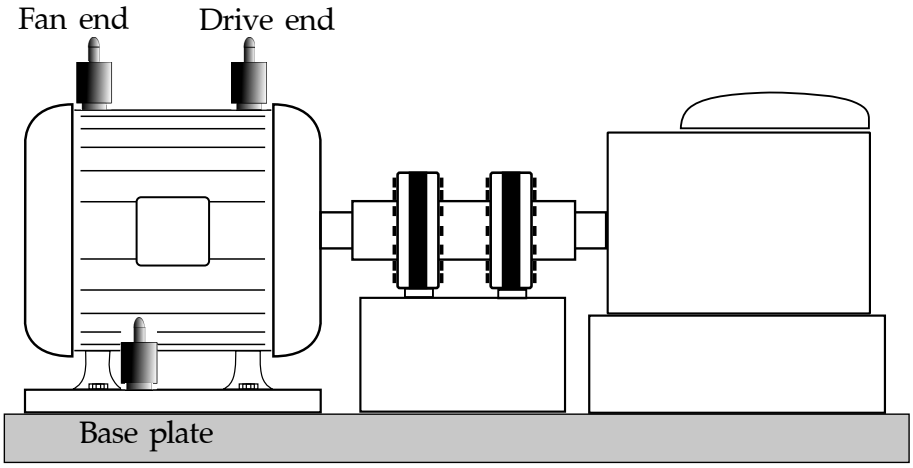


Figura 13 – Base de teste CWRU para diagnóstico de falhas nos rolamentos.

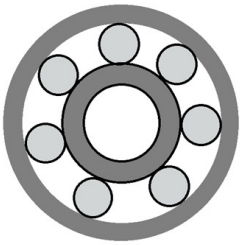


Figura 14 – Exemplo de rolamento do utilizado nos experimentos.

no sistema, na severidade das falhas e nas diferentes posições onde elas podem estar localizadas no conjunto mecânico.

Esses dados são fornecidos de forma pública e são largamente utilizados como base para pesquisa científica no contexto de detecção e diagnóstico de falhas. A disponibilidade dos dados do CWRU é destinada a possibilitar que a comunidade de pesquisadores teste seus próprios métodos de classificação e diagnóstico de falhas a fim de alcançar sistemas sofisticados que sejam capazes de realizar o diagnóstico mais preciso possível das condições da máquina. Entretanto, junto ao conjunto de dados não é fornecida nenhuma metodologia previamente descrita de como os experimentos de classificação devam ser realizados, o que favorece que haja brecha para uma grande quantidade de diferentes interpretações sobre como utilizar o conjunto.

Os arquivos do CWRU contêm uma única aquisição de sinal, na maioria dos casos, tanto para a extremidade do ventilador quanto para a extremidade do motor. Este sinal é fundamentalmente uma única amostra de uma determinada classe de condição da máquina. Para a condição normal, há quatro arquivos uma para cada nível de carga, para as demais apenas um. A maneira como esse arquivo único é transformado em um conjunto de muitos padrões é uma questão fundamental que influencia fortemente o desempenho de um sistema de diagnóstico.

Como pode ser visto nas tabelas 15, 16 e 17, nem todos os arquivos são dispo-

nibilizados para cada modo de aquisição. Por exemplo, os arquivos com os defeitos de severidade 0.028 polegada não estão disponíveis para as aquisições de *Fan End* e de *Drive End* com 48000 amostragens por segundo e mesmo para a aquisição de *Drive End* com 12000 amostragens por segundo, as falhas *Outer Race* não estão disponíveis para este valor de severidade. Falhas de *Outer Race* também são ausentes na aquisição *Fan End* para o níveis 1,2 e 3 de carga nas severidade de 0.014 e 0.021 polegadas.

Severidade	Carga HP	*Inner Race*	*Outer Race*	*Ball*
0.007 "	0	✓	✓	✓
	1	✓	✓	✓
	2	✓	✓	✓
	3	✓	✓	✓
0.014 "	0	✓	✓	✓
	1	✓	✓	✓
	2	✓	✓	✓
	3	✓	✓	✓
0.021 "	0	✓	✓	✓
	1	✓	✓	✓
	2	✓	✓	✓
	3	✓	✓	✓
0.028 "	0	✓	×	✓
	1	✓	×	✓
	2	✓	×	✓
	3	✓	×	✓

Figura 15 – Arquivos CWRU de aquisição de *Drive End* com 12000 amostragens por segundo.

2.6.3 Uso da base CWRU no diagnóstico de falhas

Cada arquivo que faz parte do *dataset* contém dados gerados por um, dois ou três acelerômetros com duração aproximada de dez segundos. Para um classificador baseado no paradigma de aprendizado supervisionado é necessário que haja a divisão de conjuntos de treinamento e de teste, mas para o CWRU não existe recomendações a respeito de como realizar o procedimento fazendo a divisão do sinal bruto como ele é fornecido. Algumas questões são peça chave para a relevância de um modelo de classificação que faça utilização dos dados e essas se referem a como é construída a metodologia para segmentação dos dados como: o número de padrões que podem ser obtidos a partir de todo o sinal fazendo sua divisão em várias partes, se esses segmentos podem ou não estar sobrepostos, qual o comprimento mínimo de um segmento para que não haja degradação

Severidade	Carga HP	*Inner Race*	*Outer Race*	*Ball*
0.007 "	0	✓	✓	✓
	1	✓	✓	✓
	2	✓	✓	✓
	3	✓	✓	✓
0.014 "	0	✓	✓	✓
	1	✓	✓	✓
	2	✓	✓	✓
	3	✓	✓	✓
0.021 "	0	✓	✓	✓
	1	✓	✓	✓
	2	✓	✓	✓
	3	✓	✓	✓
0.028 "	0	×	×	×
	1	×	×	×
	2	×	×	×
	3	×	×	×

Figura 16 – Arquivos CWRU de aquisição de *Drive End* com 48000 amostragens por segundo.

das informações contidas, fazendo com que os resultados gerados não sejam mais confiáveis e como o conjunto de dados pode ser dividido em diferentes conjuntos para evitar resultados tendenciosos.

Severidade	Carga HP	*Inner Race*	*Outer Race*	*Ball*
0.007 "	0	✓	✓	✓
	1	✓	✓	✓
	2	✓	✓	✓
	3	✓	✓	✓
0.014 "	0	✓	✓	✓
	1	✓	×	✓
	2	✓	×	✓
	3	✓	×	✓
0.021 "	0	✓	✓	✓
	1	✓	×	✓
	2	✓	×	✓
	3	✓	×	✓
0.028 "	0	×	×	×
	1	×	×	×
	2	×	×	×
	3	×	×	×

Figura 17 – Arquivos CWRU de aquisição de *Fan End*.

3 Revisão Bibliográfica

As tabelas 1 e 2 relacionam os resultados da análise crítica feita em uma compilação de publicações científicas representativas. A partir das informações colhidas sobre a metodologia seguida pelos autores, são apresentas as pontuações quantitativas de desempenho para o CWRU, destacando suas características e desvantagens. Muitos dos resultados obtidos são questionáveis, pois empregam uma metodologia simplista de divisão única de treino e teste, reutilizam os mesmos dados em vários estágios do aprendizado do classificador ou apresentam métodos irreprodutíveis e, portanto, não verificáveis. São descritas as desvantagens conceituais e depois é feita uma classificação completa dos trabalhos.

3.1 Origem e formato dos dados

Na tabela 1, é listada uma compilação de publicações que utilizaram os dados do CWRU para testar propostas de diagnóstico de falha. Os trabalhos são categorizados preliminarmente em relação ao número de amostras extraídas do arquivo, se a extração foi feita com partes sobrepostas do sinal Quais as posicoes dos acelerômetros foram utilizadas.

Como mencionado anteriormente, cada condição da máquina é caracterizada por um único arquivo. Se mais de uma amostra é necessária para a abordagem do modelo de aprendizado de máquina, a divisão deste sinal único deve ser bem definida. Desta forma é importante saber como os intervalos de tempo são fixados. Também é importante saber se houve sobreposição durante essa divisão, já que sobrepor o sinal implicitamente significa usar o conhecimento mais do que uma vez. Isso também pode ser uma fonte de super otimismo. Finalmente, alguns trabalhos usam apenas os dados do acelerômetro do *drive-end*(DE), outros usam também o sinal obtido do *fan-end* (FE).

3.2 Metodologia da revisão

A tabela 2 caracteriza os trabalhos da tabela 1 com relação a metodologia de experimentação empregada. Diferentes aspecto importantes são considerados como aspectos de avaliação e como foi feita a divisão dos dados em treinamento e teste.

O número de amostras de treinamento está diretamente relacionado com o intervalo de confiança das estimativas de desempenho de uma classificação. Dentre os trabalhos analisados, é variada a quantidade de amostras extraídas dos arquivos de dados do CWRU. (EREN; INCE; KIRANYAZ, 2019) utilizou a menor quantidade de amostras, 200. (ZHANG et al., 2017; ZHANG et al., 2018; HOANG; KANG, 2019) utilizaram 400 amostras por

arquivo, (RAUBER; BOLDT; VAREJÃO, 2014; LEI et al., 2016; YANG; FU; HE, 2018; BERREDJEM; BENIDIR, 2018; GUO; CHEN; SHEN, 2016; LI; FANG; HUANG, 2015) utilizam entre 1000 e 1200 amostras, sendo este o número médio entre todos os trabalhos. Alguns trabalhos fazem uso de uma maior quantidade de amostras com (RODRIGUEZ et al., 2013; YU et al., 2015; WU et al., 2013; WU et al., 2012; LI et al., 2019) que reportam o uso de 2048 amostras enquanto (WEN et al., 2017) e (SREEJITH; VERMA; SRIVIDYA, 2008) usam respectivamente 4096 e 6000 amostras por arquivo de dados. (PAN et al., 2017; BOLDT; RAUBER; VAREJAO, 2013; BOLDT et al., 2015; KAVATHEKAR; UPADHYAY; KANKAR, 2016; SHEN et al., 2013; PILTAN et al., 2019; HAN; JIANG, 2016; ZHAO et al., 2014; YIAKOPOULOS; GRYLLIAS; ANTONIADIS, 2011; ZHAO et al., 2011) e (YU, 2011) não informam a quantidade de amostras utilizadas.

Outro fator importante para o enviesamento dos resultados de desempenho do método que é a sobreposição de amostras durante a extração, (RAUBER; BOLDT; VAREJÃO, 2014; RAZAVI-FAR; FARAJZADEH-ZANJANI; SAIF, 2017; LI et al., 2019; YANG; FU; HE, 2018; WU et al., 2012; HOANG; KANG, 2019; BERREDJEM; BENIDIR, 2018; SREEJITH; VERMA; SRIVIDYA, 2008; ZHANG et al., 2018) coletaram amostras sem sobreposição do sinal, já (LEI et al., 2016; ZHANG et al., 2017; WEN et al., 2017; RODRIGUEZ et al., 2013) utilizaram algum valor de sobreposição das amostras extraídas enquanto (WU et al., 2013; YU et al., 2015; PAN et al., 2017; EREN; INCE; KIRANYAZ, 2019; BOLDT et al., 2015; KAVATHEKAR; UPADHYAY; KANKAR, 2016; BOLDT; RAUBER; VAREJAO, 2013; SHEN et al., 2013; GUO; CHEN; SHEN, 2016; PILTAN et al., 2019; HAN; JIANG, 2016; LI; FANG; HUANG, 2015; ZHAO et al., 2014; YIAKOPOULOS; GRYLLIAS; ANTONIADIS, 2011; ZHAO et al., 2011; YU, 2011) não informam se os dados foram coletados com sobreposição ou não.

Foi analisado quais posições de aquisição de dados os trabalhos utilizaram na sua metodologia, se foi utilizado dados de *Drive end*, *Fan end* ou ambos. (RAUBER; BOLDT; VAREJÃO, 2014; RAZAVI-FAR; FARAJZADEH-ZANJANI; SAIF, 2017; LI et al., 2019; YANG; FU; HE, 2018; WU et al., 2012; BOLDT et al., 2015; BOLDT; RAUBER; VAREJAO, 2013) utilizam *Drive end* e *Fan end* como fontes. Apenas (SHEN et al., 2013) utiliza apenas *Fan end*. A grande maioria do trabalhos como (LEI et al., 2016; ZHANG et al., 2017; WEN et al., 2017; WU et al., 2013; HOANG; KANG, 2019; YU et al., 2015; PAN et al., 2017; BERREDJEM; BENIDIR, 2018; EREN; INCE; KIRANYAZ, 2019; KAVATHEKAR; UPADHYAY; KANKAR, 2016; BERREDJEM; BENIDIR, 2018; SREEJITH; VERMA; SRIVIDYA, 2008; RODRIGUEZ et al., 2013; GUO; CHEN; SHEN, 2016; PILTAN et al., 2019; ZHANG et al., 2018; LI; FANG; HUANG, 2015; ZHAO et al., 2014; YIAKOPOULOS; GRYLLIAS; ANTONIADIS, 2011; YU, 2011) usam apenas *Drive end* para seu método. (HAN; JIANG, 2016; ZHAO et al., 2011) não informam qual posição dos acelerômetros foi usada para obtenção das amostras.

Em relação ao método de validação utilizada para estimar o desempenho do sistema de diagnóstico foi identificado que a grande maioria dos trabalhos como (LEI et al., 2016; ZHANG et al., 2017; WEN et al., 2017; LI et al., 2019; YANG; FU; HE, 2018; WU et al., 2012; WU et al., 2013; HOANG; KANG, 2019; BERREDJEM; BENIDIR, 2018; SREEJITH; VERMA; SRIVIDYA, 2008; RODRIGUEZ et al., 2013; SHEN et al., 2013; HAN; JIANG, 2016; ZHANG et al., 2018; LI; FANG; HUANG, 2015; ZHAO et al., 2014; ZHAO et al., 2011; YU, 2011) utiliza a técnica de *hodl-out*, (RAUBER; BOLDT; VAREJÃO, 2014; YU et al., 2015; EREN; INCE; KIRANYAZ, 2019; GUO; CHEN; SHEN, 2016) Utilizaram *Cross Validation*, (RAZAVI-FAR; FARAJZADEH-ZANJANI; SAIF, 2017) foi o único que utilizou *Nested Cross Validation* para avaliação, (BOLDT et al., 2015) empregou *Leave-One-out*, enquanto (PAN et al., 2017; KAVATHEKAR; UPADHYAY; KANKAR, 2016; PILTAN et al., 2019; YIAKOPOULOS; GRYLLIAS; ANTONIADIS, 2011) fazem uso de resubstituição.

O número rodadas que foram executadas para calcular o desempenho final do método foi analisado e o mais frequente encontrado em (RAUBER; BOLDT; VAREJÃO, 2014; LEI et al., 2016; ZHANG et al., 2017; RAZAVI-FAR; FARAJZADEH-ZANJANI; SAIF, 2017; LI et al., 2019; YANG; FU; HE, 2018; HOANG; KANG, 2019; YU et al., 2015; PAN et al., 2017; BERREDJEM; BENIDIR, 2018; EREN; INCE; KIRANYAZ, 2019; BOLDT et al., 2015; KAVATHEKAR; UPADHYAY; KANKAR, 2016; BOLDT; RAUBER; VAREJAO, 2013; SHEN et al., 2013; GUO; CHEN; SHEN, 2016; PILTAN et al., 2019; HAN; JIANG, 2016; LI; FANG; HUANG, 2015; YIAKOPOULOS; GRYLLIAS; ANTONIADIS, 2011; YU, 2011) foi a realização de apenas uma rodada de avaliação para obter o valor de desempenho definitivo, à excessão, (WEN et al., 2017; WU et al., 2012; WU et al., 2013; RODRIGUEZ et al., 2013; ZHANG et al., 2018; ZHAO et al., 2014; ZHAO et al., 2011) fizeram uso de várias rodadas de treinamento e testes antes de alcançar o valor final de classificação que foi obtido pelas médias de todas as rodadas.

Outra dimensão pontuada é como foram utilizados os hiperparâmetro nos classificadores que compõe o método proposto, é verificado se foi realizado o ajuste manual ou automático dos hiperparâmetros, ou se o ajuste não foi feito. (RAUBER; BOLDT; VAREJÃO, 2014; LEI et al., 2016; WEN et al., 2017; LI et al., 2019; YANG; FU; HE, 2018; WU et al., 2012; WU et al., 2013; YU et al., 2015; BERREDJEM; BENIDIR, 2018; PAN et al., 2017; EREN; INCE; KIRANYAZ, 2019; BOLDT et al., 2015; KAVATHEKAR; UPADHYAY; KANKAR, 2016; SREEJITH; VERMA; SRIVIDYA, 2008; BOLDT; RAUBER; VAREJAO, 2013; SHEN et al., 2013; GUO; CHEN; SHEN, 2016; HAN; JIANG, 2016; ZHANG et al., 2018; LI; FANG; HUANG, 2015; ZHAO et al., 2014; YIAKOPOULOS; GRYLLIAS; ANTONIADIS, 2011; YU, 2011) Não utilizaram nenhum tipo de ajuste de hiperparâmetros, (ZHANG et al., 2017; HOANG; KANG, 2019; PILTAN et al., 2019) ajustaram hiperparâmetros manualmente, e somente (RAZAVI-FAR; FARAJZADEH-ZANJANI; SAIF, 2017; RODRIGUEZ et al., 2013; ZHAO et al., 2011) fizeram o ajuste

automático dos valores.

Diferentes quantidade de condições da máquina (classes) foram consideradas nas obras, a maior parte, como neste trabalho, os artigos (WU et al., 2012; HOANG; KANG, 2019; PAN et al., 2017; BOLDT et al., 2015; KAVATHEKAR; UPADHYAY; KANKAR, 2016; SREEJITH; VERMA; SRIVIDYA, 2008; RODRIGUEZ et al., 2013; SHEN et al., 2013; GUO; CHEN; SHEN, 2016; PILTAN et al., 2019; HAN; JIANG, 2016; LI; FANG; HUANG, 2015; ZHAO et al., 2014; YIAKOPOULOS; GRYLLIAS; ANTONIADIS, 2011; YU, 2011) consideram 4 classes para classificação, (ZHAO et al., 2011) usa 10 classes, (LEI et al., 2016) 11 classes, (ZHANG et al., 2017; LI et al., 2019; YU et al., 2015; EREN; INCE; KIRANYAZ, 2019) classificam 6 diferentes classes, (RAZAVI-FAR; FARAJZADEH-ZANJANI; SAIF, 2017; WEN et al., 2017; BERREDJEM; BENIDIR, 2018; ZHANG et al., 2018) classificam 7 classes, (YANG; FU; HE, 2018; WU et al., 2013) 8 classes, (RAUBER; BOLDT; VAREJÃO, 2014) identifica 19 classes para o trabalho e (BOLDT; RAUBER; VAREJAO, 2013) 21 classes.

Quando avaliado se a mesma fonte de dados (aquisição) foi utilizada para o aprendizado, estimativa de desempenho e também para o ajuste dos hiperparâmetros, todos os trabalhos que realizaram o ajuste, com exceção de (RODRIGUEZ et al., 2013), utilizaram fontes iguais para o treino, avaliação do método e para o ajuste dos hiperparâmetros.

Praticamente todas as obras relatam altos valores de desempenho em seus métodos de avaliação. A Acurácia é a única métrica utilizada para mensurar demonstrar os resultados. (LI et al., 2019; SREEJITH; VERMA; SRIVIDYA, 2008; SHEN et al., 2013; PILTAN et al., 2019; ZHANG et al., 2018; LI; FANG; HUANG, 2015; YIAKOPOULOS; GRYLLIAS; ANTONIADIS, 2011) Informam 100% de acerto de classificação, (RAUBER; BOLDT; VAREJÃO, 2014; LEI et al., 2016; WEN et al., 2017; YANG; FU; HE, 2018; WU et al., 2012; HOANG; KANG, 2019; YU et al., 2015; PAN et al., 2017; EREN; INCE; KIRANYAZ, 2019; BOLDT; RAUBER; VAREJAO, 2013; HAN; JIANG, 2016) reportam Acurácia de 99%, (WU et al., 2013; BOLDT et al., 2015) mostra ter obtido 98%, (GUO; CHEN; SHEN, 2016; ZHAO et al., 2014) 97% de acerto, (BERREDJEM; BENIDIR, 2018; ZHAO et al., 2011) relata o valor 96%, (ZHANG et al., 2017) apenas 84% e por fim, (KAVATHEKAR; UPADHYAY; KANKAR, 2016) e (RODRIGUEZ et al., 2013) divulgam o valor de 90% de Acurácia.

Nas figuras 18 e 19 é exibida a síntese quantitativa das informações obtidas nas publicações relativas a todos os aspectos relevantes da pesquisa. Os eixos do gráfico, comum a todas a dimensões, representa o valor máximo do numero amostral de publicações analisadas (29). Em cada característica, a amplitude do valor do eixo representa a frequência com que esse atributo é identificado na revisão do trabalho.

Tabela 1 – Artigos da CWRU classificados em relação ao número de amostras de dados, sobreposição de amostras e posição dos acelerômetros.

Artigo	Amostras por arquivo de dados	Sobreposição	Posição dos acelerômetros
(RAUBER; BOLDT; VAREJÃO, 2014)	1200	Ausente	DE - FE
(LEI et al., 2016)	1200	Presente	DE
(ZHANG et al., 2017)	400	Presente	DE
(RAZAVI-FAR; FARAJZADEH-ZANJANI; SAIF, 2017)	1024	Ausente	DE - FE
(WEN et al., 2017)	4096	Presente	DE
(LI et al., 2019)	2048	Ausente	DE - FE
(YANG; FU; HE, 2018)	1024	Ausente	DE - FE
(WU et al., 2012)	2048	Ausente	DE - FE
(WU et al., 2013)	2048	Não infor.	DE
(HOANG; KANG, 2019)	400	Ausente	DE
(YU et al., 2015)	2000	Não infor.	DE
(PAN et al., 2017)	Não infor.	Não infor.	DE
(BERREDJEM; BENIDIR, 2018)	1024	Ausente	DE
(EREN; INCE; KIRANYAZ, 2019)	200	Não infor.	DE
(BOLDT et al., 2015)	Não infor.	Não infor.	DE - FE
(KAVATHEKAR; UPADHYAY; KANKAR, 2016)	Não infor.	Não infor.	DE
(SREEJITH; VERMA; SRIVIDYA, 2008)	6000	Ausente	DE
(BOLDT; RAUBER; VAREJAO, 2013)	Não infor.	Não infor.	DE - FE
(RODRIGUEZ et al., 2013)	2040	Presente	DE
(SHEN et al., 2013)	Não infor.	Não infor.	FE
(GUO; CHEN; SHEN, 2016)	1024	Não infor.	DE
(PILTAN et al., 2019)	Não infor.	Não infor.	DE
(HAN; JIANG, 2016)	Não infor.	Não infor.	Não infor.
(ZHANG et al., 2018)	1000	Ausente	DE
(LI; FANG; HUANG, 2015)	1024	Não infor.	DE
(ZHAO et al., 2014)	Não infor.	Não infor.	DE
(YIAKOPOULOS; GRYLLIAS; ANTONIADIS, 2011)	Não infor.	Não infor.	DE
(ZHAO et al., 2011)	Não infor.	Não infor.	Não infor.
(YU, 2011)	Não infor.	Não infor.	DE

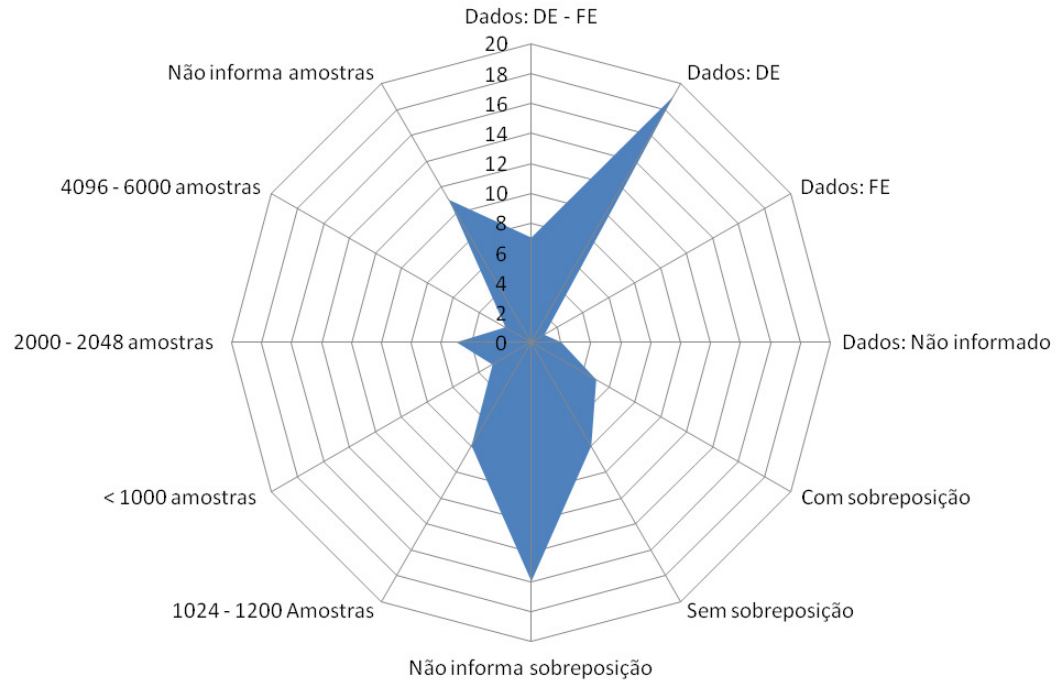


Figura 18 – Amostragem.

Tabela 2 – Artigos relacionados CWRU - Metodologia de experimentação.

Artigo	Método de avaliação	Rodadas	Ajuste	Treino	Teste	Fonte Treino-Teste	Classes	Acurácia
(RAUBER; BOLDT; VAREJÃO, 2014)	*Cross validation*	Única	Inexistente	2295	19	Igual	19	99.97
(LEI et al., 2016)	Hold-out	Única	Inexistente	20000	2000	Igual	11	99.66
(ZHANG et al., 2017)	Hold-out	Única	Manual	4832	1208	Igual	6	84.20
(RAZAVI-FAR; FARAJZADEH-ZANJANI; SAIF, 2017)	*Nested CV*	Única	Automatico	3060	306	Igual	7	Não Inf.
(WEN et al., 2017)	Hold-out	Várias	Inexistente	2000	400	Igual	7	99.79
(LI et al., 2019)	Hold-out	Única	Inexistente	340	340	Igual	6	100
(YANG; FU; HE, 2018)	Hold-out	Única	Inexistente	2090	522	Igual	8	99.77
(WU et al., 2012)	Hold-out	Várias	Inexistente	318,636,954,1272	2862,25440,2226,1908	Igual	4	99.55
(WU et al., 2013)	Hold-out	Várias	Inexistente	16	64	Igual	8	98.81
(HOANG; KANG, 2019)	Hold-out	Única	Manual	2024	400	Igual	4	99.83
(YU et al., 2015)	*Cross validation*	Única	Inexistente	220	440	Igual	6	99.64
(PAN et al., 2017)	Resubstituição	Única	Inexistente	800	-	Igual	4	99.63
(BERREDJEM; BENIDIR, 2018)	Hold-out	Única	Inexistente	504	504	Igual	7	96.08
(EREN; INCE; KIRANYAZ, 2019)	*Cross validation*	Única	Inexistente	3630	363	Igual	6	99.93
(BOLDT et al., 2015)	*Leave-One-out*	Única	Inexistente	129	1	Igual	4	98.27
(KAVATHEKAR; UPADHYAY; KANKAR, 2016)	Resubstituição	Única	Inexistente	55	-	Igual	4	73.21
(SREEJITH; VERMA; SRIVIDYA, 2008)	Hold-out	Única	Inexistente	12	8	Igual	4	100
(BOLDT; RAUBER; VAREJAO, 2013)	*Leave-one-out*	Única	Inexistente	2414	1	Igual	21	99.96
(RODRIGUEZ et al., 2013)	Hold-out	Várias	Automático	30	10	Diferente	4	90.00
(SHEN et al., 2013)	Hold-out	Única	Inexistente	120	240	Igual	4	100
(GUO; CHEN; SHEN, 2016)	*Cross validation*	Única	Inexistente	1000	100	Igual	4	97.90
(PILTAN et al., 2019)	Resubstituição	Única	Manual	Ausente	Ausente	Igual	4	100
(HAN; JIANG, 2016)	Hold-out	Única	Inexistente	80,160,240,320	320,240,160,80	Igual	4	99.29
(ZHANG et al., 2018)	Hold-out	Várias	Inexistente	80	120	Igual	7	100
(LI; FANG; HUANG, 2015)	Hold-out	Única	Inexistente	500	500	Igual	4	100
(ZHAO et al., 2014)	Hold-out	Várias	Inexistente	5,10,15,20	75,70,65,60	Igual	4	97.65
(YIAKOPOULOS; GRYLLIAS; ANTONIADIS, 2011)	Resubstituição	Única	Inexistente	8192	-	Igual	4	100
(ZHAO et al., 2011)	Hold-out	Várias	Automático	760	160	Igual	10	96
(YU, 2011)	Hold-out	Única	Inexistente	150	146	Igual	4	Não Inf.

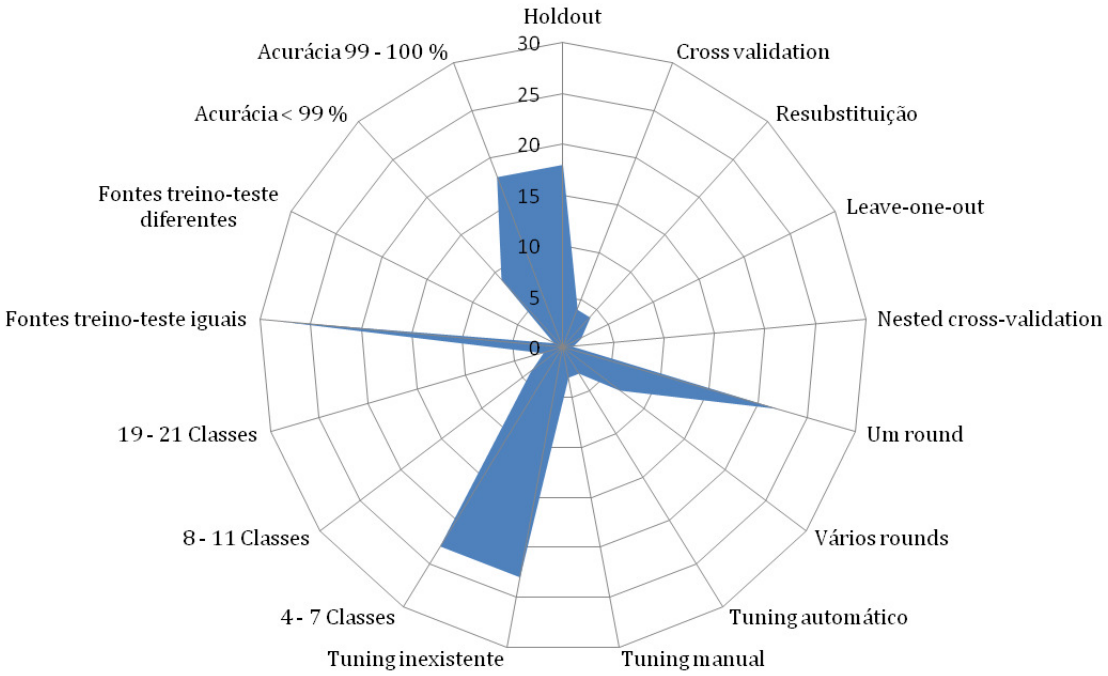


Figura 19 – Metodologia.

4 Método de Avaliação Proposto

A grande maioria das publicações, analisadas e compiladas na tabela 2, usa métodos clássicos de Validação Cruzada (CV) para estimar o desempenho do diagnóstico de falhas do sistema proposto. A acurácia é utilizada como principal critério de desempenho do classificador. Os dados disponíveis são divididos em partes de treinamento e teste, com a parte de treinamento, os parâmetros do classificador são aprendidos e a parte de teste é usada para obter os resultados da classificação. Na maioria dos casos, os hiperparâmetros do sistema são definidos manualmente *a priori*.

4.1 Estimativa de hiperparâmetros

Classificadores possuem como parte significativamente crítica para sua performance a determinação dos seus hiperparâmetros. O número K de vizinhos em um classificador KNN, ou sua métrica de distância (geralmente euclidiana) deve ser previamente definida. O tipo de *kernel* e seus valores de parâmetros associados de um classificador SVM são outro exemplo, assim como o número de árvores de um classificador RF. Isso se torna um problema quando os esses valores são ajustados manualmente com base na mesma validação cruzada dos dados disponíveis. Frequentemente, a escolha de hiperparâmetros é feita reciclando continuamente os mesmos dados que são usados para estimar o desempenho. Isso fornece resultados fortemente super otimistas. Essa prática errônea induz a uma classificação pretensiosa e super ajustada ao conjunto de dados. Um problema semelhante é frequentemente observado com o uso dos valores padrão dos modelos classificadores de ferramentas de software, por exemplo, utilizar um classificador da biblioteca Python *scikit-learn*, e posteriormente apresentar os valores de pontuação de desempenho sem mencionar que valores de hiperparâmetros estavam associados ao classificador empregado.

4.2 Diferença entre *Nested Cross Validation* e *Cross Validation*

O trabalho de (FUKUNAGA, 1990) de reconhecimento estatístico de padrões utilizando conjuntos de dados gaussianos multivariados sintéticos permite estabelecer o limite Bayesiano teórico (DUDA; HART; STORK, 2012), além do qual os resultados são certamente tendenciosos ao super otimismo. Os conjuntos de dados gerados podem ser usados como um validador. Isso significa que um pesquisador pode utilizar esses conjuntos de dados em seu algoritmo de validação de desempenho como, por exemplo, *K-Fold cross validation*. Se uma aplicação em particular no diagnóstico de falhas tem n amostras de dados, exatamente essa quantidade de n amostras aleatórias podem ser geradas

Tabela 3 – Parâmetros de conjuntos de dados gaussianos de (FUKUNAGA, 1990), e *"null" dataset* de (VARMA; SIMON, 2006).

Dataset	Classe	Vetor Médio μ	Matriz de Covariância Σ	Bayes Acurácia
I-I	C_a C_b	$[0,0,0,0,0,0,0,0]$ $[2.56,0,0,0,0,0,0,0]$	$diag(1,1,1,1,1,1,1,1)$ $diag(1,1,1,1,1,1,1,1)$	90.0%
I-4I	C_a C_b	$[0,0,0,0,0,0,0,0]$ $[0,0,0,0,0,0,0,0]$	$diag(1,1,1,1,1,1,1,1)$ $diag(4,4,4,4,4,4,4,4)$	$\approx$ 91.0%
I-Λ	C_a C_b	$[0,0,0,0,0,0,0,0]$ $[3.86,3.10,0.84,0.84,1.64,1.08,0.26,0.01]$	$diag(1,1,1,1,1,1,1,1)$ $diag(8.41,12.06,0.12,0.22,1.49,1.77,0.35,2.73)$	$\approx$ 98.1%
null	C_a C_b	$[0,0,0,0,0,0,0,0]$ $[0,0,0,0,0,0,0,0]$	$diag(1,1,1,1,1,1,1,1)$ $diag(1,1,1,1,1,1,1,1)$	50.0%

para os conjuntos de dados de *Fukunaga* e, em seguida, o procedimento de estimativa de desempenho proposto revelaria ser um resultado irrealista *a priori* se os valores de desempenho forem melhores que o limite bayesiano esperado, em particular, inferior a taxa de erro de Bayes.

4.2.1 Conjuntos de dados Gaussianos Multivariados Sintéticos

Três conjuntos de dados de 8 variáveis Gaussianas são definidas. Duas classes C_a e C_b que caracterizam-se exclusivamente por seus vetores médios μ_a, μ_b e matrizes de covariância Σ_a, Σ_b, respectivamente. Não há covariância, ou seja, os elementos fora da diagonal σ_{ij}, $i,j = 1,\ldots,8$ em ambos Σ_a e Σ_b são zero. Isso é justificável, porque quaisquer duas matrizes de covariância não diagonais podem ser simultaneamente diagonalizados por uma transformação linear. Além disso, um vetor médio diferente de zero pode ser deslocado para a origem sem prejuízo para o caso geral que é definido por média diferente de zero vetores e covariâncias diferentes de zero (FUKUNAGA, 1990). Portanto, apenas 32 parâmetros devem ser especificados para definir as duas funções de densidade de probabilidade, dois vetores médios de 8 dimensões e 16 variações nas duas diagonais das matrizes de covariância. A probabilidade *a priori* idênticaP$(C_a) = P(C_b) = 50\%$ de cada classe é simulado por um número igual de n amostras para cada classe. Para garantir a reprodutibilidade de qualquer pesquisador, por uma geração aleatória de n amostras para cada classe, ou seja, um total de $n * 2$ amostras, a semente do gerador de números aleatórios foi ajustada para 66649. A classe Python `numpy.random` com método `numpy.random.nomal` foi usado para gerar as amostras. A tabela 3 mostra a especificação de cada um dos três conjuntos de dados definidos por Fukunaga. Além disso, o assim chamado *"null" dataset* (VARMA; SIMON, 2006) está incluído, onde os dois gaussianos tem zero médias e variações de unidade. Portanto, as duas classes se sobrepõem perfeitamente, o que dá origem a um erro de classificação de 50% .

4.2.2 Experimento para estimativa de desempenho

Os conjuntos de dados da tabela 3 são submetidos a experimentos de estimativa de desempenho para investigar especialmente a influência do número n de amostras e o efeito de uma validação cruzada aninhada. Embora possam ser usadas métricas de desempenho mais sofisticadas, foi utilizada a medida de acurácia do classificador nas duas classes. O objetivo desses testes é sugerir que poucas amostras de treinamento no diagnóstico de falhas são um problema grave e que uma validação cruzada aninhada pode pelo menos fornecer uma ideia menos tendenciosa, em comparação com técnicas simples de CV. Uma das principais hipóteses deste trabalho é mostrar que uma validação cruzada aninhada é sempre melhor que a validação cruzada convencional.

4.2.3 Comparação experimental de validação cruzada simples e aninhada

Nas experiências relatadas a seguir, duas técnicas de CV são comparadas. A primeira é o algoritmo de validação cruzada aninhada proposto para este trabalho. O segundo é o método CV convencional. A validação cruzada *k-fold* é feita no da seguinte maneira: Para todas as combinações possíveis dos valores de hiperparâmetros, que também são usados na fase de ajuste do modelo, é realizada uma validação cruzada convencional *k-fold*. Apenas o melhor resultado é relatado, ou seja, desse conjunto de hiperparâmetros, apenas os que foram responsáveis pelo maior valor de desempenho. Isso simula o comportamento de um ser humano que tenta repetidamente empurrar o resultado em direção à otimização variando os valores da hiperparâmetros, usando a mesma divisão de treino e teste várias vezes. A figura 20 sobrepõe os *boxplots* da CV simples e aninhado para um classificador *Support Vector Machine*, em uma validação cruzada *k-fold* com $K = 10$, $L = 9$ e $n = 2 * 25 = 50$ amostras no conjunto de dados "I_4I". Um total de 20 combinações diferentes de hiperparâmetros são pesquisados, variando o par (C, γ), que são utilizados para o ajuste de hiperparâmetros. O CV simples que é chamado *"Biased"* na figura, ultrapassa com valor de 92.50% o limite teórico de acurácia de Bayes 91%. O CV aninhado é chamado "Nested" na figura, permanece com 87.83% abaixo do limite. Isso mostra que a abordagem aninhada, em geral, pode exibir melhores valores teoricamente possíveis, especialmente para um pequeno número de amostras. No próximo experimento, o número n de amostras disponíveis é analisada. Devido ao alto custo computacional, para o CNN-1D neste modelo, o número de rodadas e *folds* foi limitado a três e cinco, respectivamente. Caso contrário, para os dados sintéticos, os valores são baseadas em $R = 10$ *rounds*, $K = 10$ *folds*, e $L = 9$ *folds* ajustados.

Para o classificador KNN, o número máximo de vizinhos foi limitado a cinco, pois para um pequeno número de amostras, o número de vizinhos não pode ser superior ao número de amostras de treinamento, daí a número de vizinhos é selecionado no conjunto $\{1, 3, 5\}$. A figura 21 mostra a evolução da acurácia estimada para os quatro classificadores

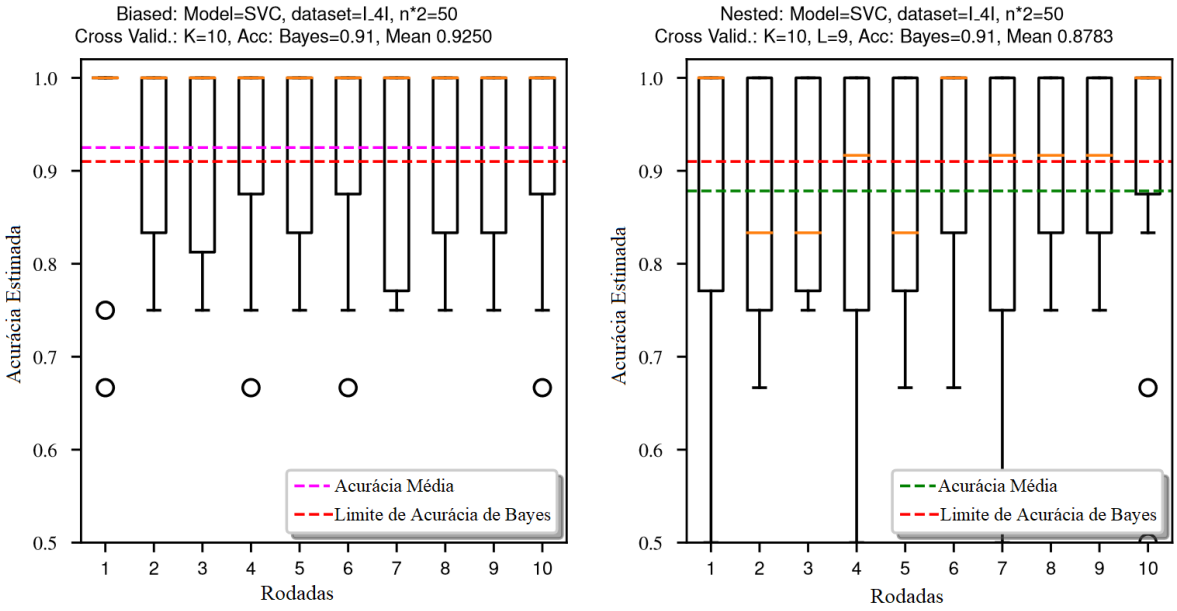
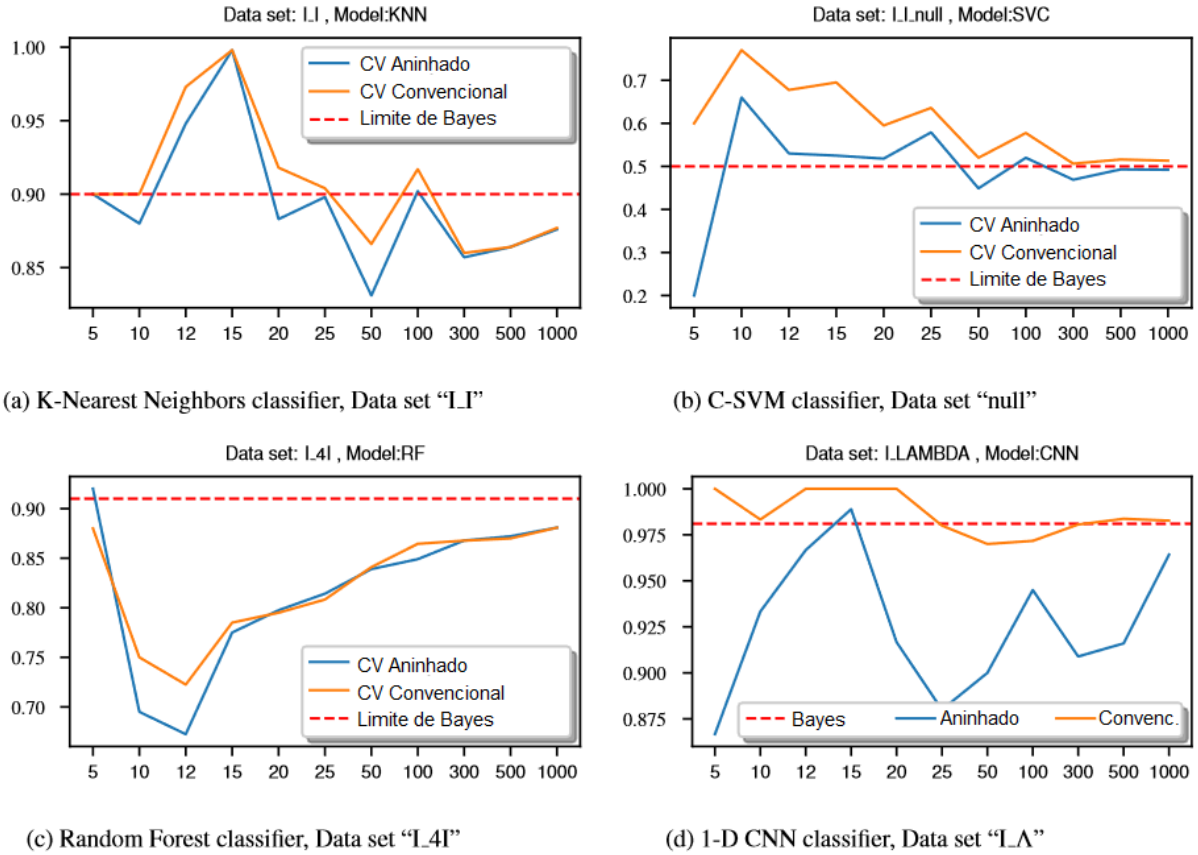


Figura 20 – Comparação de um simples *K-fold* e uma validação cruzada aninhada. Dois hiperparâmetros são ajustados, C e γ do SVM.



(a) K-Nearest Neighbors classifier, Data set "I_I"

(b) C-SVM classifier, Data set "null"

(c) Random Forest classifier, Data set "I_4I"

(d) 1-D CNN classifier, Data set "I_A"

Figura 21 – Comparação de um simples *K-fold* e uma validação cruzada aninhada para diferentes arquiteturas de classificadores e conjunto de dados. O eixo x é número de amostras por classe e o eixo y é a acurácia estimada.

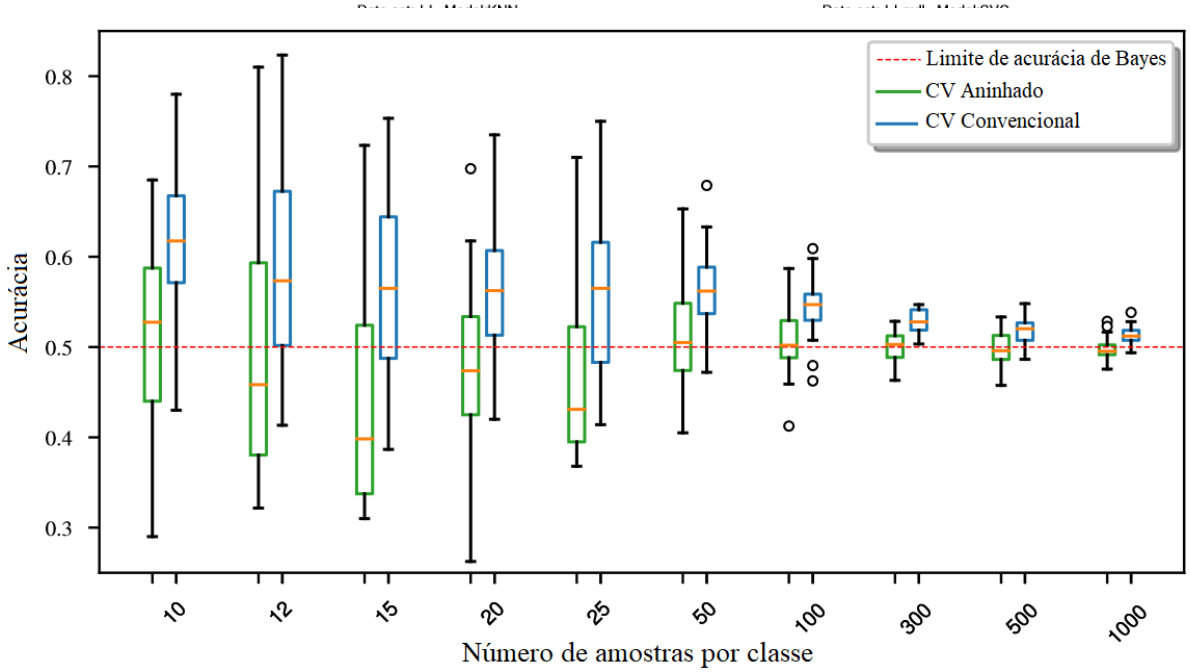


Figura 22 – *Boxplots* para validação cruzada aninha comparada com validação *K-fold*. O classificador *Support Vector Machine* com os dados "null" é repetidamente submetido ao *framework*, usando 30 diferentes sementes.

diferentes e os quatro diferentes conjuntos de dados da tabela 3. As duas abordagens de validação de desempenho, aninhados e convencionais são comparados entre si, aumentando gradualmente o número n de amostras geradas aleatoriamente. Quatro exemplos típicos são representados na figura.

Os valores de desempenho desta experiência sugerem:

1. Para um número pequeno n de amostras em uma tarefa de aprendizado supervisionado, a estimativa de desempenho é quase sempre super otimista, independente do método de validação cruzada;

2. Para um grande número de amostras, técnicas de validação cruzada simples e mais sofisticadas apresentam assintoticamente os mesmos resultados;

3. Para um pequeno número de amostras, a validação cruzada aninhada fornece valores de desempenho mais realistas, pois ultrapassa o limite de erro de Bayes com menos frequência que as técnicas convencionais de validação cruzada, como *k-Fold*.

O experimento é repetido com sementes diferentes, a fim de comparar estatisticamente as abordagens CV convencional e aninhada. Um total de 30 sementes é gerado. As 30 precisões estimadas são usadas para gerar um par de *boxplots* para cada um dos possíveis números de amostras, exceto cinco amostras que apresentaram uma variação muito alta. Cada par compara a avaliação aninhada com o convencional *k-fold cross validation*. Pode-se observar que as acurácias estimadas se aproximam da acurácia teórica

de Bayes 50% quando o número de amostras é aumentado. Simultaneamente, com a variação do valor de 30, os valores diminuem, mas o viés super otimista da validação cruzada convencional é claramente distinguível da estimativa mais realista da validação cruzada aninhada como mostrado na figura 22.

Para tirar conclusões finais do estudo de simulação, foram realizados dois conjuntos de testes estatísticos. No primeiro conjunto de testes, a versão tradicional do par *t-test* Student (CASELLA; BERGER, 2002) é empregado para comparar o desempenho médio das duas abordagens CV para cada tamanho de amostra. O segundo conjunto de testes investiga o viés das metodologias de validação cruzada empregando a amostra *t-test* para comparar os resultados de cada método com a acurácia teórica de Bayes (50% nesse caso). Os resultados confirmam os achados do parágrafo anterior. De fato, o primeiro conjunto de testes é altamente significativo com o máximo *p-value* valendo $1.36e-13$, que foi alcançado para o teste com os resultados da amostra de tamanho igual a 12 ($n = 12$), significando que a validação cruzada aninhada é uma abordagem mais confiável para relatar os desempenhos. Em posse dessa conclusão, pode-se verificar o viés super otimista do CV convencional aplicando o segundo conjunto de testes. Por exemplo, para $n = 2000$ o *t-test* leva a conclusão de que a acurácia média do CV convencional é maior que a precisão teórica de Bayes (*p-value* of 6.19e-08), considerando que o mesmo teste não rejeite a hipótese de que a precisão média da validação cruzada aninhada é a taxa de erro teórica de Bayes ($p\text{-}value = 0.257$).

4.2.4 Reprodutibilidade

Informações ausentes ou até contraditórias para descrever o experimento tornam impossível para um pesquisador comparar os resultados de um artigo ao seu próprio trabalho, ou verificar os valores de desempenho apresentados. É fortemente desejável que um bom trabalho forneça uma descrição minuciosa da estrutura e metodologia experimental empregada, como publicar as sementes de geradores de números aleatórios, para permitir a qualquer pesquisador da comunidade verificar a veracidade dos valores de desempenho obtidos de acordo com o critério e utilizar o modelo para comparação e aperfeiçoamento do seu próprio experimento.

Devido a perceptível fragilidade das metodologias encontradas na literatura e a necessidade de métodos mais realistas para o problema do CWRU, é proposto neste trabalho a comparação sistemática de desempenho dos modelos mais usuais das publicações e da metodologia aqui proposta. A ideia deste trabalho não é a proposta de mais um modelo de extração de características ou arquitetura de classificador para melhorar quase 100% da métrica avaliação.

Os resultados dos experimentos de classificação são relatados com objetivo principal de avaliar a capacidades de generalização do modelo, aos classificadores são submetidos

Tabela 4 – Acurácia estimada[%] em função do número n de amostras por classe. Os valores acima do limite teórico de Bayes estão sublinhadas.

Model	Dados	*Cross Validation*	n por classe (= 2 ∗ n amostras) 5	10	12	15	20	25	50	100	300	500	1000
KNN	I_I	NESTED	90.00	88.00	94.75	99.75	88.25	89.83	83.10	90.15	85.72	86.35	87.61
		BIASED	90.00	90.00	97.25	99.75	91.75	90.42	86.60	91.65	85.97	86.43	87.67
	I_4I	NESTED	60.00	61.50	65.75	75.50	61.50	67.17	59.70	71.00	76.60	76.35	81.19
		BIASED	60.00	61.50	65.75	75.50	63.00	68.58	62.30	71.00	76.83	76.74	81.31
	I_Λ	NESTED	90.00	100.0	96.50	100.0	99.50	90.50	91.70	97.35	95.60	96.34	96.75
		BIASED	90.00	100.0	98.50	100.0	99.75	92.33	92.60	97.90	95.95	96.57	96.84
	null	NESTED	50.00	55.00	65.75	68.50	62.00	57.58	46.60	50.40	48.98	49.19	49.05
		BIASED	50.00	61.00	71.25	75.50	63.75	62.00	49.70	52.75	50.12	50.14	49.89
SVM	I_I	NESTED	0.00	86.50	91.00	94.00	83.25	88.33	89.60	92.50	88.25	87.83	89.53
		BIASED	100.0	88.00	95.75	96.50	88.25	91.83	90.10	93.90	88.90	88.46	90.01
	I_4I	NESTED	100.0	94.00	87.75	93.50	89.75	87.83	92.40	90.20	92.08	90.07	90.29
		BIASED	100.0	95.00	90.75	94.25	91.50	92.50	92.40	91.35	92.17	90.33	90.47
	I_Λ	NESTED	90.00	99.00	93.00	97.50	97.75	92.67	92.70	97.15	96.08	96.75	97.59
		BIASED	100.0	100.0	100.0	100.0	98.25	94.25	94.80	97.95	96.67	97.17	97.73
	null	NESTED	20.00	66.00	53.00	52.50	51.75	57.92	44.90	51.95	46.90	49.30	49.22
		BIASED	60.00	77.00	67.75	69.50	59.50	63.58	52.00	57.75	50.63	51.60	51.32
RF	I_I	NESTED	90.00	81.00	88.50	97.00	89.25	86.83	88.80	91.15	87.10	87.13	89.49
		BIASED	93.00	79.00	87.75	96.00	88.25	87.25	89.50	90.90	87.75	87.34	89.48
	I_4I	NESTED	92.00	69.50	67.25	77.50	79.75	81.42	83.90	84.90	86.78	87.21	88.08
		BIASED	88.00	75.00	72.25	78.50	79.50	80.83	84.10	86.45	86.77	86.99	88.08
	I_Λ	NESTED	80.00	92.00	95.00	94.00	92.50	90.08	93.20	95.80	96.65	97.24	97.57
		BIASED	83.00	90.00	93.00	96.25	93.75	90.58	92.10	95.80	96.88	97.22	97.45
	null	NESTED	88.00	48.50	56.00	59.25	51.50	50.67	48.60	52.00	50.77	47.98	50.14
		BIASED	84.00	49.00	57.75	63.50	49.50	52.50	48.30	51.80	51.28	48.52	49.82
CNN	I_I	NESTED	73.33	63.33	86.11	77.78	70.00	74.67	83.67	87.17	87.22	84.90	88.73
		BIASED	100.0	91.67	97.78	96.67	79.17	78.67	93.67	94.50	91.22	93.53	92.80
	I_4I	NESTED	60.00	51.67	62.78	83.33	70.00	76.00	79.00	70.50	77.72	78.83	80.27
		BIASED	100.0	98.33	78.33	88.89	92.50	88.00	81.67	77.50	79.50	79.27	70.08
	I_Λ	NESTED	86.67	93.33	96.67	98.89	91.67	88.00	90.00	94.50	90.89	91.60	96.42
		BIASED	100.0	98.33	100.0	100.0	100.0	98.00	97.00	97.17	98.06	98.37	98.27
	null	NESTED	43.33	40.00	53.89	47.78	48.33	54.67	52.67	53.00	50.28	49.70	49.43
		BIASED	90.00	70.00	95.56	83.33	75.00	75.33	74.00	79.00	61.17	60.10	60.88

dados para identificação que, mesmo representando as mesmas condições de falhas, não estiveram presentes durante o treinamento. O trabalho visa mostrar qualitativamente que os valores de alto desempenho relatados na maioria das publicações não são realistas para um ambiente de diagnóstico de falhas no mundo real.

4.2.5 Viés de similaridade

O principal objetivo deste trabalho é propor um método que evite o que aqui é chamado de viés de semelhança ou similaridade. Este viés surge através da extração de vários padrões de arquivos que contêm uma única aquisição de sinal, esses padrões geralmente são muito semelhantes. Se um experimento de aprendizado de máquina usa padrões extraídos do mesmo arquivo para treinar e testar os classificadores, a tarefa de classificação se torna relativamente fácil, se comparando ao exercício de consulta de padrões

idênticos em uma tabela de banco de dados, semelhante a um processo de memorização. Este trabalho denomina esse problema como um problema de viés de similaridade. Para evitar este viés, as metodologias empregadas devem garantir que na divisão dos *folds* para treinamento e teste não haja possibilidade que os padrões extraídos de um arquivo estejam presentes em mais de um *fold*. Provavelmente, existem diferentes maneiras de distribuir os padrões nos *folds*, de maneira a evitar o viés de similaridade. Portanto, a divisão deve ser cuidadosamente projetada, tentando manter os *folds* estratificados e com tamanho equivalente. Para contornar o viés de similaridade no processo de ajuste dos hiperparâmetros, recomenda-se que a divisão de *folds* usada no *loop* externo também seja mantida no *loop* interno e que o valor de L seja igual a $K-1$ no algoritmo 1.

4.3 Validação Cruzada *K-fold* com *loop* externo de teste e *loop* interno de validação

Esta abordagem consiste na utilização de validação cruzada aninhada *nested cross validation*, em que um processo de validação cruzada interno é realizado com o objetivo de estimar os melhores hiperparâmetros e selecionar modelos, e um processo de validação cruzada externo é realizado para avaliar o erro de predição dos modelos selecionados. O modelo é selecionado em cada conjunto de treinamento externo, usando o *loop* interno do CV, e seu desempenho é medido no conjunto de teste externo correspondente.

A Figura 23 apresenta a estrutura do modelo de avaliação de desempenho aninhada proposto neste trabalho, nele, um conjunto de teste é retido de maneira análoga a uma validação cruzada *K-fold* normal, entretanto, o diferencial está na existência de um *loop* interno que utiliza os *folds* da divisão dos dados para treinamento $(K-1)$, que ao contrário do *Kfold* normal, que utiliza todos os *folds* para treinamento, separa um *fold* para servir como conjunto de validação. De uma maneira geral, o conjunto de dados do *loop* interno pode ser dividido em qualquer número $1 < L < n_{inner}$ de *folds*, em que n_{inner} é o número de amostras passadas para o *loop* interno. Para o presente trabalho $L = K-1$ e a divisão dos *folds* é realizada apenas uma vez. O conjunto de validação é usado continuamente para fazer a estimativa sobre a qualidade dos hiperparâmetros atuais.

Em todo o processo, o conjunto de teste do *loop* externo é mantido inalterado pelo *loop* interno, o *loop* interno utiliza apenas os dados de treinamento e os dados de validação. Se uma melhoria nos resultados da classificação foi alcançada pela variação dos hiperparâmetros, esse conjunto de hiperparâmetros é preservado para produzir o resultado do K-ésimo *fold* da parte externa ciclo. Uma vez definidos os melhores hiperparâmetros nesse *loop* interno, a pontuação final de desempenho do K-ésimo *fold* é realizada no conjunto de teste deixado de fora no *loop* externo. Após a definição dos melhores valores dos hiperparâmetros, o classificador é treinado usando esses valores e todo o conjunto de

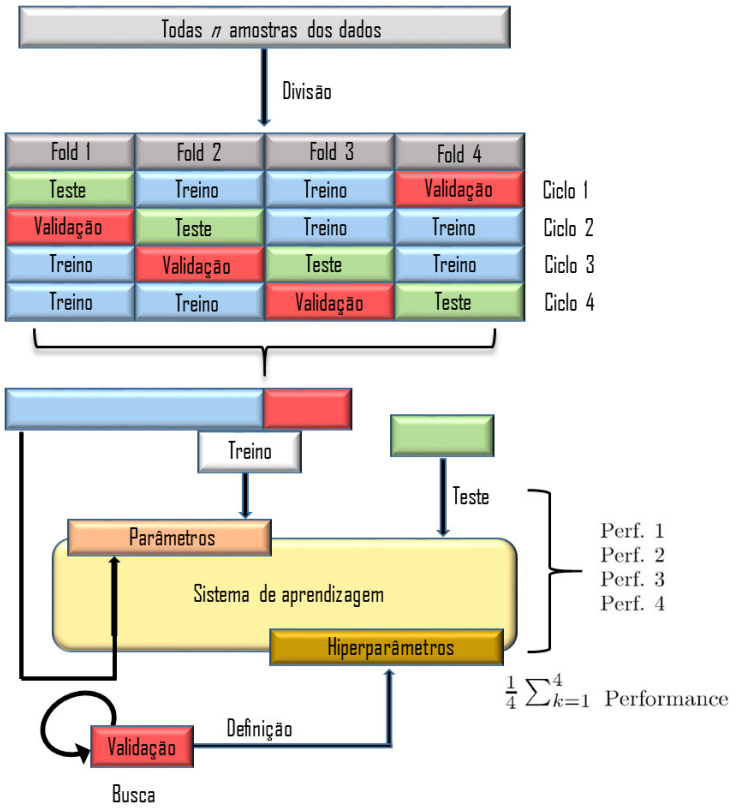


Figura 23 – Validação cruzada aninhada com *loops* internos e externos com K =4 *folds*.

treinamento (incluindo os exemplos de validação).

Para cada um dos K *folds* do *loop* externo, geralmente se obtém diferentes conjuntos de melhores hiperparâmetros. Consequentemente, a avaliação aninhada é incapaz de fornecer uma definição final do conjunto de melhores hiperparâmetros para todo o conjunto de dados. Considerando o exemplo de um classificador *K-Nearest Neighbors* (COVER; HART, 1967) e supondo que todos os outros hiperparâmetros sejam fixos como a métrica da distância (euclidiana, etc.), o único hiperparâmetro restante é o número K de vizinhos. Supondo ainda que apenas um ou três mais vizinhos próximos são considerados, o hiperparâmetro pode assumir os valores $K = 1$ e $K = 3$. Considerando uma divisão ilustrativa mínima dos dados em apenas quatro *folds*, indicados como $Fold1$, $Fold2$, $Fold3$ e $Fold4$. Portanto, existem quatro ciclos do loop externo. No primeiro ciclo do *loop* externo, $Fold1$ é armazenado como o *fold* de teste. $Fold2$, $Fold3$ e $Fold4$ são passados para os *loops* internos. Destacando que, em geral, os dados passados para os *loops* internos podem ser divididos em novas partes arbitrárias, mas a divisão de *folds* herdada do *loop* externo não é alterada.

Como foi definido que no *loop* interno há um *fold* a menos do que no *loop* externo, temos três *folds* para três ciclos internos. Os hiperparâmetros são ajustados utilizando $Fold2$, $Fold3$ e $Fold4$. No primeiro ciclo interno, a união de $Fold2$ e $Fold3$ é usada para treinamento e $Fold4$ é usado para teste. No segundo ciclo interno, a união dos conjuntos

$Fold2$ e $Fold4$ é usada para treinamento e $Fold3$ é usada para teste. No terceiro ciclo interno, a união de $Fold3$ e $Fold4$ é usado para treinamento e $Fold2$ é usado para teste. O desempenho médio destes três ciclos internos é calculado para cada possível combinação dos hiperparâmetros, neste caso para os diferentes números possíveis 1, 3 de vizinhos do classificador *K-Nearest Neighbors*. Considerando que $K = 1$ mostrou o melhor resultado, consequentemente, o hiperparâmetro está fixado em $K = 1$. Desta forma, o primeiro ciclo externo está pronto para calcular o desempenho. Com o melhor hiperparâmetro, o conjunto de treinamento é a união de $Fold2$, $Fold3$ e $Fold4$. O *K-fold* de teste é $Fold1$, que não teve participação na escolha do melhor hiperparâmetro. A performance do classificador então é gravada.

Para os três ciclos externos restantes, o procedimento é repetido e o desempenho final estimado é o valor médio de todas as quatro classificações do *loop* externo. No segundo ciclo externo, $Fold2$ é o conjunto de testes reservado e $Fold1$, $Fold3$, $Fold4$ são os conjuntos passados para o *loop* interno. O melhor hiperparâmetro avaliado pode ser diferente do primeiro ciclo externo. No terceiro ciclo externo, $Fold3$ é o conjunto de testes reservado e $Fold1$, $Fold2$, $Fold4$ são utilizados para o *loop* interno. No quarto ciclo externo, $Fold4$ é reservado para conjunto de teste e $Fold1$, $Fold2$, $Fold3$ são passados para o *loop* interno. Em todos os casos o *loop* interno funciona da mesma maneira explicada para o primeiro ciclo externo.

4.3.1 Avaliação aninhada repetida

A ideia de ajustar os hiperparâmetros e usar um teste independente em cada ciclo da validação cruzada *K-fold* fornece K resultados, um para cada ciclo do *loop* externo. Na forma convencional do *K-fold CV*, esses resultados são combinados tomando a média dos *folds*.

Neste trabalho essa ideia é levada adiante, introduzindo um nível adicional de generalização à avaliação aninhada, repetindo o experimento por várias rodadas. O *framework* de avaliação de desempenho resultante está descrito no algoritmo 1, extraído de (OLIVEIRA-SANTOS et al., 2018). Os seguintes símbolos são usados:

$\mathcal{C}$	Classificador
$\mathcal{D}$	Conjunto total de dados com n padrões
$\mathcal{T}$	Conjunto de treino
$\mathcal{V}$	Conjunto de teste
$\mathcal{P}$	Conjunto de hiperparâmetros candidatos
$\mathcal{P}^*$	Conjunto de de melhores hiperparâmetros
R	Número de rounds
K	Número de *folds* do *loop* CV externo para estimativa de desempenho geral
$f_{r,k}$	k-ésimo *fold* do r-ésimo *round*
PM	$R \times K$ Matriz de desempenho
L	Número de *folds* do *loop* CV interno para ajuste
crit	Critério de desempenho, p. ex. *accuracy*, *precision*, *recall*, *F-measure*, ROC-AUC, ...

Os valores de desempenho obtidos dos R rounds e K *folds* são armazenados em uma matriz $R \times K$ que é pós-processada para obter o resultado final da avaliação. A abordagem mais simples é tomar a média de todas as pontuações. Isso é feito neste trabalho. Também é possível processar a matriz de maneira que inclua a realização de teste de hipótese (OLIVEIRA-SANTOS et al., 2016). Normalmente, até três resultados diferentes são obtidos para o conjunto de melhores hiperparâmetros. Isso é expresso no algoritmo 1 pelo fato que a função *Tuning* retorna o melhor $P * k$ no k-ésimo de um total de K *folds* do *loop* externo. De qualquer forma, o resultado global é muito menos tendencioso do que uma técnica de validação convencional das figuras 8 e 10.

O algoritmo recebe como entrada n padrões rotulados D de c classes, R números de rodadas, K números de *folds* para o *loop* externo e L *folds* para o *loop* interno. Para cada uma das rodada R, são gerados K *folds* estratificados e para cada *loop* externo um

Algoritmo 1: *Framework* de avaliação de desempenho.

Function $perf \leftarrow$ *Modelo_performance(*$\mathcal{D}$*,R,K)*
Input: n padrões rotulados $\mathcal{D}$ de c classes, R números de *rounds*, números de folds: K (*loop* externo), L (*loop* interno)
Output: Para cada *round* $r \leftarrow 1, \ldots, R$ gera K *folds* estratificados. Um *fold* é fixado como teste, o restante dos *folds* é utilizado para treino e ajuste. Calcular o critério de desempenho para cada combinação de *round* and *fold* e pôr na Matriz de desempenho PM ($R \times K$). Processar PM, por exemplo a média final de desempenho.
for $r = 1$ **to** R **do** // todos *rounds*
// gerar K *folds* estratificados
$f_{r,k}, k = 1, \ldots, K$;
for $k = 1$ **to** K **do** // todos *folds*
// conjunto de treino do k-ésimo *fold* do r-ésimo *round*
$\mathcal{T} \leftarrow \{f_{r,1} \cup \ldots \cup f_{r,K}\} \setminus f_{r,k}$;
// conjunto de teste do k-ésimo *fold* do r-ésimo *round*
$\mathcal{V} \leftarrow f_{r,k}$;
$\mathcal{P}_k^* \leftarrow$ Ajuste ($\mathcal{T}$, L);
$\mathcal{C} \leftarrow$ classificador_treino ($\mathcal{T}, \mathcal{P}_k^*$);
$crit \leftarrow$ classificador_teste ($\mathcal{C}, \mathcal{V}$);
PM(r,k)$\leftarrow$ *crit*
end
end
$perf \leftarrow \overline{PM}$;
end

Function $\mathcal{P}^* \leftarrow$ *Tuning(*$\tilde{\mathcal{D}}$*,L)*
Input: Data set $\tilde{\mathcal{D}}$, número de *folds* ajustados L
Output: Conjunto de melhores hiperparâmetros $\mathcal{P}^*$ do modelo de classificação
inicializa o melhor critério $maxcrit \leftarrow 0$;
// gera L *folds* estratificados
$\tilde{f}_k, k = 1, \ldots, L$;
repeat // *grid search* para o conjunto de melhores hiperparâmetros $\mathcal{P}^*$
gera conjunto de hiperparâmetros candidatos $\mathcal{P}$;
for $k = 1$ **to** L **do** // todos *folds*
// conjunto de treino do k-ésimo *fold*
$\mathcal{T} \leftarrow \left\{\tilde{f}_1 \cup \ldots \cup \tilde{f}_L\right\} \setminus \tilde{f}_k$;
// conjunto de teste do k-ésimo *fold*
$\mathcal{V} \leftarrow \tilde{f}_k$;
$\mathcal{C} \leftarrow$ classificador_treino ($\mathcal{T}, \mathcal{P}$);
$crit_k \leftarrow$ classificador_teste ($\mathcal{C}, \mathcal{V}$);
end
$\overline{crit} \leftarrow mean(crit_k)$;
if $\overline{crit} > maxcrit$ **then**
$maxcrit \leftarrow \overline{crit}$; $\mathcal{P}^* \leftarrow \mathcal{P}$
end
until *busca completa*;
end

fold é isolado para servir como teste, o restante dos *folds* são utilizados para o treio e para o ajuste dos melhores hiperparâmetros pelo *loop* interno. No *loop* interno, o número $K-1$ de *folds* é utilizado na função *grid search* que repetirá o *loop* de aprendizado e validação até que todas as combinações dos conjuntos de hiperparâmetro candidatos tenham sido satisfeitas. Os resultados de classificação com o teste no *loop* externo são armazenados em uma matrix $R * K$ e posteriormente, é feita a média de todas as rodadas para estimação do valor final de desempenho da classificação.

4.3.2 Modos experimentais

Os dados utilizados nas experiências deste trabalho foram coletados a uma taxa de 12.000 amostras por segundo. As falhas de *Outer Race* também podem indicar a zona de carga, portanto, existem aquisições para ambas as posições, *Drive End* e *Fan End*, com falhas *Outer Race* localizadas às 3 horas (diretamente na zona de carga), às 6 horas (ortogonais à zona de carga) e às 12 horas. As classes identificadas podem ser rotuladas de acordo com a localização do rolamento, o estado do rolamento (normal ou defeituoso), a localização da falha (quando o rolamento está com defeito), a gravidade da falha (profundidade) e a carga do motor. Para falhas *outer race*, também é possível identificar a zona de carga. Geralmente, os trabalhos encontrados na literatura identificam até dez condições. Em (RAUBER; BOLDT; VAREJÃO, 2014) 19 condições são identificadas. O presente trabalho identifica apenas quatro condições distintas.

Cada condição do sistema no banco de dados CWRU é representada por um ou mais arquivos.

Para cada estado do rolamento, local da falha, gravidade da falha, carga do motor e zona de carga, existe apenas um arquivo de aquisição. Essa é uma desvantagem quando o conjunto de dados CWRU é utilizado em um sistema de aprendizado de máquina, porque a abordagem de aprendizado supervisionado necessita de uma quantidade considerável de amostras de treinamento e teste para fornecer resultados imparciais. A grande maioria dos trabalhos encontrados na literatura superam esse problema dividindo os sinais de vibração dos arquivos do CWRU em muitos blocos, sobrepondo-se frequentemente a partes idênticas do sinal. A técnica de avaliação mais comumente usada no CWRU é a validação cruzada *K-fold* . Quando um sinal de aquisição é dividido em várias amostras, cada amostra carrega informações da própria aquisição como padrões cíclicos de repetição do sinal, não apenas informações de falha. Assim, uma simples validação cruzada *K-fold* pode produzir resultados super otimistas.

Uma maneira de obter resultados mais confiáveis é não colocar amostras originárias da mesma aquisição no mesmo *fold*. Por exemplo, os *folds* podem ser divididos por carga para ter uma validação cruzada de 4 vezes, onde as amostras de alguma carga são usadas como *folds* de teste enquanto outras são unidas nos *folds* de treinamento. Essa abordagem

de avaliação reduz a influência da aquisição nas amostras. Ele também verifica a capacidade do método de aprendizado de adaptação a condições diversas de dianóstico, porque as amostras de teste representam uma condição operacional do equipamento que não está presente no conjunto de dados de treinamento. Além disso, em processos industriais reais, não há garantia de que o conjunto de dados de treinamento contenha exemplos para todos as condições operacionais do equipamento como carga aplicada ou severidade da falha, desta forma, essa avaliação produziria um *feedback* mais realista do sistema de diagnóstico.

4.3.3 Estimativa de Desempenho

O conjunto de dados CWRU é submetido ao experimento de estimativa de desempenho possibilitando investigar o efeito de uma validação cruzada aninhada. O objetivo dos testes é sugerir que a utilização de amostras de treinamento e teste de uma mesma aquisição em diagnóstico de falha é um problema grave e que uma validação cruzada aninhada pode pelo menos fornecer valores menos tendenciosos, em comparação com técnicas simples de CV. Uma das principais hipóteses que este trabalho propõe é mostrar que uma validação cruzada aninhada, embora propenso a estimativas de desempenho super otimistas, é sempre melhor que a validação cruzada convencional.

Uma função discriminatória quadrática é o classificador ideal, pois modela exatamente as densidades gaussianas (DUDA; HART; STORK, 2012). No entanto, outros modelos, como por exemplo um simples classificador KNN (DUDA; HART; STORK, 2012) produzirá valores de acerto muito próximos das pontuações do classificador ideal, ou seja, no caso de valores de desempenho super otimizados e tendenciosos, esses serão visíveis em todas as arquiteturas de classificador. Neste trabalho, quatro classificadores bastante distintos são empregados, *K-Nearest Neighbors* (COVER; HART, 1967), (SVM) (VAPNIK, 2013) com *kernel* RBF, *Random Forest* (BREIMAN, 2001a) e uma Rede Neural Convolucional Unidimensional (LECUN; BENGIO et al., 1995; BENGIO; GOODFELLOW; COURVILLE, 2017). Em todos os classificadores, é relizada uma busca em grade.

A busca em grade de hiperparâmetros realiza uma busca exaustiva pela melhor combinação de hiperparâmetros e retorna aqueles que resultaram no menor erro de validação. Para cada hiperparâmetro, é selecionado um intervalo de valores para ser explorado. É feito então um produto cartesiano de cada intervalo, para cada hiperparâmetro todas as combinações são testadas.

A busca em grade pode ser melhorada com a técnica de validação cruzada. Na validação cruzada tradicional, quando simplesmente é reservada uma parte dos dados para treino e outra para avaliação, um problema que pode surgir que é a avaliação pode ter muita variância. Isso porque ela pode depender muito de quais observações foram para os dados de treino e quais foram para os dados de avaliação. Desta forma, a avaliação pode ser diferente dependendo de como a divisão dos dados é feita.

A tabela 5 mostra as arquiteturas, juntamente com todas as combinações de hiperparâmetros que são ajustados durante a validação cruzada aninhada, conforme figura 23. Por exemplo, o SVM com *kernel* RBF pode variar dois hiperparâmetros diferentes, o parâmetro de penalidade C e o parâmetro γ do núcleo gaussiano. Cada hiperparâmetro pode ter respectivamente cinco e quatro diferentes valores. Consequentemente, o produto cartesiano cria 20 diferentes (C, γ) durante o ajuste dos hiperparâmetros, ou seja (0,001, 0,001), (0,001, 0,01),. . ., (10,0, 0,1), (10,0, 1,0).

Tabela 5 – Hiperparâmetros usados para cada arquitetura de classificador

Método	Hiperparâmetros	Valores
KNN	Número de vizinhos	$\{1, 3, 5, ..., 15\}$
	Métrica de distancia	{*Euclidean, Weighted*}
SVM	C-SVM C, $\log_{10}(.)$	$\{-3, -2, -1, 0, 1\}$
	RBF Kernel γ $\log_{10}(.)$	$\{-3, -2, -1, 0\}$
RF	Numero de árvores(*estimators*)	$\{10, 20, 50, 100, 200, 500\}$
	Numero de características para divisão	$\{1, 2 \ldots, 20\}$
CNN-1-D	*Filters*	$\{16, 32\}$
	Kernel size	$\{16, 32\}$
	Numero de camadas convolucionais	$\{1, 2\}$
	1-D *Pooling window size*	$\{8\}$

4.4 Experimentos

O experimento testou quatro classificadores: KNN, SVM, RF e uma Rede Neural com camada convolutiva unidimensional, e três métodos de avaliação progressivamente mais difíceis de generalização: Memorização, Generalização por Carga e Generalização por Carga e Severidade. Em todos os métodos, os modelos para os classificadores foram construídos de forma otimizada com o ajuste automático dos hiperparâmetros dentro do conjunto de valores pré-determinados para cada classificador. Os valores utilizados como parâmetros de *Grid Search* nos classificadores foram os apresentados na tabela 5

A aquisição dos exemplos foi feita com uma janela de 512 pontos a partir do *drive end* e *fan end* sem superposição ou lacuna entre os pontos. Os segmentos que não completaram a quantidade de pontos de uma amostra foram descartados. Foram consideradas, as classes Normal mais três tipos de defeitos com um total de 4 classes observadas. Os exemplos foram coletados com taxa de 12.000 amostras por segundo.

Dois modelos de características de (RAUBER; BOLDT; VAREJÃO, 2014) são usados: características estatísticas clássicas do domínio do tempo e da frequência são mescladas com um vetor *wavelet package*. Todos os sinais são divididos em partes de 512 pontos para formar um único padrão. Cada amostra é submetida à extração de características conforme modelo definido em (RAUBER; BOLDT; VAREJÃO, 2014) para formar o vetor de características usado nos métodos de aprendizado de máquina.

Frequentemente, os trabalhos da tabela 1 definem como classe, o sinal contido em um único arquivo CWRU, como por exemplo o 234.mat, dividindo posteriormente todo o sinal em blocos com ou sem sobreposição de intervalos. Neste trabalho, uma classe é definida em termos mais gerais, mesclando posições do acelerômetro (DE e FE), cargas simuladas (0 hp, 1 hp, 2 hp, 3 hp) e severidades (0,007 pol, 0,014 pol, 0,021 pol, 0,028 pol). Esta abordagem melhor representa as situações reais de falha, já que em caso da ocorrência de um problema, a gravidade, posição, ou outras características são inacessíveis. Portanto, a *priori*, existem apenas quatro classes: normal, *Inner Race*, *Outer Race* e *Ball*.

4.4.1 Memorização

No método de memorização o conjunto de dados é definido de maneira que contenha todos os níveis de carga e diferentes valores de severidade reunidos em um único dataset, nele é utilizado um *K-fold* com $k = 4$. Essa escolha de $K = 4$ é motivada pelo experimento subsequente em que os *folds* estão relacionadas às quatro cargas distintas do motor. Também foi definido o número de rounds R = 4.. O processo é repetido em 4 *rounds* onde cada um tem uma separação de *folds* diferente do *dataset*. O processo de divisão dos conjuntos usa sementes diferentes para divisão dos *folds* em cada round então (0), (1), (2) e (3) de cada round são diferentes conjuntos.

São esperadas altas taxas de acerto com este método pois nele, amostras provenientes da mesma aquisição física estão presentes nos *folds* de treino e teste simultaneamente. O impedimento para o resultado final não ser totalmente tendencioso é o fato desse valor representar a média de todos os rounds de classificação, além de ter os hiperparâmetros ajustados automaticamente nos algoritmos. Essa é a forma mais usual encontrada nas publicações analisadas neste trabalho de como as amostras de dados pertencentes aos sinais do CWRU são utilizadas para definir as classes, que serão usadas para o diagnóstico de falhas utilizando aprendizado de máquina sem modelo. Nos trabalhos compilados na tabela 2, observa-se que por várias vezes uma classe é definida apenas por um arquivo do CWRU. Neste método, foi utilizado, como exemplo dos trabalhos citados, a mesclagem de muitas condições da máquina em uma única classe, por exemplo, a classe *inner race* é formada por amostras de todos os 28 arquivos com diferentes gravidades, posições dos sensores e cargas do motor associado ao defeito interno da pista.

4.4.2 Generalização por carga

A generalização por carga define o conjunto de dados da seguinte maneira: como na Memorização, é feita a mesclagem de muitas condições da máquina em uma única classe, são realizados quatro *rounds* com quatro *folds*. Em cada um dos *Rounds*, há quatro ciclos externos de teste (um para cada nível de carga) e um ciclo interno de três *folds* de treino, ajuste de hiperparâmetros e validação, que são compostos pelos três níveis de carga restante. O processo é repetido até que todas as combinações dos valores de carga são exploradas.

As taxas, contudo, são menores que no modelo de memorização pois há uma maior dificuldade de generalização dos dados. Os elevados valores de acerto podem ser atribuídos a repetições nos padrões dos sinais de falhas nos diferentes níveis de carga. O principal objetivo do método é que a validação externa não tenha amostras provenientes da mesma aquisição presentes no treino e teste simultaneamente. Em cada *fold*, três diferentes combinações de estados normais e de falha pertencentes a cada nível de carga são utilizados para o treinamento e a combinação restante é usada para teste. Em principio, os resultados dos *rounds* deveria ser exatamente o mesmo, o que não acontece devido a diferenças estocásticas e na sequencia de apresentação dos exemplos aos métodos de treinamento.

Este modo é mais geral que o *K-fold* normal porque as amostras de testes não são extraídas de uma aquisição que também está presente no treino, mas por se tratar de aquisições diferentes do mesmo equipamento com cargas diferentes, não é um problema muito mais difícil que o *K-fold* normal. Portanto, ainda não avalia a generalidade de condições como deveria ser, de acordo com a proposta deste trabalho, pois não considera os diferentes valores de severidade associado às falhas que também são importantes para aumentar a generalização do sistema. O processo é repetido até que todas as combinações dos valores de carga são exploradas.

Considerando a figura 23, em cada ciclo, a rodada de teste contém um valor de carga motor que não está presente nos *folds* de treino e validação. Por exemplo, no primeiro ciclo, a carga de 0 hp está presente apenas no *fold* de teste, os *folds* de treinamento e de validação contêm as cargas de 1 hp, 2 hp e 3 hp. A aleatoriedade do resultado para as diferentes rodadas está relacionada ao repetido embaralhamento das amostras, porém nunca misturando as diferentes cargas. O objetivo deste modo do experimento é validar a capacidade de generalização em relação a um único estado operacional variável durante o processo, especificamente a carga do motor neste caso.

4.4.3 Generalização por carga e severidade

A generalização por carga e severidade define o conjunto de dados da seguinte forma: são realizados quatro *rounds*, cada um contando com um ciclo externo de quatro

folds de teste (um para cada par carga – severidade) e um ciclo interno de três *folds* de treino, ajuste de hiperparâmetros e validação com as combinações restantes, os dados para treinamento são obtidos da composição de valores de estados normais de uma carga com os valores de todos os tipos de falha de um nível de severidade. Em cada *fold*, três diferentes combinações de carga e falha são utilizadas para o treinamento e a combinação restante é usada para teste conforme a seguinte distribuição da tabela 6.

Tabela 6 – Separação dos *folds* nos quatro *rounds* para experimentação nos modos de generalização por carga e por carga e severidade.

	(Carga, Severidade)			
Round	*Fold* 1	*Fold* 2	*Fold* 3	*Fold* 4
1	(0, 7)	(1, 14)	(2, 21)	(3, 28)
2	(1, 7)	(2, 14)	(3, 21)	(0, 28)
3	(2, 7)	(3, 14)	(0, 21)	(1, 28)
4	(3, 7)	(0, 14)	(1, 21)	(2, 28)

No *round* 1, os 4 *folds* de teste são (0,7), (1, 14), (2, 21), (3, 28). Os estados normais de carga 0 são combinados com todos os tipos de defeitos de severidade 7 com qualquer carga, os normais de carga 1 são combinados com todos os tipos de defeitos de severidade 14 com qualquer carga. Quando se separa o *fold* (0,7) para teste no ciclo externo, usam-se os *folds* (1,14), (2,21), (3,28) no ciclo interno. A configuração de hiperparâmetros que obtiver melhor desempenho, será usada para treinar o classificador com os *folds* (1, 14), (2, 21), (3, 28) e para testar com o *fold* (0,7) no ciclo externo.

Essa forma de divisão dos dados torna o processo de aprendizado mais árduo e a classificação mais desafiadora de ser realizada. A tarefa corresponde a situação real onde se deseja diagnosticar uma falha em um ponto de operação da máquina que nunca tenha ocorrido no equipamento.Cabe ressalvar que em um ambiente industrial não se encontra os dados de falhas e gravidades apresentadas pelos equipamentos de forma discriminada e organizada como nos dados do CWRU, necessitando que o sistema de aprendizado para diagnóstico de falhas possua um grande poder de generalização para gerar informações confiáveis e úteis sobre o estado das máquinas.

A estrutura mostrada na figura 23 é novamente empregada para este método de avaliação. Cada *fold* é formado por um par (exemplos normais de uma carga + exemplos de uma falha com todas as cargas) que ocorre apenas uma vez no *fold* de teste. A divisão está representada na tabela 6. Cada um dos *rounds* R = 4 adicionalmente possui combinações diferentes do par carga/gravidade. Por exemplo, na terceira rodada, se o par (3, 14) for definido como o *fold* de teste, os *folds* restantes (2, 7), (0, 21) e (1, 28) são utilizados para

treinamento e ajuste de hiperparâmetros. A motivação para se ter um par carga/severidade é que a condição normal da máquina é distinguível apenas pelos diferentes níveis de carga. Vale a pena observar que na segunda parte do par, os arquivos para uma certa gravidade são compostos por todos os quatro valores de carga.

5 Estudo de Caso

Os quatro diferentes classificadores propostos, juntamente com seus valores de hiperparâmetros, foram testados para os três modos de divisão dos dados descritos seguindo um modelo de validação que ainda não foi retratado em publicações correlatas. Além da acurácia, o critério *F1 score* também foi utilizado como métrica de desempenho no diagnóstico das falhas. Os resultados sugerem que o uso de modos de generalização com maior diversidade entre as aquisições dos sinais do CWRU pode contribuir para resultados mais justos de desempenho. Embora os valores de desempenho sejam menores para os modos mais severos de generalização, podemos assegurar que não existem exemplos de uma mesma aquisição no treino e no teste.

Esse fato contribui para reforçar a pesquisa por novos modelos de diagnóstico que possibilitem maior desempenho seguindo a metodologia proposta, bem como propor modelos com maior pluralidade de condições observadas no equipamento de teste, se aproximando cada vez mais de uma situação real. O trabalho desenvolvido pode ser utilizado para futuras pesquisas apenas com pequenas modificações para inclusão ou modificação de classificadores, métodos de extração de características, novas métricas, parâmetros de *Grid search* e análise estatística dos resultados. O algoritmo proposto neste trabalho foi executado em um computador com a seguinte configuração: processador Intel Core i5 7400 3 GHz, memória RAM 16 GB e GPU NVIDIA TITAN V 12 GB.

Adiante, são reportados os valores dos resultados apurados após a execução do algoritmo para cada modo de generalização.

5.1 Memorização

A Tabela 7 e a tabela 8 mostram os resultados de desempenho para o modo Memorização considerando os critérios de acurácia e *F1 Score*. Como na maioria dos trabalhos revisados que utilizam essa técnica, os valores de desempenho são extraordinariamente altas devido a forma simplista de como é feita essa forma de avaliação. Foi constatado grandes valores de acerto entre todos os classificadores e nas figuras 24 e 25 e mostrada a distribuição dos valores por *fold* em cada critério para cada classificador.

5.2 Generalização por carga

Podem ser observados valores de alto desempenho, devido à ocorrência simultânea de padrões de sinais muito semelhantes nos conjuntos de treinamento e teste, os sinais

Tabela 7 – Memorização - Acurácia

Round	KNN	SVM	RF	CNN-1-D
1	0.9752	0.9432	0.9816	0.9909
2	0.9752	0.9428	0.9823	0.9815
3	0.9752	0.9431	0.9828	0.9916
4	0.9752	0.9438	0.9821	0.9948
Média	0.9752	0.9432	0.9822	0.9897

Tabela 8 – Memorização - F1 *Score*

Round	KNN	SVM	RF	CNN-1-D
1	0.9748	0.9443	0.9819	0.9910
2	0.9748	0.9439	0.9826	0.9820
3	0.9748	0.9442	0.9830	0.9916
4	0.9748	0.9449	0.9824	0.9948
Média	0.9748	0.9443	0.9825	0.9898

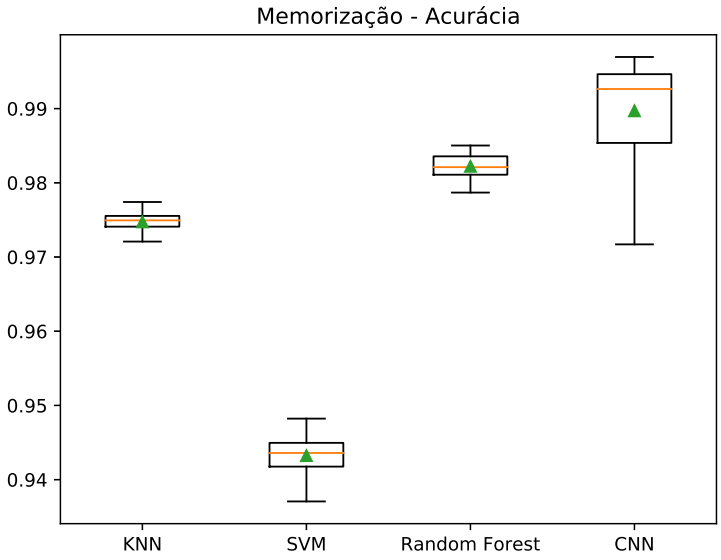


Figura 24 – Resultados de acurácia por classificador no modo memorização

diferenciam-se, em maior parte, somente pela amplitude em decorrência das diferentes cargas, o que constituir um viés de semelhança. O problema fundamental para a aplicabilidade prática desse procedimento de aprendizado é a ausência de exemplos de treinamento que abranjam todas as possíveis situações de falha. Todas as quatro arquiteturas classificadoras exibem pontuações de desempenho relativamente altas. Na tabela 9 e na tabela 10, são mostradas a acurácia e o *F1 Score* obtidos. Este modo é mais difícil que o Memorização, porque na fase de teste, o sistema vai encontrar um sinal originário de valores de carga que

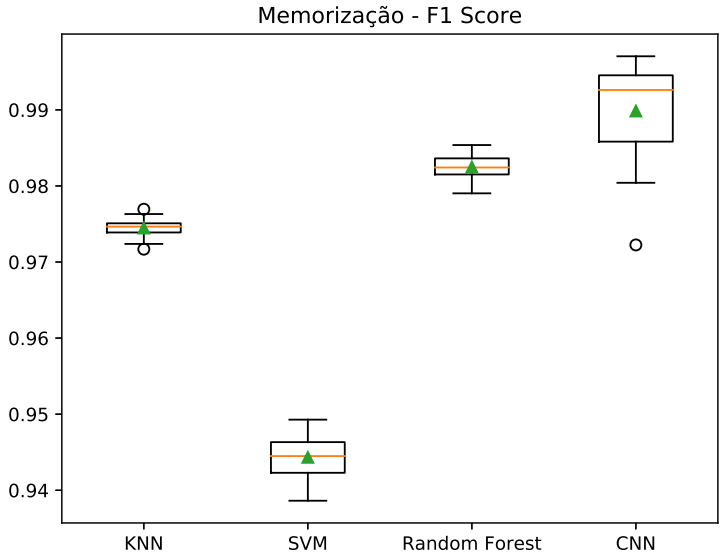


Figura 25 – Resultados de F1 *Score* por classificador no modo memorização

não foi utilizado para o treinamento. Valores de alto desempenho foram obtidos para os classificadores KNN e RF, mesmo para cargas diversas. Isso pode ser atribuído à presença de assinaturas de falhas muito semelhantes, diferenciadas apenas pela magnitude das diferentes cargas, tanto nos *folds* de treinamento quanto nos de teste. Os classificadores SVM e CNN-1D tiveram desempenho um pouco pior em média, mas ainda obtiveram alto desempenho de classificação. as figuras 26 e 27 mostram a distribuição dos valores de desempenho por *folds* para os classificadores neste modo.

Tabela 9 – Generalização por carga - Acurácia

Round	K-NN	SVM	RF	CNN-1-D
1	0.9471	0.9242	0.9513	0.9413
2	0.9494	0.9239	0.9475	0.9278
3	0.9531	0.9188	0.9528	0.9320
4	0.9543	0.9260	0.9499	0.9046
Média	0.9510	0.9232	0.9504	0.9264

5.3 Generalização por severidade

Por fim, para a tarefa de diagnóstico mais difícil os valores de acurácia e *F1 Score* estão listadas na tabela 11 e na tabela 12. Pode-se observar que os valores de desempenho caem drasticamente. Agora, uma situação mais próxima do real é simulada com maior fidelidade. Um sistema de diagnóstico de falhas deve ser treinado com os dados disponíveis. Durante o estágio operacional do sistema, sob um ponto de ajuste diferente da carga e

Tabela 10 – Generalização por carga - F1 *Score*

Round	K-NN	SVM	RF	CNN-1-D
1	0.9474	0.9258	0.9530	0.9437
2	0.9476	0.9301	0.9493	0.9276
3	0.9454	0.9293	0.9543	0.9373
4	0.9455	0.9289	0.9514	0.9375
Média	0.9465	0.9285	0.9514	0.9365

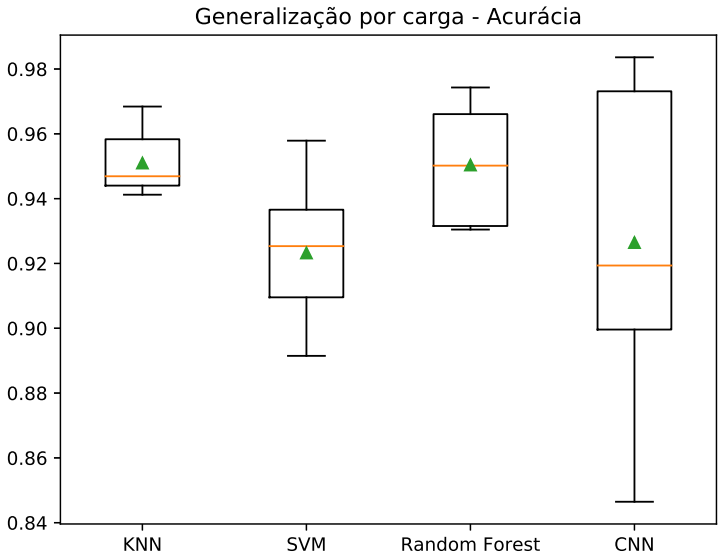


Figura 26 – Resultados de acurácia por classificador no modo generalização por carga

com um grau maior ou menor da gravidade dos defeitos dos rolamentos, um sinal deve ser diagnosticado pelo sistema treinado.

Uma observação importante é o desempenho superior da Rede Neural Convolucional. No estágio atual de nosso conhecimento, podemos apenas especular que a capacidade de generalização desse tipo de arquitetura de aprendizado para problemas de classificação mais difíceis é melhor do que métodos alternativos. Em (LOCA; RAUBER, 2019) o uso de uma Rede Neural Convolucional unidimensional mostrou ser um método satisfatório de avaliação para diagnóstico das falhas mais difíceis de serem identificadas no simulador *Tennessee Eastman*. A quantidade considerável de pesquisas bem-sucedidas relacionadas a essa abordagem de aprendizado profundo sugere que a CNN é também uma boa técnica para o diagnóstico de falhas com base no aprendizado de máquina.

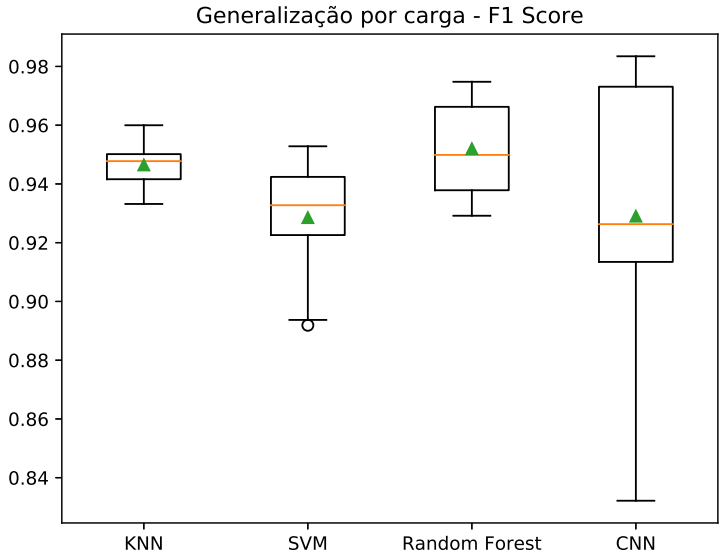


Figura 27 – Resultados de F1 *Score* por classificador no modo generalização por carga

Tabela 11 – Generalização por severidade - Acurácia

Round	K-NN	SVM	RF	CNN-1D
1	0.5185	0.4856	0.5277	0.5714
2	0.4856	0.4537	0.5061	0.5113
3	0.5137	0.4803	0.5243	0.5540
4	0.5092	0.4771	0.5165	0.5819
Média	0.5067	0.4742	0.5187	0.5546

Tabela 12 – Generalização por severidade - F1 *Score*

Round	K-NN	SVM	RF	CNN-1D
1	0.4638	0.4390	0.4667	0.5139
2	0.4635	0.4388	0.4689	0.4935
3	0.4645	0.4396	0.4609	0.5071
4	0.4614	0.4381	0.4558	0.5345
Média	0.4633	0.4389	0.4631	0.5122

5.4 Análise estatística

A análise baseada apenas nos resultados médios de desempenho pode levar a conclusões errôneas, uma vez que a variabilidade não é considerada nesta estratégia simples. Uma abordagem mais confiável é verificar a significância estatística da diferença nos resultados via teste de hipótese estatística.

Um procedimento tradicional para detectar estaticamente a diferença entre dois

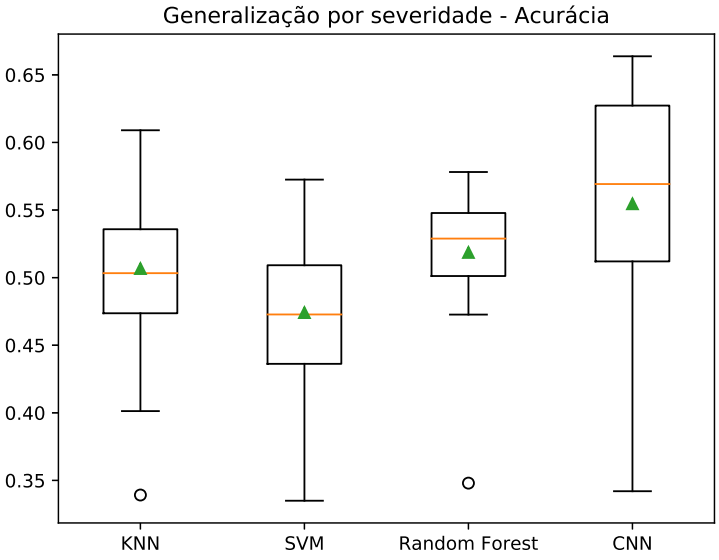


Figura 28 – Resultados de acurácia por classificador no modo generalização por severidade

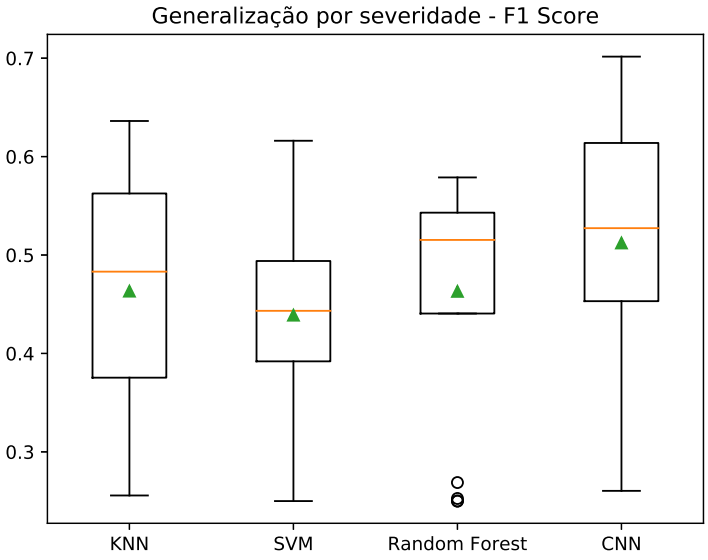


Figura 29 – Resultados de F1 *Score* por classificador no modo generalização por severidade

grupos em um experimento compartivo é o *Student* t-test (CASELLA; BERGER, 2002). No contexto da avaliação de desempenho de dois classificadores, o t-test é uma alternativa possível para detectar diferenças somente se a avaliação de desempenho for realizada em vários conjuntos de dados, uma vez que uma das premissas do t-test é a independência entre as amostras. Esse não é o caso de comparações baseadas nos valores de desempenho de um estudo de validação cruzada em um único conjunto de dados, pois a interseção dos conjuntos de treinamento não é o conjunto vazio.

(NADEAU; BENGIO, 2000) mostrou que a violação da independência gera subesti-

mação do desvio padrão. Para superar essa falha, os autores usam a proporção do tamanho dos conjuntos de treinamento e teste para estimar a correlação derivada dos conjuntos de treinamento sobrepostos. Finalmente, os autores corrijem o habitual t-test para propor a estatística para realizar o teste. Nessa equação $\overline{d}$ e s^2 são a média e a variação das diferenças no desempenho dos dois classificadores. Os valores n_{TE} e n_{TR} são o tamanho dos conjuntos de treinamento e teste, respectivamente. O número total de conjuntos de testes usados é J, portanto, considerando a validação cruzada em resulta em $J = K \times R$. Pequenos valores de t sugerem desempenho semelhante entre os classificadores, enquanto grandes valores absolutos indicam diferença significativa nos desempenhos. o p-value do teste é dado por $2 \cdot P(T > |t|)$, onde T é uma variável aleatória que segue uma distribuição t-student com $(J - 1)$ graus de liberdade. Vale a pena notar que, para evitar viés de similaridade, o tamanho dos *folds* pode diferir, e, portanto, a proporção $n_{\text{TE}}/n_{\text{TR}}$ não é constante ao longo do processo de avaliação. Nesse caso, sugerimos substituir n_{TR} e n_{TE} pelo número total de amostras usadas como conjuntos de treinamento e teste, respectivamente. Sob estes circunstâncias a relação $n_{\text{TE}}/n_{\text{TR}}$ é igual a $1/K$.

$$t = \frac{\overline{d}}{\sqrt{s^2 \left(\frac{1}{J} + \frac{n_{\text{TE}}}{n_{\text{TR}}}\right)}}$$

Nas aplicações em que a comparação de desempenho entre três ou mais métodos é necessária, é possível executar a mesma análise estatística para cada par de classificadores, porém ajustando o p-values para controlar o erro familiar, ou seja, o erro cumulativo de Tipo I.

Para controlar o erro familiar, várias soluções estatísticas foram propostas. (OLIVEIRA-SANTOS et al., 2018) empregaram o procedimento de abaixamento de Holm (HOLM, 1979) para comparar os resultados de vinte classificadores diferentes. Neste trabalho, sugerimos o uso do procedimento Benjamini-Hochberg (BENJAMINI; HOCHBERG, 1995). Esta solução é mais eficaz que a abordagem de Holms sob dependência não negativa, que é o caso neste trabalho, uma vez que a sobreposição nos conjuntos de treinamento leva a correlações positivas. O procedimento de Benjamini-Hochberg começa ordenando cada vez mais p-values associado a cada teste emparelhado, $p_{(1)}, \ldots, p_{(m)}$, onde m é a quantidade total de testes necessários, ou seja, o número de comparações aos pares. Então, para um determinado nível de significância α, deixe k seja o maior i de tal modo que

$$k = \arg\max_i \left\{ p_{(i)} \leq \frac{i}{m}\alpha \right\}$$

Por fim, rejeita as hipóteses de similaridade no desempenho das comparações associado com o p-values $p_{(1)}, \ldots, p_{(k)}$

Tabela 13 – Matriz de resultados da comparação pareada entre modelos classificadores para o modo de memorização. O triângulo superior apresenta os p-values correspondentes do t-test corrigido para cada par de métodos quando a acurácia é assumida como critério de desempenho. O triângulo inferior mostra resultados semelhantes com base no *F1 score*.

		p-value acurácia		
	KNN	1.2E-10	5.8E-5	0.006
p-value F1-Score	3.8E-10	SVM	2.9E-12	2.5E-8
	1.8E-5	7.4E-12	RF	0.115
	0.003	1.9E-8	0.100	CNN

Tabela 14 – Matriz de resultados da comparação pareada entre modelos classificadores para o modo generalização por carga.

		p-value acurácia		
	KNN	0.032	0.951	0.335
p-value F1-Score	0.063	SVM	0.053	0.891
	0.654	0.044	RF	0.224
	0.476	0.979	0.317	CNN

Tabela 15 – Matriz de resultados da comparação pareada entre modelos classificadores para o modo generalização por carga e severidade de falha.

		p-value acurácia		
	KNN	0.171	0.662	0.269
p-value F1-Score	0.420	SVM	0.199	0.027
	0.465	0.257	RF	0.404
	0.159	0.004	0.936	CNN

6 Considerações finais

O trabalho apresentou uma estrutura de avaliação de desempenho para diagnóstico de falhas mais confiável do que as técnicas convencionais de validação cruzada aninhada. O procedimento foi aplicado ao conjunto de dados do CWRU em razão da escassez de trabalhos que aplicam uma metodologia mais realista ao problema, foi proposta uma abordagem composta de três modos operacionais diferentes para uma maior diversidade de generalização. O trabalho também tem como propósito principal servir como uma referência para futuros trabalhos de comparação de diferentes abordagens para diagnósticos de falhas aplicáveis ao conjunto de dados CWRU.

Os resultados deste trabalho ilustram as dificuldades relacionadas ao diagnóstico de falhas baseado no paradigma do aprendizado supervisionado. Situações reais de falhas no maquinário são cenários extremamente difíceis de serem construídos e podem superar facilmente o modo de generalização mais difícil experimentado neste trabalho. Variáveis diversas estão envolvidas em um cenário real como a degradação das máquinas que podem causar alteração dos padrões observados, a degradação dos próprios sensores que podem gerar dados inconsistentes com a situação do equipamento, além da possibilidade de novas falhas que nunca foram observadas aparecerem no maquinário em sua fase de produção não sendo identificadas e comprometendo o diagnóstico das demais.

Outro ponto crítico para o contexto de aprendizado supervisionado é a confiabilidade da rotulação aplicada as amostras que necessitam ser avaliadas *a priori* por um especialista confiável. Isso introduz uma fonte de incerteza que também tem relevância para a aplicação do modelo de classificação.

6.1 Limitações e Questões Práticas

Neste trabalho, o modo de generalização por severidade representa a condição de uso dos dados do CWRU que mais explora a diversidade de formas de aquisição dos dados e a maneira como pode ser feita a divisão do conjunto em treino e teste. Mesmo desta forma, o método não poderia representar totalmente o domínio para o diagnóstico de falhas em uma situação real de operação do maquinário, pois nesse cenário, desafios ainda maiores para o diagnóstico seriam encontrados em função da imprevisibilidade quanto ao aparecimento de novas assinaturas de falhas nos sinais, que podem representar defeitos fora do escopo das condições que foram inicialmente utilizadas para o aprendizado, representando as condições de falhas no sistema. A consequência natural é imaginar que os valores de desempenho dos classificadores em condições reais de uso seriam menores que os apresentados para o método mais difícil desta pesquisa.

Os resultados retratados para os três modos de generalização estão limitados aos valores de parâmetros que são sensíveis para o resultado final e também para o tempo de execução do algoritmo como o uso dos quatro diferentes classificadores e do conjunto de hiperparâmetros disponíveis para o *Grid Search* de cada um deles. Seria desejável um maior número possível desses parâmetros para uma maior comparação e exatidão dos resultados, mas as quantidades utilizadas são reflexo da limitação imposta pelo hardware utilizado para executar a aplicação.

6.2 Trabalhos futuros

A metodologia proposta pode ser alterada para a experimentação de outras formas de generalização com os dados, buscando uma maior variedade de representação das condições da máquina em relação às diferentes formas de aquisição dos dados da base utilizada.

As limitações de processamento para o uso de *Grid Search* podem ser superadas no futuro com a utilização de GPU com maior processamento, o que possibilitará o uso de um maior número de combinações de hiperparâmetros na experimentação.

Novos classificadores, métricas, métodos de extração de características, etc podem ser utilizados bem como os classificadores já empregados podem sofrer adaptações para proporcionar menores taxas de erro de classificação.

Para facilitar as comparações experimentais de métodos envolvendo os dados da CWRU, o código-fonte Python e os resultados experimentais completos deste trabalho são fornecidos em http://bit.ly/2S0Dnhj. A estrutura de programação permite avaliar a classificação, extração de recursos e métodos de seleção de recursos. É implementado usando *Numpy*, *Scikit-learn* e *Keras*, seguindo o padrão de design dessas bibliotecas.

Em trabalhos futuros, outras fontes de sinal podem ser incluídas para corroborar a metodologia proposta, por exemplo, o FEMTO *Bearing Data Set* do NASA *Prognostics Center of Excellence Data Repository* (NASA. . . , 2020).

Referências

BEARING DataCenter, Paderborn University. 2020. Https://mb.uni-paderborn.de/kat/forschung/datacenter/bearing-datacenter. Accessed: 2020-06-10. Citado na página 45.

BENGIO, Y.; GOODFELLOW, I.; COURVILLE, A. *Deep learning*. [S.l.]: Citeseer, 2017. v. 1. Citado na página 72.

BENJAMINI, Y.; HOCHBERG, Y. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological)*, Wiley Online Library, v. 57, n. 1, p. 289–300, 1995. Citado na página 85.

BERREDJEM, T.; BENIDIR, M. Bearing faults diagnosis using fuzzy expert system relying on an improved range overlaps and similarity method. *Expert Systems with Applications*, Elsevier, v. 108, p. 134–142, 2018. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

BOLDT, F. d. A.; RAUBER, T. W.; VAREJAO, F. M. Feature extraction and selection for automatic fault diagnosis of rotating machinery. *X Encontro Nacional de Inteligencia Artificial e Computacional (ENIAC).-Fortaleza, Ceará*, p. 213–220, 2013. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

BOLDT, F. de A. et al. Fast feature selection using hybrid ranking and wrapper approach for automatic fault diagnosis of motorpumps based on vibration signals. In: IEEE. *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*. [S.l.], 2015. p. 127–132. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

BREIMAN, L. Random Forests. *Machine Learning*, v. 45, n. 1, p. 5–32, 2001. ISSN 0885-6125, 1573-0565. Citado na página 72.

BREIMAN, L. Random forests machine learning. 45: 5–32. *View Article PubMed/NCBI Google Scholar*, 2001. Citado na página 31.

CASE Western Reserve University, Bearing Data Center. 2014. Http://csegroups.case.edu/bearingdatacenter. Accessed: 2020-06-10. Citado na página 45.

CASELLA, G.; BERGER, R. L. *Statistical inference*. [S.l.]: Duxbury Pacific Grove, CA, 2002. v. 2. Citado 2 vezes nas páginas 64 e 84.

CAWLEY, G. C.; TALBOT, N. L. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, v. 11, n. Jul, p. 2079–2107, 2010. Citado na página 44.

CHIANG, L.; BRAATZ, R.; RUSSELL, E. *Fault Detection and Diagnosis in Industrial Systems*. Springer London, 2001. (Advanced Textbooks in Control and Signal Processing). ISBN 9781852333270. Disponível em: <http://web.mit.edu/braatzgroup/links.html>. Citado na página 38.

CORREIA, P. R.; FERREIRA, M. Reconhecimento de padrões por métodos não supervisionados: explorando procedimentos quimiométricos para tratamento de dados analíticos. *Química Nova*, SciELO Brasil, v. 30, n. 2, p. 481–487, 2007. Citado na página 36.

COVER, T.; HART, P. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, IEEE, v. 13, n. 1, p. 21–27, jan. 1967. ISSN 0018-9448. Citado 2 vezes nas páginas 67 e 72.

DASH, S.; VENKATASUBRAMANIAN, V. Integrated framework for abnormal event management and process hazards analysis. *AIChE journal*, Wiley Online Library, v. 49, n. 1, p. 124–139, 2003. Citado na página 24.

DING, S. *Model-Based Fault Diagnosis Techniques: Design Schemes, Algorithms and Tools*. [S.l.]: Springer London, 2012. (Advances in Industrial Control). ISBN 9781447147992. Citado na página 38.

DING, S. X. *Data-driven Design of Fault Diagnosis and Fault-tolerant Control Systems*. [S.l.]: Springer, 2016. Citado na página 38.

DUDA, R. O.; HART, P. E.; STORK, D. G. *Pattern classification*. [S.l.]: John Wiley & Sons, 2012. Citado 2 vezes nas páginas 59 e 72.

EREN, L.; INCE, T.; KIRANYAZ, S. A generic intelligent bearing fault diagnosis system using compact adaptive 1d cnn classifier. *Journal of Signal Processing Systems*, Springer, v. 91, n. 2, p. 179–189, 2019. Citado 6 vezes nas páginas 51, 52, 53, 54, 55 e 56.

FIORIN, D. V. et al. Aplicações de redes neurais e previsões de disponibilidade de recursos energéticos solares. *Revista Brasileira de ensino de Física*, SciELO Brasil, v. 33, n. 1, p. 01–20, 2011. Citado na página 35.

FUKUNAGA, K. *Introduction to Statistical Pattern Recognition*. 2nd. ed. San Diego: Academic Press, 1990. Citado 3 vezes nas páginas 17, 59 e 60.

GAO, Z.; CECATI, C.; DING, S. X. A survey of fault diagnosis and fault-tolerant techniques—part i: Fault diagnosis with model-based and signal-based approaches. *IEEE Transactions on Industrial Electronics*, IEEE, v. 62, n. 6, p. 3757–3767, 2015. Citado na página 38.

GEBRAEEL, N. et al. Residual life predictions from vibration-based degradation signals: a neural network approach. *IEEE Transactions on industrial electronics*, IEEE, v. 51, n. 3, p. 694–700, 2004. Citado na página 37.

GERTLER, J. *Fault detection and diagnosis in engineering systems*. [S.l.]: Routledge, 2017. Citado na página 38.

GRITLI, Y. et al. Condition monitoring of mechanical faults in induction machines from electrical signatures: Review of different techniques. In: IEEE. *2017 IEEE 11th International Symposium on Diagnostics for Electrical Machines, Power Electronics and Drives (SDEMPED)*. [S.l.], 2017. p. 77–84. Citado na página 37.

GUO, T. et al. Simple convolutional neural network on image classification. In: IEEE. *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)(*. [S.l.], 2017. p. 721–724. Citado na página 35.

GUO, X.; CHEN, L.; SHEN, C. Hierarchical adaptive deep convolution neural network and its application to bearing fault diagnosis. *Measurement*, Elsevier, v. 93, p. 490–502, 2016. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

HAN, T.; JIANG, D. Rolling bearing fault diagnostic method based on vmd-ar model and random forest classifier. *Shock and Vibration*, Hindawi, v. 2016, 2016. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

HOANG, D.-T.; KANG, H.-J. Rolling element bearing fault diagnosis using convolutional neural network and vibration image. *Cognitive Systems Research*, Elsevier, v. 53, p. 42–50, 2019. Citado 6 vezes nas páginas 51, 52, 53, 54, 55 e 56.

HOLM, S. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, JSTOR, p. 65–70, 1979. Citado na página 85.

KAVATHEKAR, S.; UPADHYAY, N.; KANKAR, P. Fault classification of ball bearing by rotation forest technique. *Procedia Technology*, Elsevier, v. 23, p. 187–192, 2016. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

KOTAK, V.; JAIWAL, N.; PATEL, S. Improvising strategies for efficiency of ie 4 scim 2.2 kw through simulation. In: IEEE. *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*. [S.l.], 2016. p. 3932–3936. Citado na página 37.

LECUN, Y.; BENGIO, Y. et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, v. 3361, n. 10, p. 1995, 1995. Citado na página 72.

LEE, J. et al. *Bearing Data Set, NASA Ames Prognostics Data Repository*. [S.l.]: NASA Ames Research Centre Moffett Field, CA, 2007. Citado na página 45.

LEI, Y. et al. An intelligent fault diagnosis method using unsupervised feature learning towards mechanical big data. *IEEE Transactions on Industrial Electronics*, IEEE, v. 63, n. 5, p. 3137–3147, 2016. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

LI, Y. et al. Entropy based fault classification using the case western reserve university data: A benchmark study. *IEEE Transactions on Reliability*, IEEE, 2019. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

LI, Z.; FANG, H.; HUANG, M. Diversified learning for continuous hidden markov models with application to fault diagnosis. *Expert Systems with Applications*, Elsevier, v. 42, n. 23, p. 9165–9173, 2015. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

LIU, Z. et al. Automatic identification of abnormalities in 12-lead ecgs using expert features and convolutional neural networks. In: IEEE. *2018 International Conference on Sensor Networks and Signal Processing (SNSP)*. [S.l.], 2018. p. 163–167. Citado na página 35.

LOCA, A.; RAUBER, T. Uso de uma rede neural convolucional unidimensional para detecção de falhas em processos industriais. In: SBC. *Anais da XIX Escola Regional de Computação Bahia, Alagoas e Sergipe*. [S.l.], 2019. p. 42–47. Citado na página 82.

MACHADO, W. C.; JUNIOR, E. S. d. F. Redes neurais artificiais aplicadas na previsão do vtec no brasil. *Boletim de Ciências Geodésicas*, SciELO Brasil, v. 19, n. 2, p. 227–246, 2013. Citado na página 33.

MCMILLAN, G.; VEGAS, P. *Process / Industrial Instruments and Controls Handbook, Sixth Edition*. [S.l.]: McGraw-Hill Education, 2019. ISBN 9781260117981. Citado na página 38.

NADEAU, C.; BENGIO, Y. Inference for the generalization error. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2000. p. 307–313. Citado na página 84.

NANDI, S.; TOLIYAT, H. A.; LI, X. Condition monitoring and fault diagnosis of electrical motors—a review. *IEEE transactions on energy conversion*, IEEE, v. 20, n. 4, p. 719–729, 2005. Citado na página 37.

NASA Prognostics Center of Excellence – Data Repository. 2020. Https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository. Accessed: 2020-06-10. Citado na página 88.

OLIVEIRA-SANTOS, T. et al. Submersible motor pump fault diagnosis system: A comparative study of classification methods. In: IEEE. *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*. [S.l.], 2016. p. 415–422. Citado na página 69.

OLIVEIRA-SANTOS, T. et al. Combining classifiers with decision templates for automatic fault diagnosis of electrical submersible pumps. *Integrated Computer-Aided Engineering*, IOS Press, v. 25, n. 4, p. 381–396, 2018. Citado 2 vezes nas páginas 69 e 85.

PAN, J. et al. Liftingnet: A novel deep learning network with layerwise feature learning from noisy mechanical data for fault classification. *IEEE Transactions on Industrial Electronics*, IEEE, v. 65, n. 6, p. 4973–4982, 2017. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

PAOLETTI, M. et al. A new deep convolutional neural network for fast hyperspectral image classification. *ISPRS journal of photogrammetry and remote sensing*, Elsevier, v. 145, p. 120–147, 2018. Citado na página 35.

PILTAN, F. et al. Rolling-element bearing fault diagnosis using advanced machine learning-based observer. *Applied Sciences*, Multidisciplinary Digital Publishing Institute, v. 9, n. 24, p. 5404, 2019. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

RASCHKA, S. *Python Machine Learning*. Packt Publishing, 2015. ISBN 9781783555147. Disponível em: <https://books.google.com.br/books?id=GOVOCwAAQBAJ>. Citado na página 42.

RAUBER, T. W.; BOLDT, F. de A.; VAREJÃO, F. M. Heterogeneous feature models and feature selection applied to bearing fault diagnosis. *IEEE Transactions on Industrial Electronics*, IEEE, v. 62, n. 1, p. 637–646, 2014. Citado 7 vezes nas páginas 52, 53, 54, 55, 56, 71 e 74.

RAZAVI-FAR, R.; FARAJZADEH-ZANJANI, M.; SAIF, M. An integrated class-imbalanced learning scheme for diagnosing bearing defects in induction motors. *IEEE Transactions on Industrial Informatics*, IEEE, v. 13, n. 6, p. 2758–2769, 2017. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

RODRIGUEZ, P. H. et al. Application of the teager–kaiser energy operator in bearing fault diagnosis. *ISA transactions*, Elsevier, v. 52, n. 2, p. 278–284, 2013. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

SCIENCE. Portal data science. 2019. Disponível em: <https://portaldatascience.com/o-algoritmo-k-nearest-neighbors-knn-em-machine-learning/>. Acesso em: 10 mai. 2020. Citado na página 29.

SELVATHI, D.; EMALA, T. Mri brain pattern analysis for detection of alzheimer's disease using random forest classifier. *Intelligent Decision Technologies*, IOS Press, v. 10, n. 4, p. 331–340, 2016. Citado na página 32.

SHEN, C. et al. Fault diagnosis of rotating machinery based on the statistical parameters of wavelet packet paving and a generic support vector regressive classifier. *Measurement*, Elsevier, v. 46, n. 4, p. 1551–1564, 2013. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

SIQUEIRA-BATISTA, R. et al. As redes neurais artificiais e o ensino da medicina. *Revista BRasileiRa de educação Médica*, SciELO Brasil, v. 38, n. 4, p. 548–556, 2014. Citado na página 33.

SREEJITH, B.; VERMA, A.; SRIVIDYA, A. Fault diagnosis of rolling element bearing using time-domain features and neural networks. In: IEEE. *2008 IEEE Region 10 and the Third international Conference on Industrial and Information Systems*. [S.l.], 2008. p. 1–6. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

VAPNIK, V. *The nature of statistical learning theory*. [S.l.]: Springer science & business media, 2013. Citado na página 72.

VARGA, A. *Solving Fault Diagnosis Problems: Linear Synthesis Techniques*. [S.l.]: Springer International Publishing, 2017. (Studies in Systems, Decision and Control). ISBN 9783319515595. Citado na página 38.

VARGAS, R. E. V. et al. A realistic and public dataset with rare undesirable real events in oil wells. *Journal of Petroleum Science and Engineering*, Elsevier, v. 181, p. 106223, 2019. Citado na página 44.

VARMA, S.; SIMON, R. Bias in error estimation when using cross-validation for model selection. *BMC bioinformatics*, BioMed Central, v. 7, n. 1, p. 91, 2006. Citado 2 vezes nas páginas 17 e 60.

VENKATASUBRAMANIAN, V. Abnormal events management in complex process plants: Challenges and opportunities in intelligent supervisory control. In: *Proceedings FOCAPO*. [S.l.: s.n.], 2003. p. 117ff. Citado na página 24.

VENKATASUBRAMANIAN, V. et al. A review of process fault detection and diagnosis: Part i: Quantitative model-based methods. *Computers & chemical engineering*, Elsevier, v. 27, n. 3, p. 293–311, 2003. Citado na página 24.

WEN, L. et al. A new convolutional neural network-based data-driven fault diagnosis method. *IEEE Transactions on Industrial Electronics*, IEEE, v. 65, n. 7, p. 5990–5998, 2017. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

WU, S.-D. et al. Multi-scale analysis based ball bearing defect diagnostics using mahalanobis distance and support vector machine. *Entropy*, Multidisciplinary Digital Publishing Institute, v. 15, n. 2, p. 416–433, 2013. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

WU, S.-D. et al. Bearing fault diagnosis based on multiscale permutation entropy and support vector machine. *Entropy*, Molecular Diversity Preservation International, v. 14, n. 8, p. 1343–1356, 2012. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

YANG, Y.; FU, P.; HE, Y. Bearing fault automatic classification based on deep learning. *IEEE Access*, IEEE, v. 6, p. 71540–71554, 2018. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

YÉLAMOS, I. et al. Enhancing abnormal events management by the use of quantitative process hazards analysis results. *Industrial & engineering chemistry research*, ACS Publications, v. 48, n. 8, p. 3921–3933, 2009. Citado na página 23.

YIAKOPOULOS, C.; GRYLLIAS, K. C.; ANTONIADIS, I. A. Rolling element bearing fault detection in industrial environments based on a k-means clustering approach. *Expert Systems with Applications*, Elsevier, v. 38, n. 3, p. 2888–2911, 2011. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

YU, J.-B. Bearing performance degradation assessment using locality preserving projections. *Expert Systems with Applications*, Elsevier, v. 38, n. 6, p. 7440–7450, 2011. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

YU, X. et al. A novel characteristic frequency bands extraction method for automatic bearing fault diagnosis based on hilbert huang transform. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 15, n. 11, p. 27869–27893, 2015. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

ZHANG, R. et al. Transfer learning with neural networks for bearing fault diagnosis in changing working conditions. *IEEE Access*, IEEE, v. 5, p. 14347–14357, 2017. Citado 7 vezes nas páginas 38, 51, 52, 53, 54, 55 e 56.

ZHANG, Y. et al. A new subset based deep feature learning method for intelligent fault diagnosis of bearing. *Expert Systems with Applications*, Elsevier, v. 110, p. 125–142, 2018. Citado 6 vezes nas páginas 51, 52, 53, 54, 55 e 56.

ZHAO, M. et al. Fault diagnosis of rolling element bearings via discriminative subspace learning: visualization and classification. *Expert Systems with Applications*, Elsevier, v. 41, n. 7, p. 3391–3401, 2014. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

ZHAO, W.; TANG, S.; DAI, W. An improved knn algorithm based on essential vector. *Elektronika ir Elektrotechnika*, v. 123, n. 7, p. 119–122, 2012. Citado na página 30.

ZHAO, X. et al. An effective procedure exploiting unlabeled data to build monitoring system. *Expert Systems with Applications*, Elsevier, v. 38, n. 8, p. 10199–10204, 2011. Citado 5 vezes nas páginas 52, 53, 54, 55 e 56.

Apêndices

APÊNDICE A – Código fonte do *framework* desenvolvido para o modelo de avaliação proposto

```
# -*- coding: utf-8 -*-
"""CWRU-EvaluationFramework.ipynb

Original file is located at
    https://colab.research.google.com/drive/1Zf8r7o_LtOyhok_NTUv2b8t9ICOQ8ZTz

The code is separated into four sections.

* In section "CWRU dataset" the CWRU Matlab files are downloaded, the
acquisitions are extracted from the Matlab files into Numpy arrays. Then, the
data is segmented and the samples are selected. The samples are selected by
their labels using regular expressions.

* Section "Experimenter" defines the splitters mentioned in this work, i.e.
GroupShuffleKFold and BySeverityKFold, and the setup of each experiment. The
samples are grouped by their original labels also using regular expressions. For
instance, to group samples by load, the regular expression '_\d' may be used. A
list of evaluation methods is defined in this section.

* Section "Classification Models" defines the estimators and their feature
extraction methods. To instantiate a classification method with feature
extraction, a Pipeline must be made. A list of classification methods is defined
in this section.

* Finally, section "Performing Experiments" executes the experiments as they
were defined in the previous sections, showing and saving  their results. It
iterates one list of classification methods and one list of evaluation methods
in $r$ rounds. In this work, four classification methods were tested by three
evaluation methods in four rounds, resulting in $4\times 3\times 4 = 48$
experiments. New classification or evaluation methods can be tested by adding
them in their respective list.

The code can be executed direct in the Colab environment when few samples and
simple classifier methods are tested. For the experiments presented in this
work, it must be run in a local GPU, with enough memory and processing capacity.
The results are presented for each round of each experiment of each
classification method. The average and standard deviation of the rounds is also
presented, as well, the average of the differences among the classification
methods.

New feature extraction methods must receive a 3-D Numpy array and returns a 2-D
Numpy array. This is necessary because the same raw dataset is used for methods
that need feature extraction of the signal acquisitions, like K-NN, SVM and
Random Forest, and convolutional neural networks that deal with 3-D arrays. The
experiments presented here used just one channel of each acquisition, but newer
experiments may use more channels.

New classifications methods that need feature extraction may be easily added. It
is only required a Pipeline with the feature extraction method and the
classifier, like those presented in the code. A new neural network base
architecture must be wrapped in a scikit-learn estimator, and its definition
must be in the method *fit*. It cannot be defined in the method *\_\_init\_\_*
due to the implementation of the Keras library. If the network architecture is
defined in the method *\_\_init\_\_*, it will remember the samples across the
```

```
folds and the rounds, giving outstanding results, that will be never replicated
in a real scenario. It is worth to highlight that some parameters of the
network, like kernel size and number of filters, should be selected by a
GridSearchCV method to provide fairer results when compared with other methods.

#CWRU database
"""

debug = False

"""## CWRU files.

Associate each Matlab file name to a bearing condition in a Python dictionary.
The dictionary keys identify the conditions.

There are only four normal conditions, with loads of 0, 1, 2 and 3 hp.
All conditions end with an underscore character followed by an algarism
representing the load applied during the acquisitions.
The remaining conditions follow the pattern:


* First two characters represent the bearing location, i.e. drive end (DE) and
fan end (FE).
* The following two characters represent the failure location in the bearing,
i.e. ball (BA), Inner Race (IR) and Outer Race (OR).
* The next three algarisms indicate the severity of the failure, where 007
stands for 0.007 inches and 0021 for 0.021 inches.
* For Outer Race failures, the character @ is followed by a number that
indicates different load zones.
"""

def cwru_12khz():
  '''
  Retuns a dictionary with the names of all Matlab files read in 12kHz located
in

http://csegroups.case.edu/sites/default/files/bearingdatacenter/files/Datafiles/
.
  The dictionary keys represent the bearing condition.
  '''
  matlab_files_name = {}
  # Normal
  matlab_files_name["Normal_0"] = "97.mat"
  matlab_files_name["Normal_1"] = "98.mat"
  matlab_files_name["Normal_2"] = "99.mat"
  matlab_files_name["Normal_3"] = "100.mat"
  # DE Inner Race 0.007 inches
  matlab_files_name["DEIR.007_0"] = "105.mat"
  matlab_files_name["DEIR.007_1"] = "106.mat"
  matlab_files_name["DEIR.007_2"] = "107.mat"
  matlab_files_name["DEIR.007_3"] = "108.mat"
  # DE Ball 0.007 inches
  matlab_files_name["DEB.007_0"] = "118.mat"
  matlab_files_name["DEB.007_1"] = "119.mat"
```

```
matlab_files_name["DEB.007_2"] = "120.mat"
matlab_files_name["DEB.007_3"] = "121.mat"
# DE Outer race 0.007 inches centered @6:00
matlab_files_name["DEOR.007@6_0"] = "130.mat"
matlab_files_name["DEOR.007@6_1"] = "131.mat"
matlab_files_name["DEOR.007@6_2"] = "132.mat"
matlab_files_name["DEOR.007@6_3"] = "133.mat"
# DE Outer race 0.007 inches centered @3:00
matlab_files_name["DEOR.007@3_0"] = "144.mat"
matlab_files_name["DEOR.007@3_1"] = "145.mat"
matlab_files_name["DEOR.007@3_2"] = "146.mat"
matlab_files_name["DEOR.007@3_3"] = "147.mat"
# DE Outer race 0.007 inches centered @12:00
matlab_files_name["DEOR.007@12_0"] = "156.mat"
matlab_files_name["DEOR.007@12_1"] = "158.mat"
matlab_files_name["DEOR.007@12_2"] = "159.mat"
matlab_files_name["DEOR.007@12_3"] = "160.mat"
# DE Inner Race 0.014 inches
matlab_files_name["DEIR.014_0"] = "169.mat"
matlab_files_name["DEIR.014_1"] = "170.mat"
matlab_files_name["DEIR.014_2"] = "171.mat"
matlab_files_name["DEIR.014_3"] = "172.mat"
# DE Ball 0.014 inches
matlab_files_name["DEB.014_0"] = "185.mat"
matlab_files_name["DEB.014_1"] = "186.mat"
matlab_files_name["DEB.014_2"] = "187.mat"
matlab_files_name["DEB.014_3"] = "188.mat"
# DE Outer race 0.014 inches centered @6:00
matlab_files_name["DEOR.014@6_0"] = "197.mat"
matlab_files_name["DEOR.014@6_1"] = "198.mat"
matlab_files_name["DEOR.014@6_2"] = "199.mat"
matlab_files_name["DEOR.014@6_3"] = "200.mat"
# DE Ball 0.021 inches
matlab_files_name["DEB.021_0"] = "222.mat"
matlab_files_name["DEB.021_1"] = "223.mat"
matlab_files_name["DEB.021_2"] = "224.mat"
matlab_files_name["DEB.021_3"] = "225.mat"
# FE Inner Race 0.021 inches
matlab_files_name["FEIR.021_0"] = "270.mat"
matlab_files_name["FEIR.021_1"] = "271.mat"
matlab_files_name["FEIR.021_2"] = "272.mat"
matlab_files_name["FEIR.021_3"] = "273.mat"
# FE Inner Race 0.014 inches
matlab_files_name["FEIR.014_0"] = "274.mat"
matlab_files_name["FEIR.014_1"] = "275.mat"
matlab_files_name["FEIR.014_2"] = "276.mat"
matlab_files_name["FEIR.014_3"] = "277.mat"
# FE Ball 0.007 inches
matlab_files_name["FEB.007_0"] = "282.mat"
matlab_files_name["FEB.007_1"] = "283.mat"
matlab_files_name["FEB.007_2"] = "284.mat"
matlab_files_name["FEB.007_3"] = "285.mat"
# DE Inner Race 0.021 inches
matlab_files_name["DEIR.021_0"] = "209.mat"
```

```
matlab_files_name["DEIR.021_1"] = "210.mat"
matlab_files_name["DEIR.021_2"] = "211.mat"
matlab_files_name["DEIR.021_3"] = "212.mat"
# DE Outer race 0.021 inches centered @6:00
matlab_files_name["DEOR.021@6_0"] = "234.mat"
matlab_files_name["DEOR.021@6_1"] = "235.mat"
matlab_files_name["DEOR.021@6_2"] = "236.mat"
matlab_files_name["DEOR.021@6_3"] = "237.mat"
# DE Outer race 0.021 inches centered @3:00
matlab_files_name["DEOR.021@3_0"] = "246.mat"
matlab_files_name["DEOR.021@3_1"] = "247.mat"
matlab_files_name["DEOR.021@3_2"] = "248.mat"
matlab_files_name["DEOR.021@3_3"] = "249.mat"
# DE Outer race 0.021 inches centered @12:00
matlab_files_name["DEOR.021@12_0"] = "258.mat"
matlab_files_name["DEOR.021@12_1"] = "259.mat"
matlab_files_name["DEOR.021@12_2"] = "260.mat"
matlab_files_name["DEOR.021@12_3"] = "261.mat"
# FE Inner Race 0.007 inches
matlab_files_name["FEIR.007_0"] = "278.mat"
matlab_files_name["FEIR.007_1"] = "279.mat"
matlab_files_name["FEIR.007_2"] = "280.mat"
matlab_files_name["FEIR.007_3"] = "281.mat"
# FE Ball 0.014 inches
matlab_files_name["FEB.014_0"] = "286.mat"
matlab_files_name["FEB.014_1"] = "287.mat"
matlab_files_name["FEB.014_2"] = "288.mat"
matlab_files_name["FEB.014_3"] = "289.mat"
# FE Ball 0.021 inches
matlab_files_name["FEB.021_0"] = "290.mat"
matlab_files_name["FEB.021_1"] = "291.mat"
matlab_files_name["FEB.021_2"] = "292.mat"
matlab_files_name["FEB.021_3"] = "293.mat"
# FE Outer race 0.007 inches centered @6:00
matlab_files_name["FEOR.007@6_0"] = "294.mat"
matlab_files_name["FEOR.007@6_1"] = "295.mat"
matlab_files_name["FEOR.007@6_2"] = "296.mat"
matlab_files_name["FEOR.007@6_3"] = "297.mat"
# FE Outer race 0.007 inches centered @3:00
matlab_files_name["FEOR.007@3_0"] = "298.mat"
matlab_files_name["FEOR.007@3_1"] = "299.mat"
matlab_files_name["FEOR.007@3_2"] = "300.mat"
matlab_files_name["FEOR.007@3_3"] = "301.mat"
# FE Outer race 0.007 inches centered @12:00
matlab_files_name["FEOR.007@12_0"] = "302.mat"
matlab_files_name["FEOR.007@12_1"] = "305.mat"
matlab_files_name["FEOR.007@12_2"] = "306.mat"
matlab_files_name["FEOR.007@12_3"] = "307.mat"
# FE Outer race 0.014 inches centered @3:00
matlab_files_name["FEOR.014@3_0"] = "310.mat"
matlab_files_name["FEOR.014@3_1"] = "309.mat"
matlab_files_name["FEOR.014@3_2"] = "311.mat"
matlab_files_name["FEOR.014@3_3"] = "312.mat"
# FE Outer race 0.014 inches centered @6:00
```

```
  matlab_files_name["FEOR.014@6_0"] = "313.mat"
  # FE Outer race 0.021 inches centered @6:00
  matlab_files_name["FEOR.021@6_0"] = "315.mat"
  # FE Outer race 0.021 inches centered @3:00
  matlab_files_name["FEOR.021@3_1"] = "316.mat"
  matlab_files_name["FEOR.021@3_2"] = "317.mat"
  matlab_files_name["FEOR.021@3_3"] = "318.mat"
  # DE Inner Race 0.028 inches
  matlab_files_name["DEIR.028_0"] = "3001.mat"
  matlab_files_name["DEIR.028_1"] = "3002.mat"
  matlab_files_name["DEIR.028_2"] = "3003.mat"
  matlab_files_name["DEIR.028_3"] = "3004.mat"
  # DE Ball 0.028 inches
  matlab_files_name["DEB.028_0"] = "3005.mat"
  matlab_files_name["DEB.028_1"] = "3006.mat"
  matlab_files_name["DEB.028_2"] = "3007.mat"
  matlab_files_name["DEB.028_3"] = "3008.mat"
  return matlab_files_name

"""##Download Matlab files
Downloads the Matlab files in the dictionary matlab_files_name.
"""

import urllib.request
import os.path

def download_cwrufiles(matlab_files_name):
  '''
  Downloads the Matlab files in the dictionary matlab_files_name.
  '''

url="http://csegroups.case.edu/sites/default/files/bearingdatacenter/files/Dataf
iles/"
  n = len(matlab_files_name)
  for i,key in enumerate(matlab_files_name):
    file_name = matlab_files_name[key]
    if not os.path.exists(file_name):
      urllib.request.urlretrieve(url+file_name, file_name)
    print("{}/{}\t{}\t{}".format(i+1, n, key, file_name))

"""##Extract data from Matlab files
Extracts the acquisitions of each Matlab file in the dictionary
matlab_files_name.
"""

import scipy.io
import numpy as np

def get_tensors_from_matlab(matlab_files_name):
  '''
  Extracts the acquisitions of each Matlab file in the dictionary
matlab_files_name.
  '''
  acquisitions = {}
```

```
  for key in matlab_files_name:
    file_name = matlab_files_name[key]
    matlab_file = scipy.io.loadmat(file_name)
    for position in ['DE','FE', 'BA']:
      keys = [key for key in matlab_file if key.endswith(position+"_time")]
      if len(keys)>0:
        array_key = keys[0]
        acquisitions[key+position.lower()] =
matlab_file[array_key].reshape(1,-1)[0]
  return acquisitions

"""##Downloading pickle file
Following, some auxiliary functions to download a pickle file in a google drive
account.
The pickle file already has the acquisitions properly extracted.
Therefore, these functions might speed up the whole process.
"""

import requests

def download_file_from_google_drive(id, destination):
    URL = "https://docs.google.com/uc?export=download"
    session = requests.Session()
    response = session.get(URL, params = { 'id' : id }, stream = True)
    token = get_confirm_token(response)
    if token:
        params = { 'id' : id, 'confirm' : token }
        response = session.get(URL, params = params, stream = True)
    save_response_content(response, destination)

def get_confirm_token(response):
    for key, value in response.cookies.items():
        if key.startswith('download_warning'):
            return value
    return None

def save_response_content(response, destination):
    CHUNK_SIZE = 32768
    with open(destination, "wb") as f:
        for chunk in response.iter_content(CHUNK_SIZE):
            if chunk: # filter out keep-alive new chunks
                f.write(chunk)

file_id = "1qJezMiROz9NAYafPUDPh9BFkxYF4nOi2"
destination = 'cwru.pickle'

try:
  download_file_from_google_drive(file_id, destination)
except:
  print("Download failed!")

"""##Save/Load data
If the cwru pickle file is already download, it will not be downloaded again,
and the dictionary with the acquisitions will be loaded.
```

```
Otherwise, the desired files are downloaded and the acquisitions are extrated.
"""

import pickle
import os

pickle_file = 'cwru.pickle'
if os.path.isfile(pickle_file):
  with open(pickle_file, 'rb') as handle:
    acquisitions = pickle.load(handle)
else:
  matlab_files_name = cwru_12khz()
  download_cwrufiles(matlab_files_name)
  acquisitions = get_tensors_from_matlab(matlab_files_name)
  with open(pickle_file, 'wb') as handle:
    pickle.dump(acquisitions, handle, protocol=pickle.HIGHEST_PROTOCOL)

"""##Segment data
Segments the acquisitions.
  sample_size is the size of each segment.
  max_samples is used for debug purpouses and
  reduces the number of samples from each acquisition.
"""

import numpy as np
def cwru_segmentation(acquisitions, sample_size=512, max_samples=None):
  '''
  Segments the acquisitions.
  sample_size is the size of each segment.
  max_samples is used for debug purpouses and
  reduces the number of samples from each acquisition.
  '''
  origin = []
  data = np.empty((0,sample_size,1))
  n = len(acquisitions)
  for i,key in enumerate(acquisitions):
    acquisition_size = len(acquisitions[key])
    n_samples = acquisition_size//sample_size
    if max_samples is not None and max_samples > 0 and n_samples > max_samples:
      n_samples = max_samples
    print('{}/{} --- {}: {}'.format(i+1, n, key, n_samples))
    origin.extend([key for _ in range(n_samples)])
    data = np.concatenate((data,
           acquisitions[key][:(n_samples*sample_size)].reshape(
               (n_samples,sample_size,1))))
  return data,origin

signal_data,signal_origin = cwru_segmentation(acquisitions, 512)
signal_data.shape

"""## Clean dataset functions
The functions below help to select samples from acquisitions and form groups
according to these acquisitions, using regular expressions.
"""
```

```
import re
import numpy as np

def select_samples(regex, X, y):
  '''
  Selects samples wich has some regex pattern in its name.
  '''
  mask = [re.search(regex,label) is not None for label in y]
  return X[mask],y[mask]

def join_labels(regex, y):
  '''
  Excludes some regex patterns from the labels,
  making some samples to have the same label.
  '''
  return np.array([re.sub(regex, '', label) for label in y])

def get_groups(regex, y):
  '''
  Generates a list of groups of samples with
  the same regex patten in its label.
  '''
  groups = list(range(len(y)))
  for i,label in enumerate(y):
    match = re.search(regex,label)
    groups[i] = match.group(0) if match else None
  return groups

"""##Selecting samples"""

#DE from 'de', FE from 'fe', Normal from 'de' and 'fe'
samples = '^(DE).*(de)|^(FE).*(fe)|(Normal).*'
X,y = select_samples(samples, signal_data, np.array(signal_origin))
print(len(set(y)),set(y))

"""#Experimenter"""

from sklearn.model_selection import cross_validate, KFold, PredefinedSplit

def experimenter(model, X, y, groups=None, scoring=None, cv=KFold(4, True),
verbose=0):
  '''
  Performs a experiment with some estimator (model) and validation.
  It works like a cross_validate function from sklearn, however,
  when a estimator has an internal validation with groups,
  it maintains the groups from the external validation.
  '''
  if hasattr(model,'cv') or (hasattr(model,'steps') and any(['gs' in step[0] for
step in model.steps])):
    scores = {}
    lstval = list(validation.split(X,y,groups))
    for tr,te in lstval:
      if groups is not None:
```

```
        innercv =
list(GroupShuffleKFold(validation.n_splits-1).split(X[tr],y[tr],np.array(groups)
[tr]))
      else:
        innercv = list(KFold(validation.n_splits-1, True).split(X[tr],y[tr]))
      if hasattr(model,'cv'):
        model.cv = innercv
      else:
        for step in model.steps:
          if 'gs' in step[0]:
            step[1].cv = innercv
      test_fold = np.zeros((len(y),), dtype=int)
      test_fold[tr] = -1
      score = cross_validate(model, X, y, groups, scoring,
PredefinedSplit(test_fold), verbose=verbose)
      for k in score.keys():
        if k not in scores:
          scores[k] = []
        scores[k].extend(score[k])
    return scores
  return cross_validate(model, X, y, groups, scoring, cv, verbose=verbose)

"""## Custom Splitter"""

# Commented out IPython magic to ensure Python compatibility.
from sklearn.model_selection import KFold
from sklearn.utils import shuffle
from sklearn.utils.validation import check_array
import numpy as np

class GroupShuffleKFold(KFold):
  '''
  Neither GroupShuffleSplit nor GroupKFold are good splitters for this case.
  A custom splitter must be made.
  '''
  def __init__(self, n_splits=4, shuffle=False, random_state=None):
    super().__init__(n_splits, shuffle=shuffle, random_state=random_state)
  def get_n_splits(self, X, y, groups=None):
    return self.n_splits
  def _iter_test_indices(self, X=None, y=None, groups=None):
    if groups is None:
      raise ValueError("The 'groups' parameter should not be None.")
    groups = check_array(groups, ensure_2d=False, dtype=None)
    unique_groups, groups = np.unique(groups, return_inverse=True)
    n_groups = len(unique_groups)
    if self.n_splits > n_groups:
      raise ValueError("Cannot have number of splits n_splits=%d greater"
                       " than the number of groups: %d."
#                        % (self.n_splits, n_groups))
    # Distribute groups
    indices = np.arange(n_groups)
    if self.shuffle:
      for i in range(n_groups//self.n_splits):
        if self.random_state is None:
```

```
          indices[self.n_splits*i:self.n_splits*(i+1)] = shuffle(
              indices[self.n_splits*i:self.n_splits*(i+1)])
        else:
          indices[self.n_splits*i:self.n_splits*(i+1)] = shuffle(
              indices[self.n_splits*i:self.n_splits*(i+1)],
              random_state=self.random_state+i)
    #print(unique_groups[indices]) #Debug purpose
    # Total weight of each fold
    n_samples_per_fold = np.zeros(self.n_splits)
    # Mapping from group index to fold index
    group_to_fold = np.zeros(len(unique_groups))
    # Distribute samples
    for group_index in indices:
      group_to_fold[indices[group_index]] = group_index%(self.n_splits)
    indices = group_to_fold[groups]
    for f in range(self.n_splits):
      yield np.where(indices == f)[0]

"""## BySeverity Splitter"""

# Commented out IPython magic to ensure Python compatibility.
from sklearn.model_selection import KFold
from sklearn.utils import shuffle
from sklearn.utils.validation import check_array
import numpy as np

class BySeverityKFold(KFold):
  '''
  Splits the CWRU dataset in severities.
  '''
  # Compatibility constructor
  def __init__(self, n_splits=4, shuffle=False, random_state=None):
    super().__init__(n_splits=4, shuffle=False, random_state=None)
    self.nround = random_state
  def _iter_test_indices(self, X=None, y=None, groups=None):
    if groups is None:
      raise ValueError("The 'groups' parameter should not be None.")
    groups = check_array(groups, ensure_2d=False, dtype=None)
    unique_groups, groups = np.unique(groups, return_inverse=True)
    n_groups = len(unique_groups)
    if self.n_splits > n_groups:
      raise ValueError("Cannot have number of splits n_splits=%d greater"
                       " than the number of groups: %d."
#                        % (self.n_splits, n_groups))
    # Distribute groups
    indices = np.arange(n_groups)
    nround = self.nround - random_state
    for i in range(self.n_splits):
      indices[i+self.n_splits] = (i+nround)%self.n_splits+self.n_splits
    #print(unique_groups[indices]) #Debug purpose
    # Total weight of each fold
    n_samples_per_fold = np.zeros(self.n_splits)
    # Mapping from group index to fold index
    group_to_fold = np.zeros(len(unique_groups))
```

```
    # Distribute samples
    for group_index in indices:
      group_to_fold[indices[group_index]] = group_index%(self.n_splits)
    indices = group_to_fold[groups]
    for f in range(self.n_splits):
      yield np.where(indices == f)[0]

"""##Experiment setup"""

from collections import namedtuple

ExperimentSetup = namedtuple('ExperimentSetup', 'groups, splitter_name,
shuffle')

validations = {
    # Validation usually seen in publications with CWRU bearing dataset.
    "Usual K-Fold": ExperimentSetup(groups = None,
                                    splitter_name = 'KFold',
                                    shuffle = True),
    # Samples with the same load cannot be in the trainning fold and
    # the test folds simultaneously.
    "By Load": ExperimentSetup(groups = get_groups('_\d',y),
                               splitter_name = 'GroupShuffleKFold',
                               shuffle = False),
    # Samples with the same severity cannot be in the trainning folds and
    # the test folds simultaneously.
    "By Severity": ExperimentSetup(groups =
get_groups('(\.\d{3})|(Normal_\d)',y),
                                   splitter_name = 'BySeverityKFold',
                                   shuffle = False),
}

"""##Common Variables"""

# Only four conditions are considered: Normal, Ball, Inner Race and Outer Race.
selected_y = join_labels('_\d|@\d{1,3}|(de)|(fe)|\.\d{3}|(DE)|(FE)',y)
rounds = 4 if not debug else 1
verbose = 3
random_state = 42
scoring = ['accuracy', 'f1_macro']#, 'precision_macro', 'recall_macro']

"""#Classification Models"""

import warnings
warnings.filterwarnings('ignore')

"""##Feature Extraction Models"""

from sklearn.base import TransformerMixin

"""###Statistical functions"""

import numpy as np
import scipy.stats as stats
```

```
def rms(x):
  '''
  root mean square
  '''
  x = np.array(x)
  return np.sqrt(np.mean(np.square(x)))

def sra(x):
  '''
  square root amplitude
  '''
  x = np.array(x)
  return np.mean(np.sqrt(np.absolute(x)))**2

def ppv(x):
  '''
  peak to peak value
  '''
  x = np.array(x)
  return np.max(x)-np.min(x)

def cf(x):
  '''
  crest factor
  '''
  x = np.array(x)
  return np.max(np.absolute(x))/rms(x)

def ifa(x):
  '''
  impact factor
  '''
  x = np.array(x)
  return np.max(np.absolute(x))/np.mean(np.absolute(x))

def mf(x):
  '''
  margin factor
  '''
  x = np.array(x)
  return np.max(np.absolute(x))/sra(x)

def sf(x):
  '''
  shape factor
  '''
  x = np.array(x)
  return rms(x)/np.mean(np.absolute(x))

def kf(x):
  '''
  kurtosis factor
  '''
```

```
  x = np.array(x)
  return stats.kurtosis(x)/(np.mean(x**2)**2)

"""### Statistical Features from Time Domain"""

class StatisticalTime(TransformerMixin):
  '''
  Extracts statistical features from the time domain.
  '''
  def fit(self, X, y=None):
    return self
  def transform(self, X, y=None):
    return np.array([
                     [
                      rms(x), # root mean square
                      sra(x), # square root amplitude
                      stats.kurtosis(x), # kurtosis
                      stats.skew(x), # skewness
                      ppv(x), # peak to peak value
                      cf(x), # crest factor
                      ifa(x), # impact factor
                      mf(x), # margin factor
                      sf(x), # shape factor
                      kf(x), # kurtosis factor
                      ] for x in X[:,:,0]
                     ])

"""### Statistical Features from Frequency Domain"""

class StatisticalFrequency(TransformerMixin):
  '''
  Extracts statistical features from the frequency domain.
  '''
  def fit(self, X, y=None):
    return self
  def transform(self, X, y=None):
    sig = []
    for x in X[:,:,0]:
      fx = np.absolute(np.fft.fft(x)) # transform x from time to frequency
domain
      fc = np.mean(fx) # frequency center
      sig.append([
                  fc, # frequency center
                  rms(fx), # RMS from the frequency domain
                  rms(fx-fc), # Root Variance Frequency
                  ])
    return np.array(sig)

"""###Statistical Features"""

class Statistical(TransformerMixin):
  '''
  Extracts statistical features from both time and frequency domain.
  '''
```

```
  def fit(self, X, y=None):
    return self
  def transform(self, X, y=None):
    st = StatisticalTime()
    stfeats = st.transform(X)
    sf = StatisticalFrequency()
    sffeats = sf.transform(X)
    return np.concatenate((stfeats,sffeats),axis=1)

"""###Wavelet Package Features"""

import numpy as np
import pywt

class WaveletPackage(TransformerMixin):
  '''
  Extracts Wavelet Package features.
  The features are calculated by the energy of the recomposed signal
  of the leaf nodes coeficients.
  '''
  def fit(self, X, y=None):
    return self
  def transform(self, X, y=None):
    def Energy(coeffs, k):
      return np.sqrt(np.sum(np.array(coeffs[-k]) ** 2)) / len(coeffs[-k])
    def getEnergy(wp):
      coefs = np.asarray([n.data for n in wp.get_leaf_nodes(True)])
      return np.asarray([Energy(coefs,i) for i in range(2**wp.maxlevel)])
    return np.array([getEnergy(pywt.WaveletPacket(data=x, wavelet='db4',
                                                  mode='symmetric', maxlevel=4)
                                                  ) for x in X[:,:,0]])

"""###Heterogeneus Features"""

class Heterogeneous(TransformerMixin):
  '''
  Mixes Statistical and Wavelet Package features.
  '''
  def fit(self, X, y=None):
    return self
  def transform(self, X, y=None):
    st = StatisticalTime()
    stfeats = st.transform(X)
    sf = StatisticalFrequency()
    sffeats = sf.transform(X)
    wp = WaveletPackage()
    wpfeats = wp.transform(X)
    return np.concatenate((stfeats,sffeats,wpfeats),axis=1)

"""##K-NN with Heterogeneous Features"""

from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.model_selection import GridSearchCV

knn = Pipeline([
                ('FeatureExtraction', Heterogeneous()),
                ('scaler', StandardScaler()),
                ('knn', KNeighborsClassifier()),
                ])

parameters = {'knn__n_neighbors': list(range(1,16,2))}
if not debug:
  knn = GridSearchCV(knn, parameters, verbose=verbose)
knn

"""##SVM with Heterogeneous Features"""

from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV

svm = Pipeline([
                ('FeatureExtraction', Heterogeneous()),
                ('scaler', StandardScaler()),
                ('svc', SVC()),
                ])

parameters = {
    'svc__C': [10**x for x in range(-3,2)],
    'svc__gamma': [10**x for x in range(-3,1)],
    }
if not debug:
  svm = GridSearchCV(svm, parameters, verbose=verbose)
svm

"""##Random Forest with Heterogeneous Features"""

from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV

rf = Pipeline([
               ('FeatureExtraction', Heterogeneous()),
               ('scaler', StandardScaler()),
               ('rf', RandomForestClassifier()),
               ])

parameters = {
    "rf__n_estimators": [10, 20, 50, 100, 200, 500],
    "rf__max_features": list(range(1,21)),
    }
if not debug:
  rf = GridSearchCV(rf, parameters, verbose=verbose)
rf
```

```
"""##Convolutional Neural Network"""

# Commented out IPython magic to ensure Python compatibility.
# %tensorflow_version 2.x

"""###F1-score macro averaged implemented for Keras"""

from keras import backend as K
import tensorflow as tf

def f1_score_macro(y_true,y_pred):
    def recall(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
        recall = true_positives / (possible_positives + K.epsilon())
        return recall
    def precision(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision
    precision = precision(y_true, y_pred)
    recall = recall(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))

"""###ANN wrapped in a scikit-learn estimator."""

from keras import layers
from keras.models import Sequential
from keras.utils import to_categorical
from sklearn.base import BaseEstimator, ClassifierMixin
import numpy as np
from keras.callbacks import EarlyStopping,ReduceLROnPlateau

class ANN(BaseEstimator, ClassifierMixin):
  def __init__(self,
               dense_layer_sizes=[64],
               kernel_size=32,
               filters=32,
               n_conv_layers=2,
               pool_size=8,
               dropout=0.25,
               epochs=50,
               validation_split=0.05,
               optimizer='sgd'#'nadam'#'rmsprop'#
               ):
    self.dense_layer_sizes = dense_layer_sizes
    self.kernel_size = kernel_size
    self.filters = filters
    self.n_conv_layers = n_conv_layers
    self.pool_size = pool_size
    self.dropout = dropout
    self.epochs = epochs
```

```
    self.validation_split = validation_split
    self.optimizer = optimizer

  def fit(self, X, y=None):
    dense_layer_sizes = self.dense_layer_sizes
    kernel_size = self.kernel_size
    filters = self.filters
    n_conv_layers = self.n_conv_layers
    pool_size = self.pool_size
    dropout = self.dropout
    epochs = self.epochs
    optimizer = self.optimizer
    validation_split = self.validation_split

    self.labels, ids = np.unique(y, return_inverse=True)
    y_cat = to_categorical(ids)
    num_classes = y_cat.shape[1]

    self.model = Sequential()
    self.model.add(layers.InputLayer(input_shape=(X.shape[1],X.shape[-1])))
    for _ in range(n_conv_layers):
      self.model.add(layers.Conv1D(filters, kernel_size))#, padding='valid'))
      self.model.add(layers.Activation('relu'))
      if pool_size>1:
        self.model.add(layers.MaxPooling1D(pool_size=pool_size))
    #self.model.add(layers.Dropout(0.25))
    self.model.add(layers.Flatten())
    for layer_size in dense_layer_sizes:
        self.model.add(layers.Dense(layer_size))
        self.model.add(layers.Activation('relu'))
    if dropout>0 and dropout<1:
      self.model.add(layers.Dropout(dropout))
    self.model.add(layers.Dense(num_classes))
    self.model.add(layers.Activation('softmax'))
    self.model.compile(loss='categorical_crossentropy',
                       optimizer=optimizer,
                       metrics=[f1_score_macro])
    if validation_split>0 and validation_split<1:
      prop = int(1/validation_split)
      mask = np.array([i%prop==0 for i in range(len(y))])
      self.history = self.model.fit(X[~mask], y_cat[~mask], epochs=epochs,
                                    validation_data=(X[mask],y_cat[mask]),
                                    callbacks=[EarlyStopping(patience=3),
ReduceLROnPlateau()],
                                    verbose=False
                                    )
    else:
      self.history = self.model.fit(X, y_cat, epochs=epochs, verbose=False)

  def predict_proba(self, X, y=None):
    return self.model.predict(X)

  def predict(self, X, y=None):
    predictions = self.model.predict(X)
```

```
    return self.labels[np.argmax(predictions,axis=1)]

"""###ANN instantiation"""

parameters = {
    'filters': [16, 32],
    'kernel_size': [16, 32],
    'n_conv_layers': [1, 2],
    #'pool_size': [2, 4, 6, 8],
    }
ann = ANN()
if not debug:
  ann = GridSearchCV(ann, parameters, verbose=verbose)
ann

"""##List of Estimators"""

clfs = [
        ('KNN - KNeighborsClassifier, Heterogeneous Features', knn),
        ('SVM - SVC with Heterogeneous Features', svm),
        ('RF - RandomForestClassifier with Heterogeneous Features', rf),
        ('ANN - Artificial Neural Network with Convolutional Layers', ann),
        ]
if debug:
  clfs = [
          ('KNN - KNeighborsClassifier, Heterogeneous Features', knn),
      ]

"""#Performing Experiments"""

import numpy as np

scores = {}
trtime = {}
tetime = {}
# Number of repetitions
for r in range(rounds):
  round_str = "Round {}".format(r+1)
  print("@"*(len(round_str)+8),'\n@@@',round_str,'@@@\n'+"@"*(len(round_str)+8))
  # Estimators
  for clf_name, estimator in clfs:
    if clf_name not in scores:
      scores[clf_name] = {}
      trtime[clf_name] = {}
      tetime[clf_name] = {}
    print("*"*(len(clf_name)+8),'\n***',clf_name,'***\n'+"*"*(len(clf_name)+8))
    # Validation forms
    for val_name in validations.keys():

print("#"*(len(val_name)+8),'\n###',val_name,'###\n'+"#"*(len(val_name)+8))
      groups = validations[val_name].groups
      if val_name not in scores[clf_name]:
        scores[clf_name][val_name] = {}
      validation = eval(validations[val_name].splitter_name
```

```
                        +'(4,shuffle='+str(validations[val_name].shuffle)
                        +',random_state='+str(random_state+r)+')')
      score = experimenter(estimator, X, selected_y, groups,
                           scoring, validation, verbose)
      for metric,s in score.items():
        print(metric, ' \t', s)
        if metric not in scores[clf_name][val_name]:
          scores[clf_name][val_name][metric] = []
        scores[clf_name][val_name][metric].append(s)

"""##Save results"""

clf = {}
val = {}
src = {}
for c, clf_name in enumerate(scores.keys()):
  if c not in clf:
    clf[c] = clf_name
  for v, val_name in enumerate(scores[clf_name].keys()):
    if v not in val:
      val[v] = val_name
    for s, scr_name in enumerate(scores[clf_name][val_name].keys()):
      scores[clf_name][val_name][scr_name] =
np.array(scores[clf_name][val_name][scr_name])
      if s not in src:
        src[s] = scr_name
      np.savetxt('{}-{}-{}.txt'.format(clf_name,val_name,scr_name),
                 scores[clf_name][val_name][scr_name], delimiter=',')
      print('{} - {} - {}\n'.format(clf_name.split('-')[0],val_name,scr_name),
            scores[clf_name][val_name][scr_name])

"""##Average & Standard Deviation"""

c,v,s = len(clf),len(val),len(src)
for i in range(s):
  print(src[i])
  for k in range(v):
    print('\t'+val[k]+' ', end='')
  print()
  for j in range(c):
    print(clf[j].split('-')[0], end='\t')
    for k in range(v):
      print("{0:.3f} ({1:.3f})".format(
          scores[clf[j]][val[k]][src[i]].mean(),
          scores[clf[j]][val[k]][src[i]].std()), end='\t')
    print()
  print()

"""##Average of diferences"""

c,v,s = len(clf),len(val),len(src)
compclf = np.zeros((s,v,c,c))
for i in range(s):
  print('*'*3, src[i], '*'*3)
```

```
  for j in range(v):
    print(val[j])
    for k in range(c):
      print('       '+clf[k].split('-')[0],end=' ')
    print()
    for k in range(c):
      for l in range(k):
        diff = scores[clf[k]][val[j]][src[i]]-scores[clf[l]][val[j]][src[i]]
        compclf[i][j][l][k] = diff.mean()
        compclf[i][j][k][l] = diff.std()
    print(compclf[i][j])

"""## Experiment results

```
@@@@@@@@@@@@@@@
@@@ Round 1 @@@
@@@@@@@@@@@@@@@
****
*** KNN - KNeighborsClassifier, Heterogeneous Features ***
****
####################
Usual K-Fold
####################
fit_time   [727.96452045 727.97627831 727.50943279 728.06793976]
score_time   [21.74185729 21.86552668 23.3405931 22.64544058]
test_accuracy   [0.9747494 0.97525381 0.97538071 0.97525381]
test_f1_macro   [0.97458885 0.97497874 0.97490906 0.97473992]
###############
By Load
###############
fit_time   [828.86680245 927.02772117 329.38242483 727.82459092]
score_time   [21.75594425 22.4729023 23.60786653 22.59058714]
test_accuracy   [0.94216113 0.96138565 0.94352036 0.94121295]
test_f1_macro   [0.94776085 0.95997728 0.94237088 0.93930294]
###################
By Severity
###################
fit_time   [1024.82461357 927.59816742 826.22390318 932.62575102]
score_time   [32.55097294 24.10155511 29.09020591 11.41447496]
test_accuracy   [0.57375479 0.47364381 0.52315645 0.5034337]
test_f1_macro   [0.63540862 0.4257751 0.53825359 0.25568118]
****
*** SVM - SVC with Heterogeneous Features ***
****
####################
Usual K-Fold
####################
fit_time   [634.13174486 634.51081061 633.77368164 633.90333509]
score_time   [22.51778221 23.25896406 22.64851904 22.43799543]
test_accuracy   [0.93706382 0.94352792 0.94403553 0.94822335]
test_f1_macro   [0.93864646 0.94433932 0.94512301 0.94928024]
###############
By Load
```
```

```
###############
fit_time         [734.78695059 736.82255578 734.65432668 736.11744142]
score_time       [21.33996463 23.88718414 24.54262042 25.35802913]
test_accuracy    [0.91820904 0.95021108 0.93247269 0.89606846]
test_f1_macro    [0.93083371 0.95009428 0.93033798 0.89188541]
###################
### By Severity ###
###################
fit_time         [930.91943145 834.79969454 830.8039732  942.56819558]
score_time       [31.03858352 22.16025257 28.52099633 11.84618878]
test_accuracy    [0.53429119 0.4712129  0.43622745 0.50079239]
test_f1_macro    [0.61608375 0.4501465  0.43970497 0.25      ]
************************************************************
*** RF - RandomForestClassifier with Heterogeneous Features ***
************************************************************
###################
### Usual K-Fold ###
###################
fit_time         [630.50709295 629.12646198 631.78579044 630.95092249]
score_time       [18.65195918 20.58970332 21.8814013  19.07876348]
test_accuracy    [0.98185509 0.98324873 0.98020305 0.98109137]
test_f1_macro    [0.98218983 0.983292   0.98079253 0.98137378]
###############
### By Load ###
###############
fit_time         [732.56729364 729.24327779 730.02082229 729.38728404]
score_time       [18.71820569 19.45225668 20.02794385 18.83754015]
test_accuracy    [0.93045727 0.97429849 0.95866435 0.94170904]
test_f1_macro    [0.93767059 0.97477395 0.95880642 0.94057358]
###################
### By Severity ###
###################
fit_time         [926.311553   931.59828091 828.1641283  835.10836077]
score_time       [25.74145246 20.01617384 23.47455072  9.86901832]
test_accuracy    [0.52337165 0.54119754 0.54552168 0.50079239]
test_f1_macro    [0.57882523 0.50779217 0.53005161 0.25      ]
************************************************************
*** ANN - Artificial Neural Network with Convolutional Layers ***
************************************************************
###################
### Usual K-Fold ###
###################
fit_time         [265.39738965 343.36738896 257.33688712 312.03943443]
score_time       [2.25397229 2.24209499 2.1947124  2.18814898]
test_accuracy    [0.99314808 0.99111675 0.9857868  0.99340102]
test_f1_macro    [0.99311445 0.99110972 0.98635947 0.99345478]
###############
### By Load ###
###############
fit_time         [281.06889892 302.09401894 184.59534931 171.34777737]
score_time       [2.04453182 2.21507621 2.46740127 2.26294756]
test_accuracy    [0.91412629 0.98361063 0.97641509 0.89123155]
test_f1_macro    [0.92638134 0.98345571 0.97631351 0.88853306]
###################
```

```
### By Severity ###
####################
fit_time         [1241.76646113 1156.54137278 1171.15828347 1216.10608244]
score_time       [2.99997783 2.21008921 2.76161528 1.15192008]
test_accuracy    [0.64061303 0.62282497 0.5129233  0.50924459]
test_f1_macro    [0.70157408 0.55756989 0.52996687 0.26639344]
@@@@@@@@@@@@@@@
@@@ Round 2 @@@
@@@@@@@@@@@@@@@
************************************************************
*** KNN - KNeighborsClassifier, Heterogeneous Features ***
************************************************************
####################
### Usual K-Fold ###
####################
fit_time         [629.42630696 629.16001964 628.83887672 629.19791722]
score_time       [22.93167543 23.89310408 23.46724343 23.36352086]
test_accuracy    [0.97741403 0.97461929 0.97360406 0.9751269 ]
test_f1_macro    [0.97693886 0.97444341 0.97327064 0.97452677]
###############
### By Load ###
###############
fit_time         [730.09751177 728.87677646 828.99845076 629.02737379]
score_time       [22.88180923 23.71258712 23.7504859  24.09755874]
test_accuracy    [0.94364413 0.95978565 0.94784036 0.94632295]
test_f1_macro    [0.94776085 0.95897728 0.94747088 0.93630294]
####################
### By Severity ###
####################
fit_time         [925.00413346 829.43727756 827.07459569 935.39933348]
score_time       [36.88735509 24.09057665 30.52437091  8.70172977]
test_accuracy    [0.60801547 0.47364381 0.52145493 0.33919775]
test_f1_macro    [0.63550027 0.4257751  0.53640962 0.25626305]
******************************************
*** SVM - SVC with Heterogeneous Features ***
******************************************
####################
### Usual K-Fold ###
####################
fit_time         [835.2218082  634.93261027 634.73015285 733.7497735 ]
score_time       [23.86916852 22.41502929 22.72918797 22.51176977]
test_accuracy    [0.94201243 0.94022843 0.94428934 0.94479695]
test_f1_macro    [0.94242206 0.94189934 0.94596372 0.94544177]
###############
### By Load ###
###############
fit_time         [733.88335633 733.84748316 834.84780931 834.7111752 ]
score_time       [20.56999111 23.90906286 24.26909876 23.93199968]
test_accuracy    [0.91820904 0.95021108 0.93247269 0.89606846]
test_f1_macro    [0.93657371 0.95281428 0.93749798 0.89370541]
####################
### By Severity ###
####################
fit_time         [928.65935802 933.67394733 1030.56625867 843.2254045 ]
```

```
score_time       [32.82621431 22.52077532 26.59487867  8.56908464]
test_accuracy    [0.57250835 0.4712129  0.43626779 0.33497537]
test_f1_macro    [0.61592054 0.4501465  0.43927772 0.25      ]
************************************************************
*** RF - RandomForestClassifier with Heterogeneous Features ***
************************************************************
####################
### Usual K-Fold ###
####################
fit_time         [830.04664826 830.01473355 930.0835495  730.20023632]
score_time       [19.55071712 19.56467962 19.53575635 19.39014673]
test_accuracy    [0.98375841 0.98096447 0.98109137 0.98350254]
test_f1_macro    [0.98384676 0.981425   0.98154185 0.98355207]
###############
### By Load ###
###############
fit_time         [30.82257223 29.92996097 31.30528164 28.42498422]
score_time       [18.27313447 20.16906404 18.67306423 18.99323606]
test_accuracy    [0.93072945 0.96933201 0.95891261 0.93116706]
test_f1_macro    [0.93791145 0.96985915 0.95927812 0.92996408]
####################
### By Severity ###
####################
fit_time         [1024.90739155 1028.32126331 1027.67100358 936.90829897]
score_time       [26.63776398 18.32003856 22.30930877  7.41716528]
test_accuracy    [0.57813324 0.54094166 0.55740643 0.34799437]
test_f1_macro    [0.5589154  0.5097983  0.53808361 0.26881994]
************************************************************
*** ANN - Artificial Neural Network with Convolutional Layers ***
************************************************************
####################
### Usual K-Fold ###
####################
fit_time         [1215.05489302 1258.50468445 1300.08051682 1305.45215034]
score_time       [2.24699092 2.2769115  2.30683041 2.35071349]
test_accuracy    [0.9717041  0.97969543 0.98134518 0.99340102]
test_f1_macro    [0.97224652 0.98041412 0.98167097 0.99358593]
###############
### By Load ###
###############

fit_time         [6300.04461122 6248.71587634 6178.11567807 6168.27698874]
score_time       [2.1462605  2.31680465 2.31082058 2.27990294]
test_accuracy    [0.91004355 0.98236901 0.97219464 0.84645913]
test_f1_macro    [0.92412493 0.98212308 0.97206877 0.83215565]
####################
### By Severity ###
####################
fit_time         [8283.32233214 9306.67986798 10196.27611208 9352.24302053]
score_time       [3.25828552 2.22704434 2.75662851 0.9025867 ]
test_accuracy    [0.66373704 0.58354657 0.4558777  0.34201267]
test_f1_macro    [0.6945503  0.51831185 0.50062502 0.26037344]
@@@@@@@@@@@@@@@
@@@ Round 3 @@@
```

```
@@@@@@@@@@@@@@
************************************************************
*** KNN - KNeighborsClassifier, Heterogeneous Features ***
************************************************************
####################
### Usual K-Fold ###
####################
fit_time         [629.26773095 629.31360865 629.26673388 629.03834438]
score_time       [23.16505194 23.40939784 23.53506184 23.7255528 ]
test_accuracy    [0.97601827 0.97728426 0.97233503 0.9751269 ]
test_f1_macro    [0.97541772 0.97630753 0.97237825 0.97472236]
###############
### By Load ###
###############
fit_time         [729.79731536 728.87577915 729.00244021 729.14007306]
score_time       [23.09822917 24.18931293 24.10952544 24.07561636]
test_accuracy    [0.94414713 0.96841565 0.94529036 0.95471295]
test_f1_macro    [0.94796085 0.95097728 0.94937088 0.93330294]
####################
### By Severity ###
####################
fit_time         [925.38511395 929.59884501 929.01839781 834.32620335]
score_time       [36.66295552 24.13246393 27.9462657  11.05842733]
test_accuracy    [0.60801547 0.47660445 0.46704133 0.5031746 ]
test_f1_macro    [0.63550027 0.43079052 0.53541346 0.25626305]
******************************************
*** SVM - SVC with Heterogeneous Features ***
******************************************
####################
### Usual K-Fold ###
####################
fit_time         [735.61276245 635.21482706 835.63974071 833.70583415]
score_time       [23.9359901  23.50635934 23.48033381 23.65374613]
test_accuracy    [0.9454384  0.9430203  0.94568528 0.93819797]
test_f1_macro    [0.94750302 0.9431637  0.9474098  0.93862111]
###############
### By Load ###
###############
fit_time         [835.9897542  833.74575543 833.22016144 832.85314345]
score_time       [20.71261001 23.29670024 23.34157991 23.49719739]
test_accuracy    [0.93584904 0.91287108 0.93487269 0.89147868]
test_f1_macro    [0.93083371 0.950089428 0.93983798 0.89637541]
####################
### By Severity ###
####################
fit_time         [926.86516309 1033.26109409 829.92400503 839.44653893]
score_time       [31.47480106 21.50245571 23.40238786 10.77819586]
test_accuracy    [0.57250835 0.47430325 0.37419506 0.5       ]
test_f1_macro    [0.61592054 0.45334101 0.43932885 0.25      ]
************************************************************
*** RF - RandomForestClassifier with Heterogeneous Features ***
************************************************************
####################
### Usual K-Fold ###
```

```
####################
fit_time         [928.86680388 928.93658853 928.18163538 928.47482038]
score_time       [17.94404149 18.95231724 18.79476714 18.5354321 ]
test_accuracy    [0.98388529 0.98185279 0.98388325 0.98172589]
test_f1_macro    [0.98401225 0.98190344 0.9844736  0.98174234]
###############
### By Load ###
###############
fit_time         [1029.04931498 1030.37875891 1029.8541286  929.71453595]
score_time       [16.96164083 19.98056746 19.92571425 20.28774595]
test_accuracy    [0.93440392 0.97318103 0.96499503 0.93860846]
test_f1_macro    [0.94094662 0.97363677 0.9650458  0.93754191]
####################
### By Severity ###
####################
fit_time         [1225.54768014 1330.02770019 1029.36347532 1135.58084846]
score_time       [28.2614212  19.61155343 21.05469537  9.39487553]
test_accuracy    [0.5546669  0.52902071 0.51340592 0.5       ]
test_f1_macro    [0.5542015  0.49779154 0.54168278 0.25      ]
************************************************************
*** ANN - Artificial Neural Network with Convolutional Layers ***
************************************************************
####################
### Usual K-Fold ###
####################
fit_time         [1372.03110313 1239.42572045 1320.26054955 1243.22156954]
score_time       [2.28090024 2.23901248 2.24200392 2.24798822]
test_accuracy    [0.9949245  0.98413706 0.99505076 0.99213198]
test_f1_macro    [0.9949436  0.98422413 0.99504654 0.99211871]
###############
### By Load ###
###############
fit_time         [1357.42702127 1292.5616231   182.50835323 1286.55269241]
score_time       [2.10536981 2.46740222 2.40656376 2.34572673]
test_accuracy    [0.89316821 0.97591259 0.93445879 0.92459382]
test_f1_macro    [0.91092018 0.97618376 0.93565673 0.92624242]
####################
### By Severity ###
####################
fit_time         [9170.25469923 8183.34170198 10340.18925595 10197.81898642]
score_time       [3.22836661 2.28090096 2.43149757 1.14793038]
test_accuracy    [0.65600281 0.59409358 0.43133123 0.53439153]
test_f1_macro    [0.68699843 0.52459393 0.50629582 0.31052142]
@@@@@@@@@@@@@@@@
@@@ Round 4 @@@
@@@@@@@@@@@@@@@@
************************************************************
*** KNN - KNeighborsClassifier, Heterogeneous Features ***
************************************************************
####################
### Usual K-Fold ###
####################
fit_time         [629.4302969  629.1739819  629.33953881 729.23581624]
score_time       [23.18898678 23.56996799 23.73851728 23.35155201]
```

```
test_accuracy    [0.97297297 0.97601523 0.97208122 0.97436548]
test_f1_macro    [0.97259117 0.97591143 0.97168737 0.97409023]
###############
### By Load ###
###############
fit_time         [829.98381567 829.49911261 829.58189154 828.94858432]
score_time       [22.95461392 23.72455502 23.73851848 24.23119998]
test_accuracy    [0.95784113 0.94624565 0.9657036 0.94748295]
test_f1_macro    [0.94486085 0.95397728 0.94987088 0.93317894]
###################
### By Severity ###
###################
fit_time         [924.74582338 930.85847735 927.085567   934.19854403]
score_time       [36.73875308 22.05501962 30.66699004 11.23695016]
test_accuracy    [0.60901265 0.40127944 0.52315645 0.5031746 ]
test_f1_macro    [0.6361736  0.41508396 0.53825359 0.25626305]
*****************************************
*** SVM - SVC with Heterogeneous Features ***
*****************************************
####################
### Usual K-Fold ###
####################
fit_time         [835.78729725 735.51303029 833.92028904 834.49275875]
score_time       [24.30599952 24.00680065 22.5147903  23.24384069]
test_accuracy    [0.94366197 0.94796954 0.94098985 0.94263959]
test_f1_macro    [0.9446449  0.94903213 0.9417423  0.94402473]
###############
### By Load ###
###############
fit_time         [936.02565861 935.68856049 935.68756366 934.79694557]
score_time       [22.29537702 24.60819149 24.52441645 24.30699682]
test_accuracy    [0.94890904 0.91264108 0.93879269 0.90369746]
test_f1_macro    [0.93183371 0.95065828 0.93367198 0.89934741]
###################
### By Severity ###
###################
fit_time         [1228.7211926  1035.32752681 1130.84950113 1942.00965643]
score_time       [33.1453619  19.78608823 26.78738618 11.4932642 ]
test_accuracy    [0.57229445 0.39968014 0.43622745 0.5       ]
test_f1_macro    [0.61568042 0.44688637 0.43970497 0.25      ]
************************************************************
*** RF - RandomForestClassifier with Heterogeneous Features ***
************************************************************
####################
### Usual K-Fold ###
####################
fit_time         [930.64704251 830.19624782 830.08252478 830.73979378]
score_time       [19.41906929 19.44001341 19.80204415 19.70231223]
test_accuracy    [0.97868291 0.98502538 0.98236041 0.98236041]
test_f1_macro    [0.97902816 0.98537647 0.98266146 0.98266208]
###############
### By Load ###
###############
fit_time         [930.69391727 929.98680878 929.08621645 928.32634568]
```

```
score_time       [18.10558081 19.88582087 18.36588526 20.02744102]
test_accuracy    [0.93168209 0.9729327  0.96437438 0.93054694]
test_f1_macro    [0.93846529 0.97350272 0.96445557 0.92917187]
####################
### By Severity ###
####################
fit_time         [825.22556782 830.48647285 927.24511838 933.68691349]
score_time       [26.65269613 15.5514338  22.38114738  8.68480301]
test_accuracy    [0.5633345  0.47266647 0.52874776 0.50132275]
test_f1_macro    [0.54704957 0.50247643 0.52085485 0.25262881]
**************************************************************
*** ANN - Artificial Neural Network with Convolutional Layers ***
**************************************************************
####################
### Usual K-Fold ###
####################

fit_time         [916.94281554 952.95212412 958.95150447 973.91706014]
score_time       [2.27491689 2.36766815 2.31480932 2.30782795]
test_accuracy    [0.99454384 0.99695431 0.99098985 0.99682741]
test_f1_macro    [0.99441217 0.99704486 0.99102743 0.99680962]
###############
### By Load ###
###############
fit_time         [978.76095223 910.31698728 902.54876089 1139.71636653]
score_time       [2.13628721 2.34373212 2.3158071  2.41155124]
test_accuracy    [0.90174197 0.93407003 0.9097567  0.87300012]
test_f1_macro    [0.9167921  0.9322561  0.91430151 0.86684068]
####################
### By Severity ###
####################
fit_time         [9265.11202931 8345.81909561 10194.14838052 12193.98420715]
score_time       [3.50063896 2.20668817 2.92418027 1.18837595]
test_accuracy    [0.662948   0.58548997 0.55491086 0.52433862]
test_f1_macro    [0.6902018  0.56390606 0.58945617 0.29440154]

```
"""
```
```