UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Ricardo de Souza Monteiro

Uma Abordagem de Teste Baseado em Propriedades para Modelos Clássicos de Aprendizado de Máquina

São João del-Rei

2023

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Ricardo de Souza Monteiro

Uma Abordagem de Teste Baseado em Propriedades para Modelos Clássicos de Aprendizado de Máquina

Dissertação apresentada como requisito para obtenção do título de mestre em Ciências no Curso de Mestrado do Programa de Pós Graduação em Ciência da Computação da UFSJ.

Orientador: Vinícius Humberto Serapilha Durelli

Universidade Federal de São João del-Rei – UFSJ

Mestrado em Ciência da Computação

São João del-Rei

2023

Ficha catalográfica elaborada pela Divisão de Biblioteca (DIBIB)
e Núcleo de Tecnologia da Informação (NTINF) da UFSJ,
com os dados fornecidos pelo(a) autor(a)

M775a

Monteiro, Ricardo de Souza.
Uma Abordagem de Teste Baseado em Propriedades para Modelos Clássicos de Aprendizado de Máquina / Ricardo de Souza Monteiro ; orientador Vinícius Humberto Serapilha Durelli. -- São João del-Rei, 2023.
145 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciência da Computação) -- Universidade Federal de São João del-Rei, 2023.

1. Teste de Software. 2. Teste Baseado em Propriedades. 3. Teste de Sistemas Baseados em Aprendizado de Máquina. 4. Teste de Modelos de Aprendizado de Máquina. I. Durelli, Vinícius Humberto Serapilha, orient. II. Título.

Ricardo de Souza Monteiro

Uma Abordagem de Teste Baseado em Propriedades para Modelos Clássicos de Aprendizado de Máquina

Dissertação apresentada como requisito para obtenção do título de mestre em Ciências no Curso de Mestrado do Programa de Pós Graduação em Ciência da Computação da UFSJ.

Trabalho aprovado. São João del-Rei, 31 de março de 2023:

Vinícius Humberto Serapilha Durelli
Orientador

André Takeshi Endo
Membro Externo

Elder José Reioli Cirilo
Membro Interno

São João del-Rei
2023

Este trabalho é dedicado aos meus pais Iraci e Cirdinei,
a minha irmã Yara,
aos meus avôs Sebastião Bonifácio (in memoriam) e Elman Tarcísio (in memoriam),
às minhas avós Maria de Lourdes (in memoriam), Geralda (in memoriam) e Nadir,
às minhas tias Carmen (in memoriam) e Helena,
e ao meu afilhado João Vítor.

Agradecimentos

Agradeço aos meus pais Iraci e Cirdinei por todo amor oferecido e por toda dedicação empenhada na criação dos seus filhos. Serei eternamente grato por vocês terem feito o melhor que puderam para oferecer aos seus filhos a oportunidade de estudar, de obter uma formação, e principalmente, de tornarem-se boas pessoas. Agradeço a minha irmã Yara pela amizade e ajuda oferecida durante toda a vida. Agradeço ao meu avô Sebastião Bonifácio *(in memoriam)* pelo exemplo de caráter e bom coração. Agradeço a minha avó Maria de Lourdes *(in memoriam)*, a minha bisavó Geralda *(in memoriam)* e a minha tia Carmen *(in memoriam)* pelos exemplos de força e perseverança. Agradeço ao meu avô Elman Tarcísio *(in memoriam)* e a minha avó Nadir por sempre terem acreditado em seus netos. Agradeço ao meu afilhado João Vítor pela amizade e momentos de diversão. Agradeço a minha tia Helena por em vários momentos ter sido como uma segunda mãe. Também agradeço aos demais membros da minha família pela convivência nos momentos felizes e pela ajuda nos momentos de dificuldades.

Agradeço aos colegas e amigos que convivi durante a graduação e mestrado em Ciência da Computação da Universidade Federal de São João del-Rei. Agradeço aos professores do Departamento de Ciência da Computação da Universidade Federal de São João del-Rei por todo aprendizado oferecido durante esses anos. Em especial, agradeço ao professor Dr. Vinícius Humberto Serapilha Durelli por ter me orientado durante o curso de mestrado, e principalmente, por ter tido paciência e dedicação para me ajudar durante toda a realização do meu trabalho. Não menos importante, agradeço aos amigos do Núcleo de Tecnologia da Informação (NTInf) da Universidade Federal de São João del-Rei pela compreensão e ajuda durante o período em que estava cursando o mestrado.

“The mass of men lead lives of quiet desperation” (Henry David Thoreau).

Resumo

Recentemente, a utilização de sistemas baseados em aprendizado de máquina cresceu consideravelmente. Com o aumento da utilização desse tipo de sistema de software em diferentes domínios, a necessidade de avaliar a confiabilidade desse tipo de sistema também tem se tornado cada vez mais importante. Todavia, sistemas baseados em aprendizado de máquina introduziram novos desafios que dificultam a realização de atividades de teste de software. Diferentemente dos sistemas tradicionais cujas regras de funcionamento são explicitamente programadas no código-fonte do sistema e com isso tendem a apresentar um comportamento determinístico, a natureza estatística associada à capacidade de realizar decisões autônomas faz com que os sistemas baseados em aprendizado sejam intrinsecamente difíceis de testar. Os principais problemas que dificultam a realização do teste desse tipo de sistema são: *(i)* o espaço de entrada abrangente; *(ii)* a inexistência de oráculos de teste; e *(iii)* a escassez de critérios de teste. A fim de abordar esses problemas, pesquisadores vêm adaptando as técnicas de teste consolidadas no teste de sistemas tradicionais para o contexto de aprendizado de máquina, propondo assim novas abordagens que possibilitam a identificação de defeitos em sistemas baseados em aprendizado de máquina. Nesse sentido, este trabalho apresenta uma abordagem de teste baseado em propriedades para modelos clássicos de aprendizado de máquina. Adicionalmente, apresenta-se uma ferramenta que foi desenvolvida como prova de conceito da abordagem proposta. Um experimento foi conduzido com o objetivo de avaliar a eficácia das amostras de dados de teste que são geradas com a execução das propriedades de teste. Os resultados sugerem que as amostras de dados de teste geradas por meio da abordagem proposta são mais eficazes do que as amostras de dados de teste que são normalmente selecionadas a partir de conjuntos de dados pré-existentes por meio da utilização de métodos de validação cruzada.

Palavras-chave: Teste de Software; Teste Baseado em Propriedades; Teste de Sistemas Baseados em Aprendizado de Máquina; Teste de Modelos de Aprendizado de Máquina.

Abstract

Recently, due to several technological advances, machine learning based software systems have gone mainstream. As a result of their widespread adoption, software testers are striving to come up with approaches to enhancing the quality and reliability of these solutions. Nonetheless, machine learning based systems pose unique testing challenges that differ from traditional software systems. Consequently, researchers have started exploring ways to adapt existing software testing techniques for machine learning based systems. To cope with the challenges of testing machine learning based systems, we propose an approach that builds on two test adequacy criteria based on decision tree models to generate property-based test cases for better evaluating machine learning models. Specifically, the proposed approach builds on two test adequacy criteria to leverage the internal structure of decision tree models. Thus, these decision tree based criteria are used to guide the selection of test inputs. This selection process is further refined by rendering the information about decisions into rules that dictate the behavior of the model. In order to explore the problem space more thoroughly, these rules are then turned into properties. To automate our approach, we developed a proof-of-concept tool that turns rules into executable code: properties. To evaluate our approach and the implementation thereof, we carried out an experiment using 21 datasets. We evaluated the effectiveness of test inputs in terms of the difference in model's behavior between the test input and the training data. The experiment results would seem to suggest that our property-based approach is suitable for guiding the generation of effective test data.

Key-words: Software Testing; Property-Based Testing; Machine Learning Based System Testing; Machine Learning Model Testing.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AM	Aprendizado de Máquina
AVLAD	Análise do Valor Limite de Cobertura de Árvore de Decisão
CAD	Cobertura de Árvore de Decisão
KNN	*K-Nearest Neighbor*
SVM	*Support Vector Machine*
TBP	Teste Baseado em Propriedades

Sumário

1 Introdução

O aprendizado de máquina fornece aos sistemas de software a capacidade de aprender por meio da utilização de dados sem a necessidade de serem explicitamente programados para isso (SHIVAHARE et al., 2022). Sarker (2021) afirma que nos últimos anos a utilização de técnicas de aprendizado de máquina no desenvolvimento de sistemas que funcionam de maneira inteligente cresceu consideravelmente. Riccio et al. (2020) até mesmo utilizaram o termo *sistema baseado em aprendizado de máquina* para distinguir esse tipo de sistema (que possui componentes que dependem da utilização de técnicas de aprendizado de máquina) dos conhecidos como *sistemas tradicionais* (que não utilizam aprendizado de máquina).

Dada a crescente utilização de sistemas baseados em aprendizado de máquina nos mais diversos domínios de aplicação (SARKER, 2021) aumentou-se a necessidade de avaliar a confiabilidade desse tipo de sistema (ZHANG et al., 2022). Porém, conforme ressaltado por Marijan et al. (2019), a realização do teste de sistemas baseados em aprendizado de máquina apresenta uma série de desafios adicionais como, por exemplo: *(i)* o espaço de entrada abrangente; *(ii)* a inexistência de oráculos de teste; e *(iii)* a escassez de critérios de teste. Para lidar com esses desafios, pesquisadores vêm realizando adaptações das técnicas consolidadas no teste de sistemas tradicionais a fim de propor novas abordagens de teste que são específicas para o teste de sistemas baseados em aprendizado de máquina (BRAIEK; KHOMH, 2020).

O teste baseado em propriedades é uma técnica de teste de sistemas tradicionais que foi popularizada por meio da ferramenta `QuickCheck` (CLAESSEN; HUGHES, 2011). A ideia empregada pela técnica de teste baseado em propriedades consiste em utilizar uma ferramenta que gera dados de teste de maneira aleatória para testar a validade de um conjunto de propriedades que descrevem o comportamento esperado do programa em teste (HUGHES, 2019). Por possibilitar a geração de uma considerável quantidade de dados de entrada que são submetidos ao programa em teste, a técnica de teste baseado em propriedades permite realizar o teste de sistemas tradicionais de maneira abrangente (LöSCHER; SAGONAS, 2018).

Zhang et al. (2022) afirmam que diferentemente de como é realizado o desenvolvimento de sistemas tradicionais, no qual as regras do sistema são implementadas diretamente no código-fonte do próprio sistema, os sistemas baseados em aprendizado de máquina são desenvolvidos utilizando um paradigma orientado a dados. Assim, as regras que definem o comportamento dos sistemas baseados em aprendizado de máquina são inferidas a partir dos dados de treinamento que são submetidos ao algoritmo de aprendi-

zado de máquina utilizado (BRAIEK; KHOMH, 2020). Como resultado deste processo é obtido um modelo de aprendizado de máquina que codifica as regras que foram inferidas desses dados (ZHANG et al., 2022). Essa mudança no paradigma de desenvolvimento torna difícil entender o comportamento de sistemas que possuem modelos de aprendizado de máquina. Resultando em sistemas que são intrinsecamente desafiadores para testar (RICCIO et al., 2020).

Diante dos problemas e desafios envolvidos no teste de sistemas baseados em aprendizado de máquina, este trabalho apresenta uma abordagem de teste baseado em propriedades para modelos clássicos de aprendizado de máquina. Os modelos clássicos de aprendizado de máquina são aqueles construídos com os algoritmos que pertencem ao aprendizado de máquina clássico, como por exemplo: os algoritmos de árvore de decisão, k-vizinhos mais próximos, regressão linear e naive bayes (ZHANG et al., 2022). Especificamente, este trabalho adapta a técnica de teste baseado em propriedades (CLAESSEN; HUGHES, 2011) utilizada no teste de sistemas tradicionais para o contexto de aprendizado de máquina. A técnica de teste baseado em propriedades é utilizada conjuntamente com os critérios de cobertura de árvore de decisão que foram propostos por Santos et al. (2021) para possibilitar o desenvolvimento de uma abordagem que possibilita a realização do teste baseado em propriedades de classificadores de aprendizado de máquina.

1.1 Justificativa

Kuwajima et al. (2020) afirmam que os modelos de aprendizado de máquina vêm se tornando componentes indispensáveis para o desenvolvimento de sistemas de diferentes domínios de aplicação. Tamanha é a relevância que um modelo de aprendizado de máquina possui em um sistema baseado em aprendizado de máquina que Riccio et al. (2020) introduziram o teste de nível de modelo: que considera o modelo de forma isolada sem levar em consideração qualquer outro componente do sistema. Fato que reforça a ideia de que o comportamento exibido por um modelo de aprendizado de máquina precisa ser verificado por meio da realização da atividade de teste (BRAIEK; KHOMH, 2020): de maneira semelhante como é abordado qualquer outro componente de software (RICCIO et al., 2020).

Devido a maneira como a propagação de erros afeta os sistemas baseados em aprendizado de máquina, Zhang et al. (2022) também alertam sobre a necessidade de testar individualmente cada componente de aprendizado de máquina existente nesse tipo de sistema. Amershi et al. (2019) afirmam que o alto nível de acoplamento existente entre os componentes de aprendizado de máquina faz com que a propagação de erros seja um problema ainda mais crítico no desenvolvimento de sistemas baseados em aprendizado de máquina do que no desenvolvimento de sistemas tradicionais (AMERSHI et al., 2019).

São encontrados na literatura trabalhos recentes que foram propostos pela comunidade científica de teste de software com o objetivo de possibilitar o teste de modelos de aprendizado profundo (modelos baseados em redes neurais profundas) (ZHANG et al., 2022). O trabalho de Pei et al. (2017) que introduziu o *framework* `DeepXplore` e o critério de cobertura de neurônios foi considerado pioneiro por possibilitar a realização de testes do tipo caixa branca em modelos baseados em redes neurais profundas. Consequentemente, o `DeepXplore` inspirou o desenvolvimento de diversas abordagens para o teste de modelos de aprendizado profundo, como por exemplo: `DeepTest` (TIAN et al., 2018), `DeepRoad` (ZHANG et al., 2018) e `DLFuzz` (GUO et al., 2018).

Em contrapartida, com exceção do trabalho de Santos et al. (2021) que introduziu dois critérios de cobertura de árvore de decisão para realização da amostragem de dados para o teste de modelos de aprendizado de máquina. Não foram encontrados trabalhos recentes envolvendo propostas que sejam específicas ou aplicáveis ao teste de modelos clássicos de aprendizado de máquina (modelos que não são baseados na utilização de redes neurais profundas). Diante dessa lacuna, foi identificada uma oportunidade para propor uma abordagem de teste para modelos clássicos de aprendizado de máquina. Com esse intuito, este trabalho apresenta uma abordagem de teste baseado em propriedades para modelos clássicos de aprendizado de máquina que adapta a utilização dos critérios de teste propostos por Santos et al. para possibilitar a construção de classes de teste (`arquivos.py`) executáveis que possibilitam realizar o teste automatizado de modelos clássicos de aprendizado de máquina.

1.2 Objetivos

O objetivo geral deste trabalho é realizar o desenvolvimento de uma abordagem de teste baseado em propriedades para modelos clássicos de aprendizado de máquina (especificamente, classificadores). Em tal contexto, espera-se que a abordagem proposta possibilite a geração automatizada de amostras de dados de teste que sejam diversas e eficazes. Com isso, espera-se ser possível realizar o teste automatizado de modelos clássicos de aprendizado de máquina.

Para ser possível alcançar o objetivo geral, foram estabelecidos os seguintes objetivos específicos:

- **Objetivo específico 1:** investigar os conceitos fundamentais relacionados ao teste de software com ênfase na técnica de teste baseado em propriedades.
- **Objetivo específico 2:** investigar os conceitos fundamentais relacionados ao aprendizado de máquina supervisionado com ênfase no processo de treinamento e avaliação de desempenho de classificadores.

- **Objetivo específico 3:** investigar os conceitos, processos, componentes, propriedades, problemas e desafios relacionados ao teste de sistemas baseados em aprendizado de máquina.
- **Objetivo específico 4:** revisar propostas relacionadas ao teste de sistemas baseados em aprendizado de máquina, e em específico, relacionadas ao teste de modelos de aprendizado de máquina que foram apresentadas na literatura.
- **Objetivo específico 5:** propor uma abordagem de teste baseado em propriedades para o teste de modelos clássicos de aprendizado de máquina, especificando e demonstrando o funcionamento do processo de geração de propriedades.
- **Objetivo específico 6:** selecionar como referência para implementação da abordagem proposta um conjunto de tecnologias existentes.
- **Objetivo específico 7:** desenvolver uma ferramenta que possibilita a automatização do processo de geração de propriedades da abordagem proposta por meio da construção de classes de teste executáveis que sejam compatíveis com as tecnologias selecionadas.
- **Objetivo específico 8:** conduzir um experimento com o objetivo de avaliar a eficácia das amostras de dados de teste que são geradas com a execução das propriedades de teste que são construídas com a abordagem proposta.

1.3 Organização do Texto

Este documento está organizado em oito capítulos. Além deste capítulo de introdução, o documento possui três capítulos de referencial teórico, um capítulo que aborda os trabalhos relacionados, um capítulo que descreve a abordagem proposta, um capítulo que descreve o experimento realizado e, por fim, um capítulo que apresenta as considerações finais. Uma descrição mais ampla sobre o conteúdo abordado nos próximos capítulos deste documento é apresentada a seguir:

- **Capítulo 2:** aborda teste de software. É realizada uma discussão sobre os conceitos básicos e as definições relacionadas ao teste de software. Tal capítulo apresenta uma visão geral sobre a automatização de testes com o *framework* de testes automatizados `pytest`. São apresentadas três técnicas de teste: *(i)* a técnica de teste funcional; *(ii)* a técnica de teste estrutural; e *(iii)* a técnica de teste baseado em propriedades. Por fim, é realizada uma discussão sobre a biblioteca `Hypothesis` de teste baseado em propriedades.

- **Capítulo 3:** aborda o tema de aprendizado de máquina. É realizada uma discussão sobre o aprendizado de máquina com ênfase em dois algoritmos específicos de aprendizado de máquina supervisionado: o algoritmo de árvore de decisão e o algoritmo k-vizinhos mais próximos. São apresentadas as etapas envolvidas em um processo de treinamento de um modelo de aprendizado de máquina. São discutidos sobre os métodos e métricas utilizadas para realização da avaliação de desempenho de modelos de aprendizado de máquina. Por fim, é apresentado o *framework* de aprendizado de máquina `scikit-learn`.

- **Capítulo 4:** aborda o tema de testes de sistemas baseados em aprendizado de máquina. É realizada uma discussão centrada na interpretação de teste de sistemas baseados em aprendizado de máquina que é dada pela comunidade científica de teste de software. São apresentados os processos de teste online e offline de modelos de aprendizado de máquina. É realizada uma discussão sobre os diferentes componentes e níveis de teste que são específicos do teste de sistemas baseados em aprendizado de máquina. São apresentadas algumas propriedades de teste, como por exemplo, a corretude. São discutidos os principais problemas que envolvem o teste de sistemas baseados em aprendizado de máquina e as principais técnicas de teste que estão sendo adaptadas para tentar solucionar esses problemas. Por fim, são apresentados os critérios de teste que foram propostos por Santos et al. (2021).

- **Capítulo 5:** apresenta os trabalhos relacionados à proposta deste trabalho. São apresentados alguns trabalhos relacionados ao teste de conjunto de dados, ao teste de implementação de algoritmos de aprendizado de máquina e ao teste de modelos de aprendizado de máquina.

- **Capítulo 6:** apresenta a abordagem de teste baseado em propriedades para modelos clássicos de aprendizado de máquina que é proposta neste trabalho. É realizada uma discussão inicial sobre o escopo de aplicação e a motivação da abordagem proposta. É fornecida uma visão geral da abordagem proposta. É especificado e demonstrado o funcionamento do processo de geração de propriedades. Por fim, é realizada a apresentação da ferramenta proposta neste trabalho que foi desenvolvida para automatizar o processo de geração de propriedades.

- **Capítulo 7:** descreve o experimento que foi conduzido neste trabalho com a finalidade de avaliar a eficácia da abordagem proposta. São descritas todas as etapas de execução do experimento. São apresentados e discutidos os resultados. Por fim, são destacadas algumas ameaças à validade do experimento.

- **Capítulo 8:** apresenta as considerações finais deste trabalho; também são discutidas algumas limitações existentes na abordagem proposta. Por fim, são descritos alguns trabalhos futuros que podem ser realizados como uma extensão deste trabalho.

2 Teste de Software

Myers et al. (2011) afirmam que o teste de software é um processo, ou uma série de processos, que são utilizados para garantir que sistemas de software façam exatamente o que foram projetados para fazer. Os autores complementam, afirmando que o software deve apresentar um comportamento previsível e consistente, que comportamentos inesperados não devem ser realizados. O processo de teste de software desempenha um papel fundamental para a melhoria da qualidade de um sistema de software (TRIPATHY; NAIK, 2011). De acordo com Tripathy e Naik (2011), as atividades de avaliação da qualidade de software podem ser divididas em duas grandes categorias, a saber, a análise estática e a análise dinâmica. Delamaro et al. (2016) destacam que a análise estática refere-se às atividades de verificação e validação que não requerem a execução ou a existência de um sistema de software para serem conduzidas, enquanto, as atividades dinâmicas, necessitam da execução do sistema de software que será testado.

De acordo com Tripathy e Naik (2011), verificação e validação são dois conceitos relacionados ao teste de software que frequentemente são utilizados por testadores e desenvolvedores. Ammann e Offutt (2016) definem a verificação como o processo que permite determinar se o produto de uma fase do processo de desenvolvimento do software atende aos requisitos estabelecidos antes do início de tal fase. De forma semelhante, os autores definem a validação como o processo que permite avaliar o software no final do seu desenvolvimento, para garantir que as expectativas e as necessidades dos usuários foram atendidas. A discussão realizada neste capítulo é focada principalmente em técnicas de verificação.

Este capítulo está organizado conforme a seguir. Na Seção 2.1 são apresentados alguns conceitos e definições relacionadas ao teste de software. Na Seção 2.2 é discutido sobre os níveis de teste de software. A Seção 2.3 aborda o assunto de automatização de testes e apresenta um exemplo simples de como o *framework* `pytest` pode ser utilizado para a realização da automatização de casos de teste. Na Seção 2.4 é discutido sobre a técnica de teste funcional e são apresentados dois critérios de teste funcional: o critério particionamento em classes de equivalência e o critério análise do valor limite. A Seção 2.5 apresenta a técnica de teste estrutural e aborda alguns critérios de teste estrutural baseados em fluxo de controle. Na Seção 2.6 é discutido sobre uma técnica de teste aleatório conhecida como teste baseado em propriedades e são apresentados exemplos de como construir propriedades com a biblioteca `Hypothesis`. Por fim, na Seção 2.7 são apresentadas as considerações finais do capítulo.

2.1 Introdução ao Teste de Software

Para iniciar a discussão sobre teste de software, primeiramente, são apresentadas as definições de alguns termos relacionados ao teste de software que frequentemente são utilizados como se fossem sinônimos e de forma errônea. Por exemplo, os termos, engano, defeito, erro, falha e *bug* (AMMANN; OFFUTT, 2016). Porém, na literatura de teste de software tradicional, cada um desses termos possui significado distinto (DELAMARO et al., 2016). Portanto, a seguir são apresentadas as definições específicas que são encontradas na literatura e utilizadas no decorrer deste documento para cada dos termos mencionados anteriormente:

- **Defeito (*Fault*):** um defeito estático no software (AMMANN; OFFUTT, 2016). Por exemplo, uma instrução, processo ou definição de dados incorreta (DELAMARO et al., 2016).
- **Engano (*Mistake*):** a ação humana que produz um defeito (DELAMARO et al., 2016).
- **Erro (*Error*):** um estado incorreto do sistema que é causado devido a manifestação de algum defeito (AMMANN; OFFUTT, 2016). O estado de execução de um programa é determinado pelo valor das variáveis do programa em memória e o pelo valor do apontador de instruções. A existência de um defeito pode ocasionar um erro durante a execução do programa (DELAMARO et al., 2016).
- **Falha (*Failure*):** comportamento incorreto do sistema em relação ao que foi especificado (AMMANN; OFFUTT, 2016). Um estado incorreto pode resultar em uma falha, ou seja, um erro pode levar a uma falha (DELAMARO et al., 2016).
- ***Bug*:** termo informal que, normalmente, é utilizado para fazer referência aos quatro termos definidos acima (AMMANN; OFFUTT, 2016).

Outras definições que também necessitam ser apresentadas em uma discussão sobre teste de software são as definições de domínios de entrada e saída de um programa, de dado de teste, de caso de teste, e de um conjunto de casos de teste. Delamaro et al. (2016) definem o domínio de entrada de um programa $\mathcal{P}$, denotado por $\mathcal{D}(\mathcal{P})$, como o conjunto de todos os possíveis valores que podem ser utilizados para executar $\mathcal{P}$. O conjunto de todos os possíveis resultados que podem ser produzidos pelo programa $\mathcal{P}$ é definido como o domínio de saída. Os autores ainda definem um dado de teste para um programa $\mathcal{P}$, como um elemento qualquer do domínio de entrada de $\mathcal{P}$. Um caso de teste é definido como um par formado por um dado de teste mais o resultado esperado para a execução do programa com aquele dado de teste. Por fim, o conjunto de todos os casos de teste

utilizados durante a realização de uma atividade de teste é definido como um conjunto de teste, conjunto de casos de teste ou suíte de testes.

Diferentes definições de casos de teste são encontradas na literatura de teste de software. Ammann e Offutt (2016) apresentaram uma definição um pouco mais abrangente. Na visão dos autores, um caso de teste é composto por, valores de caso de teste, valores prefixados, valores pós-fixados e resultados esperados, que são necessários para realizar uma execução e avaliação completa do programa em teste. Os autores definiram cada um dos diferentes tipos de valores, conforme a listagem a seguir.

- **Valores de caso de teste (*test case values*):** os valores de entrada necessários para realizar a execução do programa em teste.
- **Valores prefixados (*prefix values*):** os valores necessários para colocar o programa sendo testado no estado apropriado para receber os valores de teste.
- **Valores pós-fixados (*postfix values*):** os valores que precisam ser enviados ao programa sendo testado após o envio dos valores de teste.
- **Resultados esperados (*expected results*):** os resultados esperados que devem ser produzidos pelo programa em questão.

Perceba que a definição de caso de teste apresentada por Delamaro et al. (2016) não é divergente da definição fornecida por Ammann e Offutt (2016). De certo modo, a definição de Ammann e Offutt (2016) complementa a anterior. O que os autores definiram como valores de caso de teste, em termos práticos, é exatamente o que Delamaro et al. (2016) chamaram de dados de teste: são elementos do domínio de entrada do programa em teste. As definições de resultados esperados são equivalentes. De fato, as únicas diferenças existentes nas definições apresentadas se resumem aos valores prefixados e pós-fixados que aparecem na definição de Ammann e Offutt (2016). Segundo os autores, apenas os valores de teste obtidos do domínio do programa em teste podem não ser suficientes para realização de uma atividade de teste. Outros dados também podem ser necessários, os valores prefixados e pós-fixados, respectivamente, são utilizados para fornecer ao programa em teste os dados que irão satisfazer as pré-condições e pós-condições do programa. Em alguns casos, tais valores podem ser necessários para realização de uma atividade de teste completa.

A Figura 1 mostra um cenário típico da atividade de teste. Depois de definir um conjunto de casos de teste $\mathcal{T}$, deve ser executado o programa em teste $\mathcal{P}$, utilizando $\mathcal{T}$. Se os resultados produzidos pela execução de $\mathcal{P}$ são equivalentes aos resultados esperados, então é possível afirmar que nenhum erro foi identificado. Porém, caso algum caso de teste seja capaz de produzir um resultado diferente do esperado, então afirma-se que um

defeito foi revelado. Como sugere a Figura 1, em geral, é responsabilidade do testador, baseado na especificação do programa, $\mathcal{S}(\mathcal{P})$, ou em qualquer outro documento que define o comportamento de $\mathcal{P}$, a realização da análise da corretude de uma execução. Consequentemente, é dito que em tal contexto o testador desempenha o papel de oráculo de teste (DELAMARO et al., 2016).

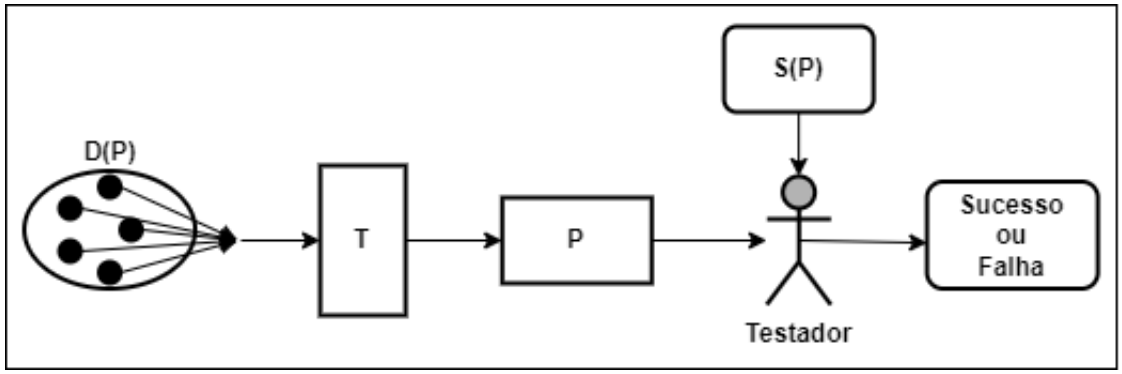

Figura 1 – Cenário típico da atividade de teste. Retirado de (DELAMARO et al., 2016).

Um oráculo de teste é um instrumento que decide se o resultado obtido com determinada execução do programa em teste coincide com o resultado esperado para a execução (DELAMARO et al., 2016). É o oráculo de teste que determina se a realização de um teste resultou em sucesso ou falha (AMMANN; OFFUTT, 2016). Em termos práticos, isso significa que o resultado esperado deve ser incluído no caso de teste para ser possível observar as falhas existentes no programa durante a realização da atividade de teste.

Um dos maiores desafios presentes no teste de software está relacionado ao tamanho do domínio de entrada do programa em teste. Mesmo para programas simples, o tamanho do domínio, $\mathcal{D}(\mathcal{P})$, pode ser demasiadamente grande. Por exemplo, um programa que recebe três números inteiros e realiza o cálculo da média dos valores, em uma máquina de 32 bits, cada parâmetro de entrada do programa terá 4 bilhões de valores possíveis. Ao considerar os três parâmetros, o programa terá um domínio de entrada com 80 octilhões de possibilidades (AMMANN; OFFUTT, 2016). Utilizar todas as possibilidades do domínio para realizar a atividade de teste é praticamente impossível. De acordo com Delamaro et al. (2016), a realização do teste exaustivo é impraticável para a maioria dos programas pois o tempo necessário para realizar a atividade de teste torna-se inviável. É importante ressaltar que a atividade de teste é capaz de indicar a presença de defeitos, porém, não é possível atestar a ausência de defeitos. Visto que não é possível testar um programa com todos os dados de entrada do domínio, uma questão fundamental é a seleção de dados de teste.

Diante da impossibilidade prática de utilizar todos os dados de entrada do domínio do programa. A alternativa viável adotada por testadores consiste em utilizar apenas alguns dados de entrada específicos para conduzir a atividade de teste. Esse cenário é exemplificado na Figura 1 que mostra a realização do teste do programa $\mathcal{P}$ por meio

da seleção de alguns pontos específicos do domínio de $\mathcal{P}$: $\mathcal{D}(\mathcal{P})$. Apenas esses dados de entrada específicos são utilizados para criar o conjunto de casos de teste $\mathcal{T}$ que é executado contra o programa $\mathcal{P}$ (DELAMARO et al., 2016).

Visto que a detecção de falhas é um aspecto fundamental do teste de software, vários pesquisadores têm enfatizado a formalização dos conceitos relacionados à identificação de defeitos e falhas em sistemas de software. Ammann e Offutt (2016), por exemplo, apresentam um modelo de defeito/falha de sofware, conhecido em inglês como, *RIP model*. A Figura 2 apresenta tal modelo. Para revelar uma falha, primeiramente, o caso de teste deve alcançar a localização do programa que contém o defeito (*Reachability*). Ao executar uma instrução com defeito o estado da execução do programa torna-se incorreto (*Infection*). Então, o estado incorreto deve se propagar para o restante da execução do programa fazendo com que o programa retorne uma saída incorreta ou finalize a execução em um estado incorreto (*Propagation*). Por fim, o testador deve ser capaz de observar a parte incorreta do estado final do programa (*Revealability*). Especificamente, o testador deve ser capaz de identificar a diferença entre o comportamento exibido e o comportamento esperado do programa. Essa identificação é realizada utilizando oráculos de teste: tal função é, normalmente, exercida pelo próprio testador. Caso o testador não seja capaz de observar a parte incorreta do estado final do programa, a falha não é identificada e o defeito não é revelado (AMMANN; OFFUTT, 2016).

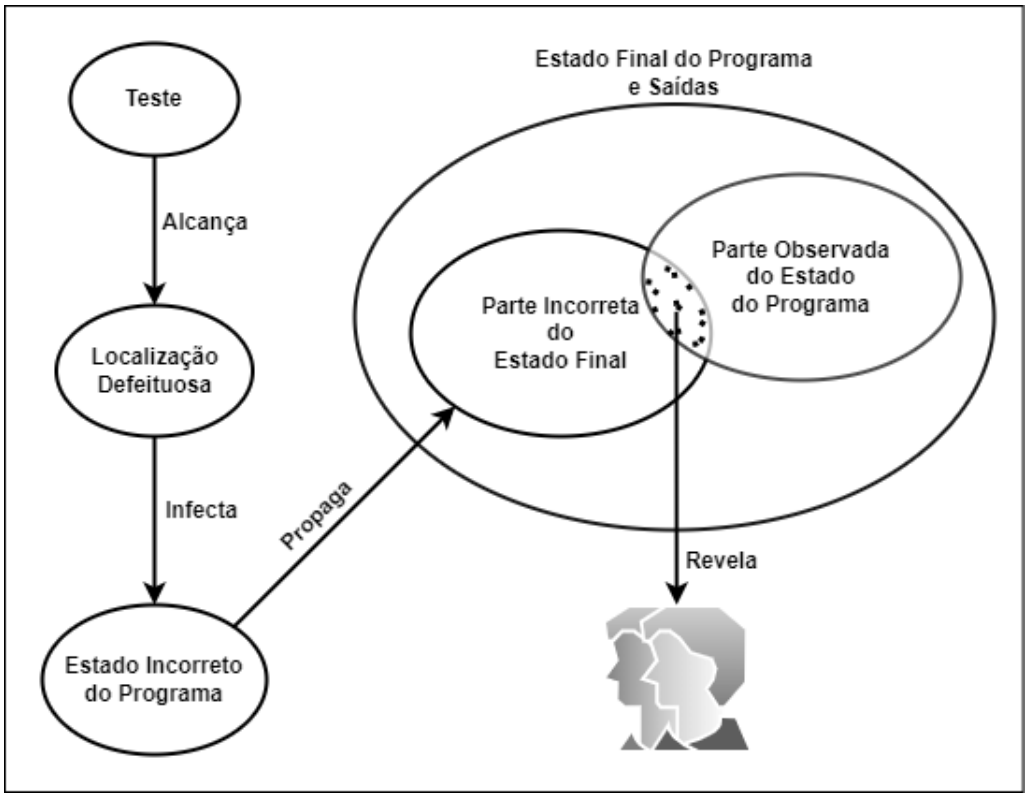

Figura 2 – Modelo de defeito/falha de software. Denominado em inglês como *Reachability, Infection, Propagation, Revealability model*. Adaptado de (AMMANN; OFFUTT, 2016).

O modelo de defeito/falha apresentado na Figura 2, torna evidente a importância da utilização de casos de teste eficazes para a realização do teste de software. A construção de um conjunto de casos de teste eficaz normalmente depende de dois fatores: da seleção de um conjunto de dados de teste que possua alta probabilidade de revelar defeitos; e da utilização de oráculos de teste capazes de determinar corretamente o comportamento esperado do programa em teste para cada dado de teste selecionado (MYERS et al., 2011).

Ammann e Offutt (2016) afirmam que a atividade de teste de software pode ser dividida em quatro tarefas: o projeto de testes; a automatização dos testes; a execução dos testes; e a avaliação dos testes. O projeto de casos de teste refere-se à construção do conjunto de casos de teste que será utilizado para testar o software. A automatização dos testes consiste na incorporação dos casos de teste em *scripts* executáveis. Na tarefa de execução dos testes é realizada a execução dos casos de teste contra o programa em teste. Por fim, a avaliação dos testes consiste na análise dos resultados que foram obtidos com os testes realizados.

O projeto de casos de testes é tecnicamente e matematicamente a tarefa mais desafiadora presente no teste de software (AMMANN; OFFUTT, 2016). Para lidar com esse desafio, pesquisadores em engenharia de software vêm conduzindo diversos esforços para propor técnicas que possibilitem a realização da atividade de teste de software de maneira sistemática. Uma técnica de teste (TRIPATHY; NAIK, 2011), que também pode ser denominada de estratégia ou metodologia de teste (MYERS et al., 2011), ou até mesmo, método de teste (MILI; TCHIER, 2015), consiste na utilização sistemática das informações de diferentes artefatos do software em desenvolvimento, para possibilitar a seleção de um subconjunto reduzido do domínio do programa em teste. Espera-se que o subconjunto reduzido seja composto por dados de teste que possuam alta probabilidade de revelar possíveis defeitos existentes no programa.

Três técnicas de teste de software amplamente abordadas na literatura de teste de softwares são: a técnica de teste aleatório; a técnica de teste funcional; e a técnica de teste estrutural (MILI; TCHIER, 2015). A técnica de teste aleatório, conforme o nome sugere, consiste em selecionar aleatoriamente um grande número de dados de teste do domínio do programa em teste. Os dados de teste selecionados são utilizados para construir um conjunto reduzido de casos de teste. Ao executar o conjunto reduzido de casos de teste contra o programa em teste, espera-se que, probabilisticamente, exista uma boa chance de que o domínio original do programa esteja razoavelmente representado com os dados de teste selecionados aleatoriamente (DELAMARO et al., 2016). Mili e Tchier (2015) afirmam que a técnica de teste aleatório possibilita construir casos de teste que simulam as condições normais de utilização do programa em teste.

A técnica de teste funcional e a técnica de teste estrutural realizam a seleção do subconjunto reduzido de dados de teste utilizando critérios de teste. Um critério de teste é

uma regra ou um conjunto de regras que determina requisitos de teste para a construção de um conjunto de casos de testes. Um requisito de teste é um elemento específico de um artefato de software que um caso de teste deve satisfazer (AMMANN; OFFUTT, 2016). Por meio da utilização de determinado critério de teste é obtido um conjunto específico de requisitos de teste. O conjunto de requisitos de teste resultante pode ser utilizado para guiar a seleção de dados de teste do domínio do programa em teste. Ao escolher pelo menos um dado de teste, que satisfaz cada requisito de teste, torna-se possível construir um conjunto de casos de teste adequado em relação ao critério de teste sendo utilizado (DELAMARO et al., 2016).

Vários critérios de teste funcional e estrutural foram propostos na literatura de teste de software (AMMANN; OFFUTT, 2016). Dependendo do critério utilizado, são obtidos conjuntos de requisitos de teste diferentes. Consequentemente, são construídos conjuntos de casos de teste distintos (DELAMARO et al., 2016). Critérios de técnicas diferentes utilizam fontes de informações distintas para produzir os casos de teste, resultando em conjuntos de casos de teste que exploram diferentes aspectos do programa sendo testado (ANICHE, 2022).

Mesmo entre os critérios provenientes de uma mesma técnica de teste, existem alguns que são mais eficazes do que outros (AMMANN; OFFUTT, 2016). Dessa forma, de acordo com Myers et al. (2011), quando possível, mais de um critério de teste deve ser combinado para construção de um conjunto de casos de teste eficaz. Adicionalmente, quando possível, critérios provenientes de diferentes técnicas de teste devem ser utilizados. Parece haver um consenso na literatura de teste de software em relação à utilização de diferentes técnicas e critérios de teste de forma complementar.

O teste funcional e o teste estrutural serão discutidos com mais detalhes nas Seções 2.4 e 2.5, respectivamente. Na Seção 2.6 será discutido uma técnica de teste aleatório conhecida como teste baseado em propriedades. A tarefa de criar casos de teste é importante, entretanto, ela não é a única tarefa presente no teste de software (AMMANN; OFFUTT, 2016). A fim de oferecer uma explicação abrangente sobre o tema a seção seguinte introduz o conceito de níveis de teste e em seguida aborda-se a tarefa de automatização de testes. As especificidades de cada uma das técnicas é também descrita no decorrer deste capítulo.

2.2 Níveis de Teste

Existe um modelo de teste de software, conhecido como Modelo V, que atribui um nível de teste distinto para cada atividade existente no processo de desenvolvimento do sistema (AMMANN; OFFUTT, 2016). Conforme pode ser observado na Figura 3, a informação utilizada em cada nível de teste é obtida da atividade de desenvolvimento

associada.

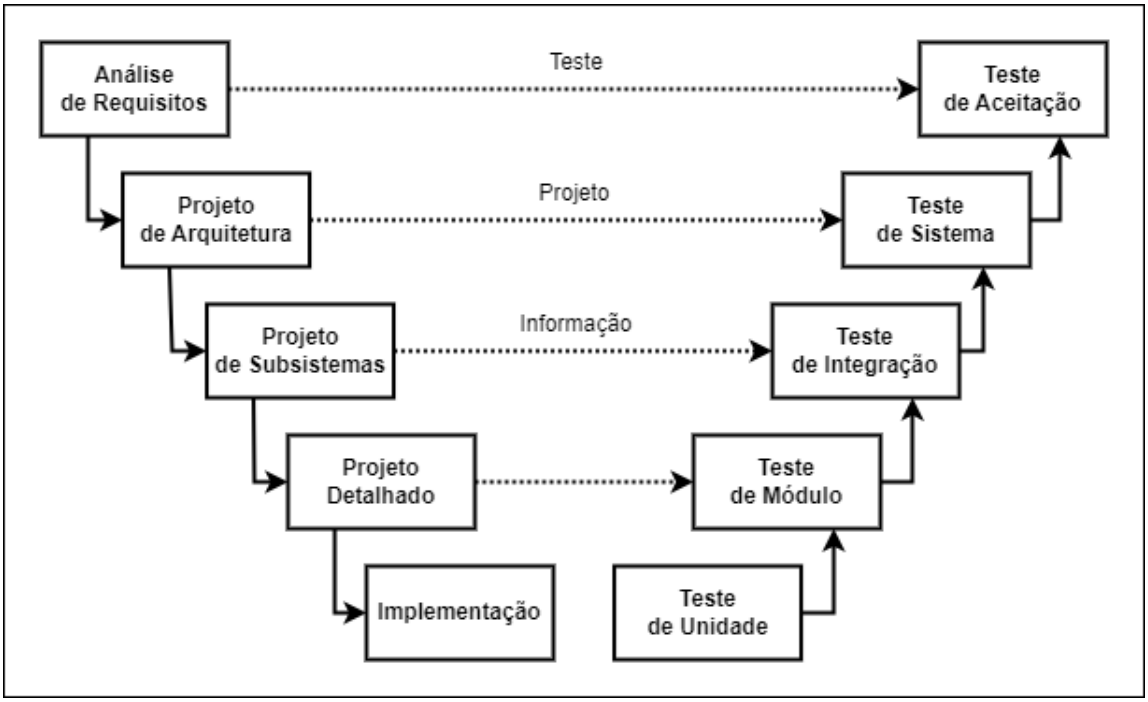

Figura 3 – Modelo V: Relaciona as atividades de desenvolvimento de sistema com os níveis de teste. Adaptado de (AMMANN; OFFUTT, 2016).

Cada nível de teste possui um objetivo específico e possibilita encontrar diferentes tipos de defeitos existentes no sistema em teste. Os níveis de teste apresentados na Figura 3, são definidos conforme a listagem a seguir.

- **Teste de unidade:** os testes de unidades são projetados para avaliar as unidades produzidas na fase de implementação do sistema (AMMANN; OFFUTT, 2016). Geralmente, são considerados como unidades os procedimentos, as funções e os métodos implementados pelos desenvolvedores (TRIPATHY; NAIK, 2011).

- **Teste de módulo:** os testes de módulos são projetados para avaliar os módulos do sistema isoladamente. Os módulos são coleções de unidades que são especificadas na fase de projeto detalhado do sistema, e podem ser arquivos, pacotes ou uma classe (AMMANN; OFFUTT, 2016). Neste nível de teste, busca-se identificar erros decorrentes da forma como as unidades de cada módulo interagem umas com as outras, e com as estruturas de dados presentes nos módulos.

- **Teste de integração:** os testes de integração são projetados para avaliar se as interfaces existentes nos módulos dos subsistemas, especificados na fase de projeto de subsistema, foram bem definidas e se comunicam corretamente (AMMANN; OFFUTT, 2016). O teste de integração assume que os módulos dos subsistemas funcionam corretamente, e com isso, busca verificar se a integração dos subsiste-

mas funcionam de forma razoavelmente estável para suportar o rigor dos testes de sistema (TRIPATHY; NAIK, 2011).

- **Teste de sistema:** os testes de sistemas utilizam as informações do projeto de arquitetura para projetar casos de teste capazes de verificar se o sistema atende a especificação (AMMANN; OFFUTT, 2016). Neste nível de teste, é considerado que todas as partes do sistema estão corretamente integradas, então, o sistema é testado como um todo (DELAMARO et al., 2016). O teste de sistema inclui uma grande variedade de tipos de teste, como o teste de funcionalidades, teste de segurança, teste de robustez, teste de desempenho e teste de confiabilidade (TRIPATHY; NAIK, 2011).

- **Teste de aceitação:** os testes de aceitação utilizam as informações da análise de requisitos para projetar casos de teste capazes de determinar se o sistema atende as necessidades do usuário (TRIPATHY; NAIK, 2011). Prioritariamente, neste nível de teste devem ser envolvidos os usuários ou especialistas do domínio do sistema (AMMANN; OFFUTT, 2016). O principal objetivo é medir a qualidade do sistema em relação às expectativas do usuário.

A construção dos casos de teste e a condução das atividades em cada nível de teste são realizadas por profissionais com perfis técnicos distintos. O teste de unidade e o teste de módulo são realizados pelos próprios desenvolvedores responsáveis pela implementação. O teste de integração pode ser conduzido pelos desenvolvedores ou por especialistas em teste de integração. A realização do teste de sistema compreende um conjunto de atividades específicas, como, a elaboração do plano de testes, a construção da suíte de testes, a preparação do ambiente de testes, a execução dos testes por meio de estratégias específicas e a monitoração do processo de execução dos testes (TRIPATHY; NAIK, 2011). Por isso, geralmente é recomendado que o teste de sistema seja realizado por uma equipe especialista em teste de software que seja diferente da equipe de desenvolvedores (AMMANN; OFFUTT, 2016).

Além dos níveis de teste apresentados, existe um outro tipo de teste, chamado de teste de regressão. O teste de regressão é realizado sempre que um componente do software é modificado. Por isso, é dito que o teste de regressão faz parte da fase de manutenção do software (DELAMARO et al., 2016). A ideia central do teste de regressão é garantir que uma modificação, recém realizada, não introduza novos defeitos em qualquer outra parte do sistema (AMMANN; OFFUTT, 2016). O teste de regressão não é considerado um novo nível de teste, esse tipo de teste é considerado uma subfase do teste de unidade, teste de módulo, teste de integração e teste de sistema (BERTOLINO; MARCHETTI, 2005). O objetivo do teste de regressão não é construir novos casos de testes. Em vez disso, os casos de teste são selecionados, priorizados e executados da suíte de testes existente para

garantir que nenhum defeito foi introduzido na nova versão do software (TRIPATHY; NAIK, 2011).

2.3 Automatização de Testes

O teste de software é trabalhoso e pode ter um alto custo dependendo da complexidade do sistema. Para lidar com este ponto negativo, um objetivo importante do teste de software é automatizar o máximo possível o processo de teste. Ammann e Offutt (2016) definem a tarefa de automatização dos testes como a utilização de software para controlar:

- a definição das pré-condições;
- a execução dos testes;
- a comparação dos resultados obtidos com os resultados esperados;
- a definição das pós-condições;
- e a disponibilização de funções para construção de relatórios de teste.

A automatização dos testes oferece uma série de vantagens para os desenvolvedores e testadores de software. De acordo com Ammann e Offutt (2016), a automatização de procedimentos repetitivos, como a manutenção de *scripts* de teste, a reexecução de testes e a comparação dos resultados obtidos com os resultados esperados, não apenas ajuda a reduzir o custo envolvido no teste de software, como ajuda a reduzir o erro humano. Além disso, eliminar este tipo de procedimento repetitivo libera o tempo dos desenvolvedores e testadores para focar em atividades mais desafiadoras como o projeto de casos de teste. Os autores ainda consideram que outra vantagem imediata da automatização é a possibilidade de executar os mesmos testes diversas vezes sem esforço humano adicional. O que é essencial para facilitar a condução dos testes de regressão.

A principal maneira utilizada para realizar a automatização do teste de software é transformando os casos de teste em testes executáveis (AMMANN; OFFUTT, 2016). Isso pode ser feito utilizando *scripts* de shell, arquivos de entrada, ou utilizando uma ferramenta capaz de controlar o sistema ou componentes do sistema em teste. Ao optar pela utilização de uma ferramenta, o ideal é que ela ofereça suporte a execução completa do teste que se deseja realizar. Por exemplo, ao utilizar uma ferramenta que oferece suporte ao teste de unidade, é interessante que essa ferramenta possibilite executar a unidade em teste com os dados de teste, que ela possibilite obter o resultado da execução, que compare o resultado da execução com o resultado esperado, e que por fim, forneça um relatório para o testador.

Ammann e Offutt (2016) definem um *framework* de teste de software como um conjunto de suposições, conceitos e ferramentas que oferecem suporte à automatização dos testes. Opções diversas de *frameworks* de teste são encontradas em diferentes linguagens de programação. Na linguagem Python, um dos *frameworks* mais conhecidos e utilizados é o `pytest`.[1] Por ser o *framework* de teste automatizado adotado no contexto deste trabalho, na próxima subseção será realizada uma breve discussão sobre o `pytest`.

2.3.1 Automatização de Testes com o *Framework* `pytest`

O `pytest` é um *framework* de testes automatizado executável por meio de linha de comando que automaticamente identifica os testes escritos pelo testador, realiza a execução dos testes, e reporta os resultados obtidos. Okken (2022) considera o `pytest` um dos melhores *frameworks* de testes disponíveis para a linguagem Python devido ao seguinte conjunto de características:

- facilidade oferecida para implementação de testes de diferentes níveis de complexidade;
- facilidade oferecida para leitura e entendimento de testes escritos por outros testadores;
- baixa curva de aprendizagem oferecida;
- a possibilidade de utilizar a palavra reservada *assert*, da linguagem Python para implementar as verificações de valores esperados dos resultados de teste;
- e a possibilidade de utilizar o `pytest` para executar testes escritos com outros *frameworks* de teste disponíveis para a linguagem Python, como por exemplo, executar testes escritos com o `unittest`[2] ou o nose[3].

A Listagem 2.1 apresenta um código de teste muito simples que foi retirado de (OKKEN, 2022), e que será utilizado como exemplo para ilustrar funcionamento básico do `pytest`. A função `test_passing()` será identificada pelo Pytest como uma função de teste pois o seu nome começa com o prefixo `test_`, e além disso, essa função está implementada em um arquivo Python, que também possui um nome que começa com o prefixo `test_`. Quando a função de teste for executada com o `pytest`, a instrução `assert` vai determinar se o teste passou ou falhou. A instrução `assert` é uma palavra reservada da linguagem Python que possui o comportamento padrão de levantar uma exceção do tipo `AssertionError` quando a expressão lógica que vem imediatamente depois de `assert`

[1] https://docs.pytest.org/en/latest/
[2] https://docs.python.org/3/library/unittest.html
[3] https://nose.readthedocs.io/en/latest/testing.html

resultar no valor falso. Por sua vez, o `pytest` determina que o teste falhou sempre que uma exceção não capturada é levantada durante a execução de um teste. Diante disso, quando a função de teste mostrada na Listagem 2.1 for executada com o `pytest`, será realizado um teste bem sucedido, devido a expressão *(1, 2, 3) == (1, 2, 3)* resultar no valor verdadeiro.

```
def test_passing():
    assert (1, 2, 3) == (1, 2, 3)
```

Listagem 2.1 – Exemplo de uma função de teste compatível com o *framework* Pytest.

O procedimento que o `pytest` executa em segundo plano para identificar quais testes devem ser executados é chamado de descoberta de teste, do termo em inglês, *test discovery* (OKKEN, 2022). Para que o *framework* consiga identificar automaticamente os testes para execução, é necessário que os testes sejam implementados seguindo convenções de nomes específicas. As principais convenções de nomes adotadas pelo `pytest` são as seguintes:

- os arquivos Python contendo casos de teste devem ser nomeados com o prefixo `test_` ou com o sufixo `_test.py`, por exemplo, `test_<nome>.py` ou `<nome>_test.py`;
- os métodos ou funções de teste devem ser nomeados com o prefixo `test_`, por exemplo, `def test_<nome>():`;
- e as classes de teste devem ser nomeadas com o prefixo `Test`, por exemplo, `class Test<nome>`.

2.4 Técnica de Teste Funcional

Teste funcional é uma técnica de teste que considera o sistema em teste como uma caixa preta (DELAMARO et al., 2016). Por isso, essa técnica também costuma ser denominada de teste caixa-preta. O testador não possui acesso ou não considera os detalhes internos de implementação do sistema para projetar os casos de teste. Os casos de teste são projetados utilizando como fonte de informação as especificações do sistema ou propriedades identificadas no próprio domínio de entrada do sistema em teste (TRIPATHY; NAIK, 2011). Por esse motivo, alguns autores como Jorgensen e DeVries (2021) e Aniche (2022) preferem utilizar o termo "teste baseado em especificação" para nomear essa técnica de teste.

O nome "teste funcional" foi adotado para fazer referência ao fato de que qualquer programa pode ser considerado uma função que realiza o mapeamento de dados do domínio de entrada para dados do domínio de saída do programa em teste (JORGENSEN; DEVRIES, 2021). A noção de um programa como uma função de mapeamento, associada

à consideração do programa em teste como uma caixa preta, possibilita realizar a avaliação do sistema em teste por meio do ponto de vista do usuário (DELAMARO et al., 2016). No contexto do teste funcional, o testador concentra-se apenas nas funcionalidades do sistema e nos resultados observáveis que são produzidos como respostas as entradas fornecidas (TRIPATHY; NAIK, 2011). A próxima subseção descreve um exemplo de critério de teste funcional que é relevante no contexto do trabalho proposto.

2.4.1 Critério Particionamento em Classes de Equivalência

O projeto de casos de teste utilizando o critério particionamento em classes de equivalência é realizado por meio de dois processos distintos: *(i)* a identificação das classes de equivalência; e *(ii)* a seleção dos dados de teste. A identificação das classes de equivalência inicia com uma análise inicial da especificação do programa em teste com o objetivo de identificar as condições de entrada que são determinadas na especificação. Para tal, o testador busca por frases ou sentenças que possam fazer referência a essas condições. Uma vez identificada, uma condição deve ser particionada em dois ou mais conjuntos disjuntos. É importante ressaltar que neste processo, dois tipos de classes de equivalência devem ser identificados: as classes válidas que representam os dados de teste que são válidos e as classes inválidas que representam dados de teste inválidos (MYERS et al., 2011).

Delamaro et al. (2016) destacam que o ponto forte do critério particionamento em classes de equivalência reside na capacidade que o critério possui de diminuir o tamanho do domínio de entrada do programa em teste e no fato da seleção dos dados de teste ser baseada apenas na especificação do programa. Além disso, os autores acrescentam que esse critério é mais adequado para aplicações em que as variáveis de entrada podem ser identificadas com facilidade e assumem valores específicos, visto que isso possibilita que as classes de equivalência sejam identificadas com mais facilidade.

2.4.2 Critério Análise do Valor Limite

De acordo com Myers et al. (2011), os dados de teste que exploram condições limites possuem maior probabilidade de revelar defeitos. As condições limites correspondem aos valores que estão imediatamente abaixo, exatamente sobre e imediatamente acima dos valores limites das classes de equivalência do programa em teste (DELAMARO et al., 2016).

O critério análise do valor limite difere do critério particionamento em classes de equivalência devido a dois fatores (MYERS et al., 2011). Primeiro, ao contrário de selecionar qualquer dado de teste pertencente a uma determinada classe de equivalência, o critério análise do valor limite requer que um ou mais dados de teste sejam selecionados

de modo que cada limite da classe de equivalência seja explorado com a realização da atividade de teste. Além disso, ao invés de apenas considerar o domínio de entrada do programa em teste para selecionar os dados de teste, o domínio de saída do programa também deve ser levado em consideração.

2.5 Técnica de Teste Estrutural

A técnica de teste estrutural utiliza o código-fonte do sistema em teste como a principal fonte de informação (AMMANN; OFFUTT, 2016). As informações da estrutura interna do código-fonte do sistema são utilizadas para projetar casos de teste com base em detalhes específicos de implementação (JORGENSEN; DEVRIES, 2021). Por esse motivo, essa técnica de teste também é chamada de teste caixa-branca (DELAMARO et al., 2016). Ao utilizar um critério de teste estrutural, o testador examina o código-fonte do sistema em teste com foco no fluxo de controle ou no fluxo de dados do programa (TRIPATHY; NAIK, 2011).

De acordo com Delamaro et al. (2016), uma representação do programa em teste, denominada de grafo de fluxo de controle ou de grafo de programa, é utilizada para definir a maioria dos critérios de teste da técnica estrutural. Dado um programa escrito em uma linguagem de programação imperativa, o seu grafo de fluxo de controle é um grafo direcionado no qual os nós representam blocos de instruções existentes no programa e as arestas representam mudanças no fluxo de controle do programa (JORGENSEN; DEVRIES, 2021). A seguir serão discutidos alguns critérios de teste estrutural baseados em fluxo de controle que normalmente são abordados em livros-texto de teste de software.

2.5.1 Critérios de Teste Estrutural Baseados em Fluxo de Controle

Os critérios baseados em fluxo de controle mais comumente descritos na literatura relacionada são descritos a seguir:

- **Critério todos-nós:** requer que cada nó do grafo de fluxo de controle que representa o programa em teste seja alcançado no mínimo uma vez (DELAMARO et al., 2016). Em termos práticos, isso implica que cada instrução do programa sendo testado deve ser executada pelo menos uma vez.
- **Critério todas-arestas**: requer que cada aresta do grafo de fluxo de controle que representa o programa em teste seja alcançada no mínimo uma vez (DELAMARO et al., 2016). Na prática, isso significa que cada desvio de controle do programa em teste deve ser exercitado pelo menos uma vez.
- **Critério todos-caminhos**: requer que todos os caminhos possíveis do grafo de fluxo de controle que representa o programa em teste sejam executados (DELAMARO

et al., 2016). Como apontado por Ammann e Offutt (2016), a execução de todos os caminhos de um programa é impraticável caso o grafo de fluxo de controle que representa o programa em teste possua um ciclo. Uma vez que, o ciclo pode resultar em uma quantidade infinita de caminhos e, consequentemente, uma quantidade infinita de requisitos de teste.

2.6 Técnica de Teste Baseado em Propriedades

O teste baseado em propriedades é uma técnica aleatória de teste de software (HUGHES, 2019), onde: *(i)* é definido um conjunto de propriedades que descrevem o comportamento esperado do sistema em teste; e *(ii)* é gerado aleatoriamente um conjunto de dados de entrada para testar a validade dessas propriedades (LöSCHER; SAGONAS, 2017). É dito que o teste de uma propriedade resultou em sucesso, quando a propriedade é mantida para todos os dados que foram gerados e submetidos ao programa em teste. Caso contrário, é dito que o teste resultou em uma falha. Em caso de falha, o dado de teste que causou a falha, é reduzido para a sua forma mais simples que ainda é capaz de gera-lá, em um processo conhecido como *shrinking*, de forma a facilitar que o desenvolvedor consiga reproduzir a falha encontrada no sistema (PAPADAKIS; SAGONAS, 2011).

A técnica de teste baseado em propriedades foi popularizada por meio da ferramenta `QuickCheck` (CLAESSEN; HUGHES, 2011). Uma ferramenta de teste baseado em propriedades criada com o objetivo de reduzir o custo do teste de sistemas desenvolvidos com a linguagem Haskell. A redução de custo implementada pela ferramenta é resultado de como as propriedades são definidas pelo desenvolvedor e testadas pela ferramenta. Ao escrever testes baseados em exemplos concretos, o desenvolvedor precisa escrever manualmente um caso de teste para cada cenário que deseja testar. Porém, ao utilizar uma ferramenta de teste baseado em propriedades, é necessário apenas definir as propriedades do sistema em teste e especificar os devidos geradores de dados de teste (LöSCHER; SAGONAS, 2017). Assim, cada propriedade será testada com diversos dados de teste, gerados aleatoriamente pela ferramenta de teste baseado em propriedades utilizada.

A técnica de teste baseado em propriedades possibilita a criação de uma suíte de testes de maneira rápida (LöSCHER; SAGONAS, 2017). As propriedades definidas também constituem uma suíte de testes muito mais concisa do que uma suíte de testes construída com vários casos de teste baseados em exemplos concretos. Quando usada de forma apropriada, a técnica de teste baseado em propriedades possibilita a realização de uma atividade de teste mais abrangente (LöSCHER; SAGONAS, 2018): considerando que o sistema em teste será submetido a uma variedade muito maior de dados de teste do que qualquer testador estaria disposto ou seria capaz de escrever manualmente.

Essencialmente, a eficácia da técnica de teste baseado em propriedades está as-

sociada a dois fatores: *(i)* a qualidade das propriedades definidas pelo desenvolvedor; e *(ii)* a diversidade dos dados de teste que serão gerados aleatoriamente pela ferramenta utilizada. A qualidade das propriedades está relacionada à capacidade dos testadores em lidar com o problema do oráculo de teste durante a utilização de uma ferramenta de teste baseado em propriedades (HUGHES, 2019). Para ajudar os testadores, Hughes (2019) e Hebert (2019) apresentaram diversos tipos de propriedades que podem ser identificadas em diferentes sistemas de software. A diversidade dos dados gerados aleatoriamente, por sua vez, está diretamente relacionada à distribuição de dados que é definida junto aos geradores de dados utilizados com cada propriedade.

Normalmente, as ferramentas que automatizam a condução de atividades de teste baseado em propriedades disponibilizam um conjunto de geradores para os tipos primitivos e para as coleções de tipos primitivos da linguagem alvo (LöSCHER; SAGONAS, 2017). Para dar suporte a geração de dados para tipos definidos pelo usuário, ferramentas disponibilizam mecanismos para que o desenvolvedor possa definir geradores customizados. Porém, independente do tipo de gerador utilizado, é permitido e recomendado que desenvolvedores controlem a distribuição de dados de acordo com o conhecimento do domínio do sistema em teste (CLAESSEN; HUGHES, 2011).

A técnica de teste baseado em propriedades apresenta algumas limitações (CLAESSEN; HUGHES, 2011). Uma das limitações é a falta de uma medida de cobertura de teste que indique quando testes suficientes foram executados, ou seja, não existe um critério de parada que determine claramente quando o sistema em teste foi suficientemente testado. É responsabilidade do desenvolvedor investigar a distribuição dos dados de teste e decidir se um número suficiente de testes foi executado. Outra limitação inerente a esse tipo de teste é a dificuldade em gerar aleatoriamente contra-exemplos para casos específicos onde o domínio de entrada do sistema em teste é muito abrangente e os dados de entrada efetivamente capazes de revelar falhas são raros (LöSCHER; SAGONAS, 2017). Com um melhor conhecimento do domínio de entrada do sistema em teste a probabilidade de gerar um contra-exemplo pode ser aumentada. Por exemplo, direcionando o gerador de dados utilizado para regiões específicas do espaço de entrada do sistema em teste, onde uma falha é mais provável de ser identificada. Porém, geradores desse tipo são mais específicos e podem ser mais complicados de implementar (LöSCHER; SAGONAS, 2018).

2.6.1 Problema do Oráculo de Teste na Construção de Propriedades

Ao utilizar um critério de teste funcional ou estrutural para projetar casos de teste, o testador realiza o que normalmente é chamado de teste baseado em exemplos (ANICHE, 2022). O critério de teste utilizado produz um conjunto de requisitos de teste que devem ser satisfeitos por meio da seleção de dados de teste do domínio de entrada do programa em teste (DELAMARO et al., 2016). Para cada um dos dados de teste selecionados,

devem ser definidos os devidos resultados esperados, que juntos com as pré-condições e pós-condições de teste, formam o conjunto de casos de testes. Em termos práticos, um critério de teste possibilita ao testador definir e implementar exemplos concretos de casos de teste. O testador sabe exatamente quais dados de teste serão submetidos ao programa em teste e quais serão os resultados esperados.

Uma das principais diferenças existentes entre o projeto de casos de teste baseado em exemplos para a construção de propriedades refere-se ao fato de que ao utilizar uma biblioteca de teste baseado em propriedades os dados de teste serão escolhidos somente em tempo de execução da biblioteca (ANICHE, 2022). É difícil para o testador determinar quais serão os resultados esperados durante a definição das propriedades, visto que os dados de teste que serão gerados pela biblioteca ainda são desconhecidos e serão definidos somente em tempo de execução.

Segundo Hughes (2019), essa é uma das principais dificuldades encontradas pelos testadores durante a definição de propriedades. Na visão do autor, tal dificuldade está diretamente relacionada à capacidade dos testadores em lidar com o problema do oráculo de teste em um cenário onde os dados de teste não são previamente conhecidos. Hebert (2019) afirma que escrever propriedades eficazes para realização do teste baseado em propriedades é um processo que demanda muita prática e criatividade.

2.6.2 Teste Baseado em Propriedades com a Biblioteca `Hypothesis`

A ferramenta `QuickCheck` (CLAESSEN; HUGHES, 2011) inspirou o desenvolvimento de ferramentas de teste baseado em propriedades em várias linguagens de programação (PARASKEVOPOULOU et al., 2015). Atualmente, bibliotecas que provêem suporte ao teste baseado em propriedades encontram-se disponíveis em diversas linguagens. Porém, como afirmado por Aniche (2022), a interface de programação disponibilizada por cada biblioteca pode variar consideravelmente. Fazendo com que a realização de um estudo sobre o funcionamento básico da biblioteca utilizada seja imprescindível. Diante disso, nesta subseção é realizada uma breve apresentação da biblioteca de teste baseado em propriedades adotada neste trabalho.

A `Hypothesis` é uma biblioteca de teste baseado em propriedades amplamente utilizada para testar sistemas desenvolvidos na linguagem Python (MACIVER et al., 2019). De acordo com dados disponibilizados no PyPIStats[4], apenas no mês de Janeiro de 2023, o projeto da `Hypothesis` contabilizou mais de 4 milhões de downloads. Além de ser utilizada na indústria para apoiar a realização do teste de sistemas de larga escala, MacIver et al. (2019) afirmam que a `Hypothesis` também pode ser utilizada por pesquisadores que desenvolvem software científico na linguagem Python e que precisam aumentar a qualidade dos sistemas de software sendo desenvolvidos. Os autores complementam ao

[4] https://pypistats.org/packages/hypothesis

afirmar que a `Hypothesis` também pode ser utilizada por pesquisadores de teste de software que necessitam de uma plataforma para desenvolver pesquisa sobre o teste baseado em propriedades.

A `Hypothesis` disponibiliza geradores para os principais tipos de dados existentes na linguagem Python. Os autores da biblioteca denominam tais geradores de estratégias, do termo em inglês, *strategies*. O módulo `hypothesis.strategies` implementado na biblioteca disponibiliza as funções que os desenvolvedores devem utilizar para definir estratégias de geração de dados para as propriedades sendo testadas. A listagem a seguir apresenta alguns exemplos dos tipos de geradores de dados que podem ser definidos com as funções disponibilizadas no módulo mencionado.

- ***hypothesis.strategies.booleans()*:** retorna uma estratégia que permite gerar instâncias do tipo `bool`.
- ***hypothesis.strategies.characters()*:** retorna uma estratégia que permite gerar caracteres.
- ***hypothesis.strategies.dates()*:** retorna uma estratégia que permite gerar instâncias do tipo `datetime.date`.
- ***hypothesis.strategies.decimals()*:** retorna uma estratégia que permite gerar instâncias do tipo `decimal.Decimal`.
- ***hypothesis.strategies.dictionaries(keys, values)*:** retorna uma estratégia que permite gerar dicionários de dados contendo chaves extraídas do argumento `keys` e valores extraídos do argumento `values`.
- ***hypothesis.strategies.floats()*:** retorna uma estratégia que permite gerar valores numéricos de ponto flutuante.
- ***hypothesis.strategies.integers()*:** retorna uma estratégia que permite gerar valores numéricos do tipo inteiro.
- ***hypothesis.strategies.lists(elements)*:** retorna uma estratégia que permite gerar listas contendo elementos obtidos da coleção `elements`.
- ***hypothesis.strategies.sampled_from(elements)*:** retorna uma estratégia que permite gerar qualquer elemento presente na coleção `elements`.
- ***hypothesis.strategies.sets(elements)*:** retorna uma estratégia que permite gerar conjuntos contendo elementos obtidos da coleção *elements*.
- ***hypothesis.strategies.tuples()*:** retorna uma estratégia que permite gerar tuplas de dados.

- ***hypothesis.strategies.text(alphabet)*:** retorna uma estratégia que permite gerar strings com caracteres extraídos de `alphabet`.

A implementação de propriedades de teste com a biblioteca `Hypothesis` é realizada em duas partes: primeiro, deve ser implementado uma função ou um método de teste utilizando um *framework* de teste automatizado, porém, a função ou o método implementado deve possuir argumentos de entrada definidos para habilitar a geração de dados de teste com a `Hypothesis`, posteriormente, o testador deve utilizar o *decorator* `@given`, cuja principal função é especificar como a `Hypothesis` deverá gerar os dados de teste que serão fornecidos como argumentos de entrada para a função ou método de teste em questão. A seguir são apresentados dois exemplos de propriedades de teste que foram obtidos na documentação[5] oficial da própria `Hypothesis`.

```
1 from hypothesis import given, strategies as st
2
3 @given(st.integers(), st.integers())
4 def test_ints_are_commutative(x, y):
5     assert x + y == y + x
```

Listagem 2.2 – Código de teste que valida se a propriedade comutativa da adição é respeitada para todos os valores inteiros gerados pela `Hypothesis`.

```
1 from hypothesis import given, strategies as st
2
3 @given(st.lists(st.integers()))
4 def test_reversing_twice_gives_same_list(xs):
5     ys = list(xs)
6     ys.reverse()
7     ys.reverse()
8     assert xs == ys
```

Listagem 2.3 – Código de teste de uma propriedade que valida se a aplicação sucessiva de duas operações de inversão de elementos de uma lista resulta na lista original.

A Listagem 2.2 especifica uma propriedade de teste que verifica se dois números inteiros respeitam a propriedade comutativa da adição e a Listagem 2.3 especifica uma propriedade que verifica se a aplicação sucessiva de duas operações de inversão de elementos, sobre uma lista de números inteiros, resultará na lista original. Para implementar ambas as propriedades foi necessário importar o *decorator* `@given` e o módulo `strategies`. A utilização conjunta do *decorator* `@given` com os diferentes tipos de geradores de dados disponibilizados no módulo `strategies` específica como a `Hypothesis` deverá gerar os dados de teste que serão fornecidos para os argumentos que foram definidos em cada uma das propriedades. No caso da propriedade mostrada na Listagem 2.2, foram utilizados dois geradores de valores inteiros que serão fornecidos para os argumentos `x` e `y`. No caso da

[5] https://hypothesis.readthedocs.io/en/latest/quickstart.html

propriedade da Listagem 2.3, foi definido a geração de listas contendo valores do tipo inteiro que são fornecidas como argumento para `xs`. É importante mencionar que no último exemplo dois tipos diferentes de geradores de dados foram utilizados para possibilitar a geração do tipo necessário para condução dos testes.

O comportamento padrão da `Hypothesis` é o de gerar até 100 casos de teste para cada propriedade implementada. Caso nenhuma das 100 execuções de casos de teste de uma determinada propriedade resulte em falha, a ferramenta encerra a geração de casos de teste e sinaliza que a execução da propriedade em questão foi bem sucedida. Porém, caso uma falha seja identificada durante as execuções de casos de teste, a `Hypothesis` encerra a fase de geração de caso de teste e inicia a realização de uma fase denominada *shrinking*. *Shrinking* é o processo realizado a fim de tentar encontrar outro caso de teste que também seja capaz de fazer a propriedade resultar em falha, porém, que seja mais legível para o testador. Casos de teste legíveis são necessários para facilitar a realização de atividades de depuração, permitindo que o testador identifique o defeito que gerou a falha.

A `Hypothesis` permite modificar o seu comportamento padrão por meio da utilização do *decorator* `@settings`. Para aumentar a quantidade de casos de teste a ser gerado para uma determinada propriedade o *decorator* `@settings` deve ser utilizado juntamente com o atributo `max_examples` conforme a seguir: `@settings(max_examples=500)`. Além disso, a ferramenta também permite utilizar o *decorator* `@settings` para alterar toda a sequência lógica do processo de teste de uma determinada propriedade.

O processo de teste de uma propriedade implementada com a `Hypothesis` é dividido em seis fases distintas. O testador possui flexibilidade para escolher quais fases executar por meio da utilização conjunta do *decorator* `@settings` com o atributo `phases` e com a *enumeration* `Phase`. Por exemplo, o seguinte comando pode ser utilizado para executar apenas a fase de geração de casos de teste de uma determinada propriedade: `@settings(phases=[Phase.generate])`. Ao executar uma propriedade que possua essa configuração, a `Hypothesis` ignora todas as outras fases durante o processo de teste, dessa forma, a fase que corresponde ao procedimento de *Shrinking* não seria executada. A listagem a seguir descreve todas as seis fases que compõem o processo de teste de uma propriedade implementada com a `Hypothesis`.

- ***Phase.explicit*:** fase que permite executar exemplos de casos de teste fornecidos explicitamente pelo testador. Os exemplos devem ser fornecidos com o *decorator* `@example`. A `Hypothesis` executará todos os exemplos fornecidos explicitamente pelo testador antes de tentar gerar qualquer caso de teste. Caso algum dos exemplos fornecidos falhe, a ferramenta encerra o processo de teste e não avança para as outras fases.

- ***Phase.reuse*:** fase que permite reexecutar automaticamente casos de teste que resultaram em falhas em sessões de teste realizadas previamente.
- ***Phase.generate*:** fase executada para realizar a geração de casos de teste. Os dados de teste são gerados de acordo com as estratégias de geração de dados definidas na propriedade.
- ***Phase.target*:** fase que permite executar uma variação da técnica de teste baseado em propriedades conhecida como *targeted property-based testing* (LöSCHER; SAGONAS, 2017).
- ***Phase.shrink*:** fase executada para realizar o processo de *shrinking*.
- ***Phase.explain*:** fase executada para tentar identificar o defeito que resultou na falha observada. Caso identificado, a `Hypothesis` reporta o defeito ao testador.

2.7 Considerações Finais

Neste capítulo foi realizada uma discussão sobre os conceitos, as técnicas e alguns critérios de teste de software que são necessários ou que podem contribuir para a compreensão da proposta deste trabalho. O capítulo iniciou com uma apresentação dos termos e conceitos básicos de teste de software que são encontrados na literatura, como por exemplo, foram apresentadas as definições de: defeito, engano, erro, falha, *bug* e caso de teste. Também foi explicado logo no início do capítulo sobre a importância da escolha de dados de teste eficazes para realização da atividade de teste e sobre a necessidade de oráculos de teste para identificação dos resultados esperados da realização da atividade de teste. Para consolidar essa discussão inicial foi apresentado o modelo de defeito/falha de software conforme descrito por Ammann e Offutt (2016).

Foi prosseguida com a discussão sobre teste de software por meio da apresentação dos diferentes níveis de teste que são descritos em livros textos de teste de software, como por exemplo: os testes de unidade, integração e aceitação. Também foi destacado a importância da automatização de testes que possibilita que os casos de teste sejam transformados em testes executáveis (AMMANN; OFFUTT, 2016). Ainda sobre automatização de testes, também foi brevemente discutido sobre o *framework* de testes automatizado `pytest` que foi utilizado durante o desenvolvimento deste trabalho.

O capítulo foi finalizado após a apresentação de três técnicas: a técnica de teste funcional; a técnica de teste estrutural; e a técnica de teste baseado em propriedades. A técnica de teste baseado em propriedade é um dos pilares da proposta deste trabalho. Por esse motivo, ela foi discutida de maneira mais ampla que as demais. Além disso, também foi realizada uma discussão sobre a biblioteca `Hypothesis` de teste baseado em propriedades que foi adotada como referência para o desenvolvimento deste trabalho.

3 Aprendizado de Máquina

Neste capítulo é realizada uma discussão sobre os conceitos, as técnicas e alguns algoritmos de aprendizado de máquina. Porém, antes de prosseguir com a discussão dos conceitos e técnicas é necessário introduzir a definição de alguns termos relacionados ao aprendizado de máquina que são amplamente utilizados neste documento. A seguir os termos usados no decorrer do documento são elucidados:

- **Amostra de dados:** um registro de dados que armazena informações sobre um objeto (ZHANG et al., 2022).
- **Característica:** uma propriedade mensurável de uma amostra de dados (ZHANG et al., 2022).
- **Rótulo:** um valor ou uma categoria que é atribuída para cada amostra de dados (ZHANG et al., 2022). Também pode ser denominado de **classe** da amostra de dados (MÜLLER; GUIDO, 2016). Neste documento, os termos **rótulo** e **classe** de uma amostra de dados são utilizados de maneira intercambiável.
- **Atributo:** em aprendizado de máquina um atributo representa um tipo de dados. Porém, é muito comum os termos **característica** e **atributo** serem utilizados de maneira intercambiável (GÉRON, 2019). Neste documento, o termo **atributo** é utilizado para representar tanto uma **característica** quanto um **rótulo**.
- **Conjunto de dados:** um conjunto de amostras de dados que é utilizado para construir ou avaliar um modelo de aprendizado de máquina (ZHANG et al., 2022).
- **Modelo:** o artefato que codifica a lógica de decisão (lógica de predição) que é aprendida por meio da realização de um processo de treinamento utilizando um conjunto de dados de treinamento e um algoritmo de aprendizado de máquina (ZHANG et al., 2022). Neste documento os termos **modelo** e **modelo de aprendizado de máquina** são utilizados de maneira intercambiável.

O objetivo da utilização das técnicas e algoritmos de aprendizado de máquina pode ser resumido como a construção de um modelo que receberá uma entrada relacionada a determinado fenômeno e produzirá uma saída com o resultado desejado (MOHAMMED et al., 2016). O modelo construído é considerado uma aproximação do fenômeno ou processo que as máquinas devem reproduzir. Por ser uma aproximação, é possível que o modelo produza uma resposta incorreta para uma determinada entrada. Todavia, o objetivo é fazer com que o modelo construído produza respostas corretas na maioria das vezes.

Portanto, normalmente, são utilizados métodos e métricas de avaliação de desempenho para estimar a probabilidade com que um modelo é capaz de produzir respostas corretas (GÉRON, 2019).

O restante deste capítulo está organizado conforme a seguir. Na Seção 3.1 é apresentada uma contextualização sobre o aprendizado de máquina. A Seção 3.2 discute sobre o aprendizado de máquina supervisionado e aborda os algoritmos de árvore de decisão e o algoritmo k-vizinhos mais próximos. Na Seção 3.3 é discutido sobre o processo de treinamento de um modelo de aprendizado de máquina. A Subseção 3.3.1 discute sobre alguns métodos utilizados para realização da avaliação de desempenho de classificadores. Na Subseção 3.3.2 são abordadas algumas métricas utilizadas durante a avaliação de desempenho de classificadores. A Seção 3.4 aborda a necessidade da utilização de bibliotecas ou *frameworks* de aprendizado de máquina e discute sobre o *framework* `scikit-learn`. Por fim, na Seção 3.5 são apresentadas as considerações finais.

3.1 Aprendizado de Máquina

O aprendizado de máquina é normalmente definido como o campo de estudo que fornece aos computadores a capacidade de aprender sem serem explicitamente programados para isso (SAMUEL, 1959). Yeturu (2020) apresenta uma definição semelhante, ao definir o aprendizado de máquina como o estudo científico dos algoritmos que os sistemas de computador utilizam para executar uma tarefa específica de forma eficiente sem precisar utilizar várias instruções explícitas. O autor complementa ao afirmar que os algoritmos de aprendizado de máquina possuem um modelo matemático embutido que utiliza amostras de dados para realizar predições ou tomar decisões.

De forma semelhante, Dol e Geetha (2021) definiram o aprendizado de máquina como um método que possibilita que computadores aprendam e se aprimorem por meio da utilização de dados, permitindo que as aplicações de software sejam capazes de realizar predições com mais precisão sem a necessidade de serem programadas explicitamente. Os autores também destacam que as técnicas de aprendizado de máquina são caracterizadas pela utilização de dados que devem ser fornecidos como entradas e pela aplicação de análise estatística sobre esses dados para realização de predições.

Os algoritmos de aprendizado de máquina são normalmente divididos em quatro abordagens de aprendizado de máquina distintas (MOHAMMED et al., 2016): *(i)* aprendizado supervisionado; *(ii)* aprendizado não supervisionado; *(iii)* aprendizado semi-supervisionado; e *(iv)* aprendizado por reforço. Cada uma dessas técnicas são utilizadas para resolver problemas dinstintos (SARKER, 2021). Como o tema deste trabalho refere-se ao teste de modelos de aprendizado de máquina supervisionado, em específico, ao teste de classificadores, na próxima seção essa abordagem é descrita.

3.2 Aprendizado Supervisionado

O aprendizado de máquina supervisionado é o tipo de aprendizado de máquina mais amplamente utilizado (JORDAN; MITCHELL, 2015). Neste tipo de aprendizado é necessário um conjunto de dados de treinamento constituído por amostras rotuladas (ZHANG et al., 2022). Cada amostra do conjunto de treinamento é composta por várias características (atributos) e um rótulo (classe) (GU et al., 2021). Normalmente, as características de cada amostra são representadas por um vetor x_i e o rótulo por um valor y_i. O conjunto de todos os vetores de atributos, também chamado de conjunto de entrada, normalmente é representado por $\mathcal{X}$ e o conjunto de todos os rótulos, também chamado de conjunto de saída ou conjunto de resultados, normalmente é representado por $\mathcal{Y}$ (TELIKANI et al., 2021). De posse dos conjuntos $\mathcal{X}$ e $\mathcal{Y}$, algoritmos de aprendizado supervisionado buscam encontrar uma função capaz de mapear os dados de entrada para a saída desejada (VERBRAEKEN et al., 2020). Desse modo, essa função de mapeamento pode ser utilizada para realizar predições referentes a novos dados de entrada, cujos resultados ainda são desconhecidos (THOMAS; GUPTA, 2020).

O objetivo do aprendizado supervisionado pode ser definido formalmente conforme a seguir. Seja $\mathcal{X} = (x_1, ..., x_m)$ o conjunto de treinamento composto por amostras não rotuladas. Seja $\mathcal{Y} = (c(x_1), ..., c(x_m))$ o conjunto de rótulos correspondentes a cada amostra de treinamento x_i. Seja o conceito $\mathcal{C} : \mathcal{X} \rightarrow \mathcal{Y}$ o padrão de mapeamento verdadeiro existente de $\mathcal{X}$ para $\mathcal{Y}$. O objetivo do aprendizado supervisionado é aprender um padrão de mapeamento que seja similar ao conceito verdadeiro $\mathcal{C}$, normalmente com um pequeno erro de generalização (ZHANG et al., 2022). Essencialmente, o objetivo do aprendizado supervisionado é construir um modelo h baseado em $\mathcal{X}$ e $\mathcal{Y}$.

Os algoritmos de aprendizado supervisionado são normalmente utilizados para resolver dois tipos de problemas: *(i)* problemas de classificação e *(ii)* problemas de regressão (MOHAMMED et al., 2016). Quando o objetivo é utilizar um algoritmo supervisionado para atribuir uma classe (categoria) para cada amostra de dados é dito que o problema sendo abordado é um problema de classificação. Quando o objetivo é realizar a predição de um valor contínuo, então o problema é de regressão. O problema de classificação ainda pode ser dividido em três tipos de problemas específicos (SARKER, 2021): *(i)* problemas de classificação binária; *(ii)* problemas de classificação multiclasse; e *(iii)* problemas de classificação multirótulo. A definição de cada tipo específico de problema de classificação é dada conforme a seguir.

- **Classificação binária:** refere-se ao problema de classificação em que dada uma amostra de dados o objetivo é determinar a qual classe, entre apenas duas classes possíveis, uma determinada instância pertence (SARKER, 2021).
- **Classificação multiclasse:** refere-se ao problema de classificação em que dada

uma amostra de dados o objetivo é determinar qual única classe entre várias classes possíveis uma dada instância pertence (SARKER, 2021).

- **Classificação multirrótulo:** refere-se ao problema de classificação em que cada amostra de dados pode pertencer a várias classes, ou seja, pode possuir mais de um rótulo. Por isso, o problema de classificação multirótulo é considerado uma generalização do problema de classificação multiclasse (SARKER, 2021).

O teste de modelos de classificação binária e multiclasse são abordados diretamente neste trabalho. Porém, o teste de modelos de classificação multirrótulo não faz parte do escopo do trabalho. Nas subseções seguintes descreve-se os dois algoritmos de aprendizado de máquina supervisionado que foram utilizados neste trabalho: o algoritmo de árvore de decisão e o algoritmo k-vizinhos mais próximos.

3.2.1 Árvore de Decisão

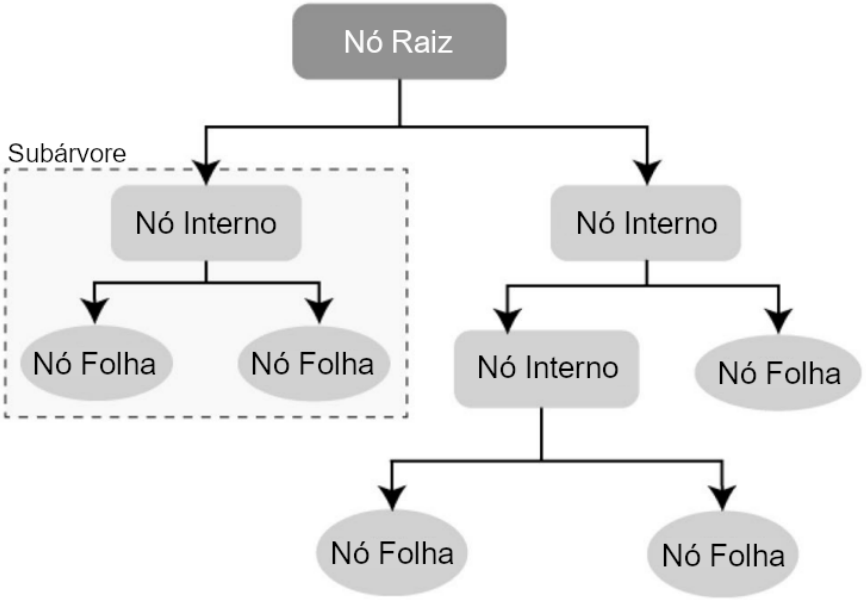

Figura 4 – Estrutura de uma árvore de decisão. Adaptado de (SARKER, 2021).

A Figura 4 apresenta a estrutura de uma árvore de decisão. Conforme pode ser observado, uma árvore de decisão é um grafo acíclico direcionado em que cada nó pode ser um nó interno (nó de divisão) com dois ou mais sucessores ou um nó folha (LORENA et al., 2021). O primeiro nó interno da árvore é denominado de nó raiz. Além disso, cada divisão realizada por um nó interno resulta na criação de uma subárvore. Conforme mostrado na Figura 4, a divisão realizada pelo nó raiz resultou na criação de subárvores.

Os algoritmos de árvore de decisão constituem um tipo de algoritmo não paramétrico de aprendizado de máquina supervisionado que são utilizados tanto em problemas

de classificação quanto em problemas de regressão (NICULAESCU, 2018). A ideia geral empregada pelos algoritmos de árvore de decisão pode ser resumida como a construção de uma árvore $\mathcal{T}$ a partir de um conjunto de amostras de dados de treinamento $\mathcal{S}$. Normalmente, o algoritmo utilizado inicia a construção da árvore de decisão por meio da criação de um único nó. Caso todas as amostras de dados do conjunto $\mathcal{S}$ possuam o mesmo rótulo $\mathcal{C}$ (ou seja, pertencerem à mesma classe $\mathcal{C}$), então o único nó que foi criado deve ser considerado como um nó folha e portanto deve receber o rótulo $\mathcal{C}$ (a classe $\mathcal{C}$). Caso contrário, o algoritmo de árvore de decisão seleciona o atributo mais informativo do conjunto $\mathcal{S}$ para construir subárvores até que não seja mais possível construir nenhuma subárvore (PHALAK et al., 2014).

Para identificar qual é o atributo mais informativo que deverá ser utilizado para construir uma subárvore (realizar uma divisão no conjunto de amostra de dados de treinamento $\mathcal{S}$) normalmente é utilizada a equação do cálculo de entropia (Equação 3.1). Com a entropia calculada, é realizado o cálculo do ganho de informação (GI) oferecido por cada atributo (Equação 3.2). Então, o atributo capaz de oferecer o maior ganho de informação é utilizado para construção de uma subárvore.

$$Entropia(S) = -\sum_{x \in S} P(x) log_2 P(x) \tag{3.1}$$

$$GI(S, A) = Entropia(S) - \sum_{v \in A} \frac{|S_v|}{|S|} \times Entropia(S_v) \tag{3.2}$$

Diferentes algoritmos foram propostos na literatura para possibilitar a construção de árvores de decisão (PHALAK et al., 2014). A seguir o algoritmo CART é descrito.

Algoritmo CART

O algoritmo *Classification And Regression Trees* (CART) foi desenvolvido por Breiman et al. (1984), como o próprio nome sugere, o algoritmo CART possui a capacidade de construir modelos de árvore de decisão tanto para problemas de classificação quanto para problemas de regressão (RATHEE; MATHUR, 2013). O tipo de árvore de decisão construída com o algoritmo CART é determinado com base no tipo da variável dependente do conjunto de dados de treinamento utilizado. Caso a variável dependente seja uma variável categórica o CART constrói uma árvore de classificação, caso seja uma variável numérica, é construído uma árvore de regressão (BRIJAIN et al., 2014).

Uma característica que difere o CART de outros algoritmos de árvore de decisão refere-se a separação binária de atributos que o algoritmo CART utiliza (ROKACH; MAIMON, 2005). Todas as divisões realizadas por nós internos de uma árvore de decisão construída com o algoritmo CART são divisões binárias. Como consequência, o algoritmo sempre constrói uma árvore de decisão binária (ALMANA; AKSOY, 2014): uma árvore

binária é aquela em que cada nó interno da árvore possui exatamente duas arestas de saída (ROKACH; MAIMON, 2005).

Outra característica do algoritmo CART refere-se a utilização do índice Gini (Equação 3.3) como medida de impureza para selecionar o atributo que deve compor o próximo nó da árvore (ALMANA; AKSOY, 2014). O atributo capaz de promover a maior redução de impureza é utilizado para dividir as amostras desse nó. O CART também possui um procedimento de poda da árvore denominado de *cost complexity pruning* que é utilizado para evitar o sobreajuste *(overfitting)* do modelo de árvore de decisão (ROKACH; MAIMON, 2005). Além disso, o algoritmo é capaz de manipular atributos com valores categóricos, valores numéricos contínuos e valores faltantes (BRIJAIN et al., 2014).

$$Gini(E) = 1 - \sum_{i=1}^{c} p_i^2 \tag{3.3}$$

3.2.2 K-Vizinhos Mais Próximos

Os algoritmos de aprendizado de máquina supervisionado baseados em distância utilizam uma função de distância para realizar a classificação das amostras de dados por meio do cálculo da distância existente entre a amostra de teste e as amostras de treinamento (ALI et al., 2019). Um algoritmo baseado no uso de medidas de distância que é amplamente utilizado é o algoritmo *K-Vizinhos mais Próximos*, do termo em inglês, *K-Nearest Neighbor* (K-NN) (COVER; HART, 1967). A noção de classificação utilizada pelo algoritmo K-NN é aplicada de forma bastante direta: uma amostra de dados de teste é classificada com base na classe dos K vizinhos mais próximos (CUNNINGHAM; DELANY, 2021).

Na Figura 5 é ilustrada a ideia básica de funcionamento do algoritmo K-NN ao ser utilizado para solucionar um problema de classificação binária (classes O e X) em um espaço de características de duas dimensões (características C1 e C2). Por meio da utilização do K-NN, foi obtido um classificador 3-NN que tem como objetivo realizar a classificação das amostras de teste q_1 e q_2 apresentadas na imagem. Neste exemplo, a classificação da amostra de teste q_1 é realizada de forma direta: como os três vizinhos mais próximos de q_1 pertencem à classe O, a amostra q_1 é imediatamente classificada como pertencendo à classe O. Porém, a situação para a amostra de teste q_2 é um pouco mais complicada: q_2 possui dois vizinhos que pertencem a classe X e um vizinho que pertence à classe O. Para determinar a classe da amostra q_2, o classificador 3-NN pode utilizar voto majoritário (votação por maioria simples) ou voto ponderado em relação a distância da amostra de teste e as amostras de treinamento (CUNNINGHAM; DELANY, 2021).

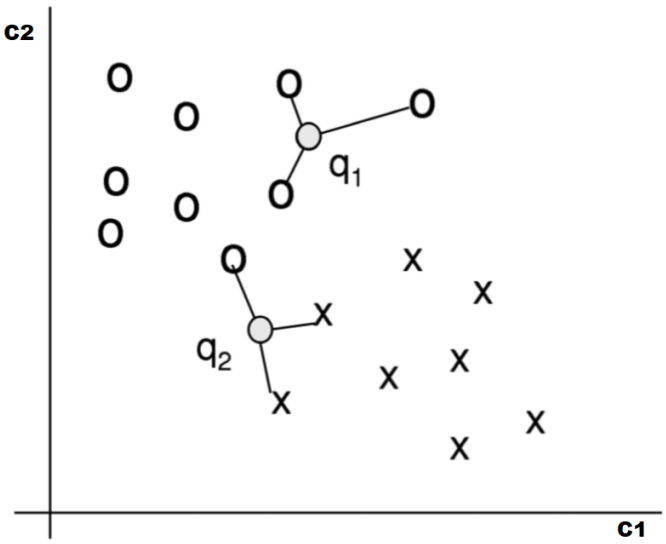

Figura 5 – Classificação realizada por um classificador 3-NN em um espaço de características de duas dimensões. Adaptado de (CUNNINGHAM; DELANY, 2021).

De acordo com o exemplo apresentado na Figura 5, pode ser observado que o processo de classificação do algoritmo K-NN possui duas etapas principais (CUNNINGHAM; DELANY, 2021): a primeira refere-se a identificação dos K vizinhos mais próximos da amostra de teste; e a segunda consiste na determinação da classe da amostra de teste com base nas classes dos vizinhos mais próximos. Para medir a semelhança (distância) existente entre a amostra de teste e as amostras de dados de treinamento normalmente é realizado o cálculo da distância Euclidiana (JIANG et al., 2007). Para determinar a classe da amostra de teste, normalmente é realizada uma votação simples entre as classes das K amostras de treinamento que forem mais semelhantes a amostra de teste. Por fim, o valor de K utilizado para determinar a quantidade de vizinhos mais próximos é um número artificial que deve ser fornecido como hiperparâmetro para o algoritmo (ALI et al., 2019).

3.3 Processo de Treinamento de um Modelo de Aprendizado de Máquina

Amershi et al. (2019) apresentaram um processo de aprendizado de máquina que consiste em nove etapas distintas. Conforme mostrado na Figura 6, o processo de treinamento de um modelo inicia com a etapa de análise de requisitos do modelo. Nesta etapa, os projetistas do sistema devem identificar quais funcionalidades dependem ou podem ser aprimoradas com a utilização de aprendizado de máquina. Adicionalmente, deve ser definido nesta etapa quais são os tipos de modelos mais adequados para tratar o problema abordado pelo sistema. Durante a etapa de coleta de dados deve ser realizada uma busca por conjuntos de dados relacionados ao domínio do problema tratado pelo sistema. Como resultado, podem ser identificados conjuntos de dados constituídos por dados coletados

internamente dentro da própria organização ou conjuntos de dados de domínio público. A etapa de limpeza de dados consiste na realização da limpeza dos conjuntos de dados: devem ser removidas as amostras que possam conter ruídos ou qualquer outro tipo de inconsistência.

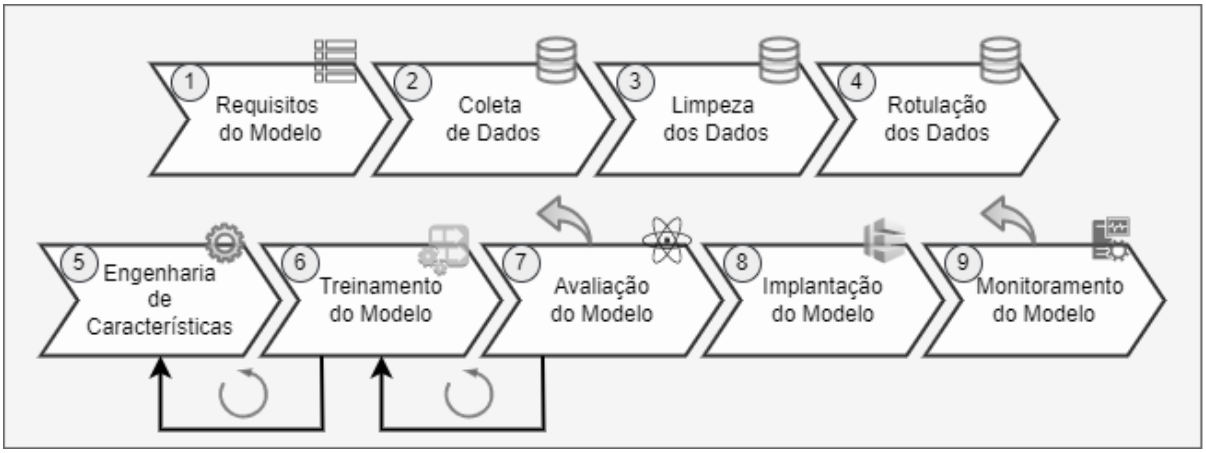

Figura 6 – Processo de treinamento de um modelo de aprendizado de máquina. Adaptado de (AMERSHI et al., 2019).

Conforme mencionado, para possibilitar a construção de um modelo, os algoritmos de aprendizado supervisionado necessitam que o conjunto de dados utilizado contenha os rótulos que determinam a classe de cada amostra de dados existente no conjunto (AMERSHI et al., 2019). Por exemplo, um desenvolvedor pode ter em mãos um conjunto de imagens que ainda não foram rotuladas com os objetos presentes em cada imagem. Conforme mostrado na Figura 6, na etapa de rotulagem de dados rótulos devem ser atribuídos para cada instância do conjunto de dados. Os rótulos podem ser fornecidos pelos próprios desenvolvedores ou por especialistas de domínio do sistema. Na etapa de engenharia de características (*feature engineering*) são realizadas as atividades necessárias para seleção e extração das características mais informativas que existem nos conjuntos de dados e que serão utilizadas para construir o modelo (GÉRON, 2019).

Durante a realização da etapa de treinamento do modelo pode ser necessário realizar uma atividade normalmente chamada de seleção de modelo (MÜLLER; GUIDO, 2016). Esta atividade consiste no treinamento de diferentes modelos de forma iterativa, em cada iteração, deve ser fornecido para o algoritmo de aprendizado de máquina utilizado uma combinação distinta de valores de hiperparâmetros. O objetivo é selecionar os valores de hiperparâmetros que possibilitam treinar o modelo que oferece o melhor desempenho de generalização (AMERSHI et al., 2019). O modelo treinado com os melhores valores de hiperparâmetros identificados deve ser avaliado na etapa de avaliação do modelo utilizando amostras de dados de teste diferentes das amostras que foram utilizadas no seu processo de treinamento. Conforme mostrado na Figura 6, se o desempenho apresentado pelo modelo for satisfatório ele pode ser disponibilizado para utilização dos usuários finais por meio da realização da etapa de implantação do modelo. Após a implantação, o

modelo deve ser constantemente monitorado para identificar se o comportamento exibido pelo modelo é adequado em relação às requisições de seus usuários.

Amershi et al. (2019) destacam que apesar do processo de treinamento de modelos de aprendizado de máquina mostrado na Figura 6 parecer linear, no mundo real, dificilmente este processo é linear. De fato, é comum que este processo seja não linear devido aos diversos iteracões que podem existir durante a sua realização. Por exemplo, as setas superiores apresentadas em cima das etapas de avaliação e monitoramento do modelo indicam que durante a realização dessas etapas o processo pode ser retornado para qualquer uma das etapas anteriores. Situação que pode acontecer rotineiramente caso seja identificado que o modelo não apresenta o desempenho esperado (durante a avaliação) ou que apresenta um comportamento inesperado (durante o monitoramento). A avaliação de desempenho de um modelo de aprendizado de máquina pode ser realizada utilizando diferentes métodos e métricas de avaliação (MÜLLER; GUIDO, 2016). Nas Subseções 3.3.1 e 3.3.2 apresenta-se uma discussão sobre os métodos e as métricas de avaliação de desempenho de classificadores.

3.3.1 Métodos de Avaliação de Desempenho de Classificadores

Uma das principais etapas existentes no processo de treinamento de um modelo de aprendizado de máquina é a etapa de avaliação de desempenho do modelo (RASCHKA; MIRJALILI, 2017). Porém, os dados que foram utilizados para treinar o modelo não devem ser utilizados para realizar a avaliação de desempenho (GÉRON, 2019). Isso ocorre porque um modelo é capaz de reconhecer facilmente todas as amostras de dados que foram utilizadas para realizar o seu treinamento. Portanto, o modelo é capaz de realizar uma predição correta para qualquer uma dessas amostras. Como o objetivo da avaliação é obter um indicador da capacidade de generalização do modelo, ou seja, obter um indicador de qual será o desempenho do modelo quando exposto a dados desconhecidos.

Método *holdout* com divisão em conjuntos de treinamento e testes

O método *holdout* é um método clássico muito utilizado para estimar o desempenho de generalização de modelos de aprendizado de máquina (RASCHKA; MIRJALILI, 2017). Ao utilizar o método *holdout*, o conjunto de dados inicial é dividido em dois conjuntos distintos: um conjunto de dados de treinamento e um conjunto de dados de teste (GÉRON, 2019). O conjunto de treinamento, como o próprio nome sugere, deve ser utilizado para realização do treinamento do modelo. O conjunto de dados de teste deve ser utilizado para realizar a avaliação de desempenho do modelo.

Método *holdout* com divisão em conjuntos de treinamento, validação e testes

A utilização do método *holdout* que separa o conjunto de dados inicial em dois conjuntos, de treino e teste, não é recomendada quando é necessário realizar a atividade

de seleção de modelo. A reutilização sucessiva do conjunto de teste em cada iteração da seleção de modelo pode fazer com que, inevitavelmente, o conjunto de teste torne-se parte dos dados de treinamento (RASCHKA; MIRJALILI, 2017). Depois de sucessivas iterações, existe a possibilidade de ser treinado um modelo sobreajustado aos dados de teste. Com isso, esse modelo provavelmente apresentará um bom desempenho devido ao sobreajuste ao conjunto de teste (*overfitting*) e não devido à seleção de valores de hiperparâmetros adequados.

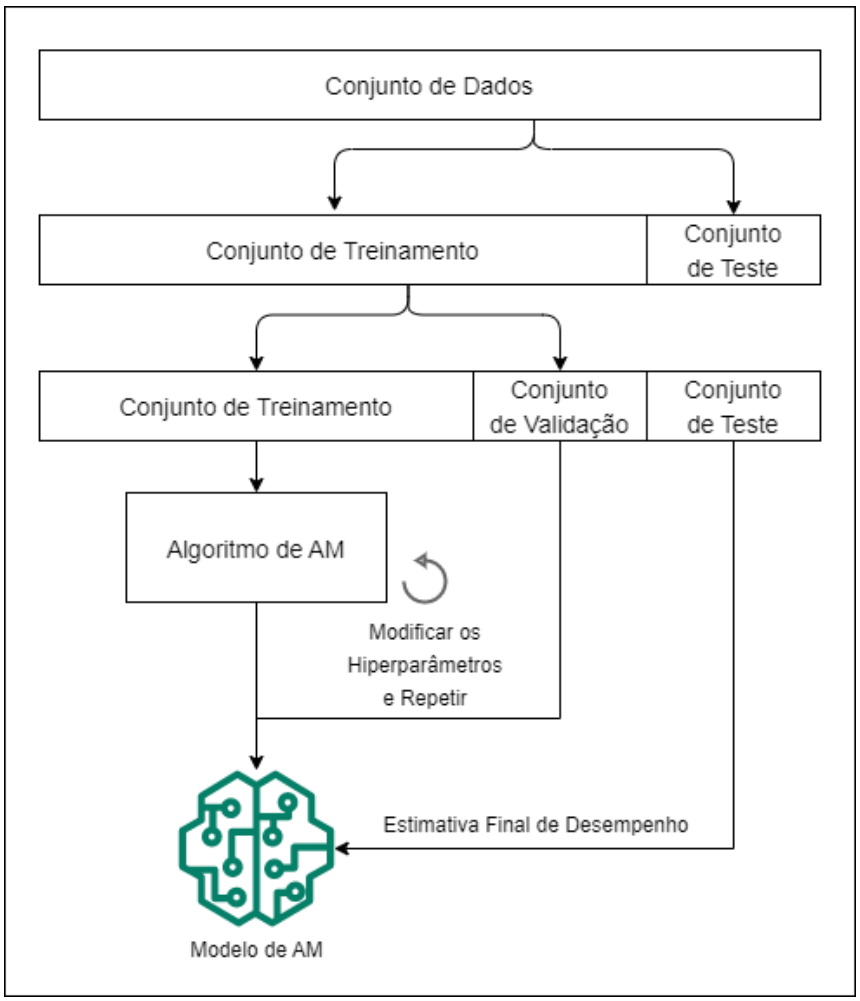

Figura 7 – Método *holdout* com divisão do conjunto de dados inicial nos conjuntos de treinamento, validação e teste. Adaptado de (RASCHKA; MIRJALILI, 2017).

Uma maneira mais recomendada de utilização do método *holdout* conjuntamente com a realização da seleção de modelo é por meio da divisão do conjunto de dados inicial em três conjuntos distintos (RASCHKA; MIRJALILI, 2017): um conjunto de dados de treinamento, um conjunto de dados de validação e um conjunto de dados de teste. A Figura 7 exemplifica essa maneira de utilização do método *holdout*. Conforme pode ser visualizado, o conjunto de dados de treinamento deve ser utilizado para treinar iterativamente cada modelo durante a realização da seleção de modelo. O conjunto de dados de validação deve ser utilizado para avaliar o desempenho desses modelos, ou seja, o con-

junto de dados de validação deve ser utilizado para avaliar o desempenho dos modelos treinados com diferentes configurações de hiperparâmetros. Durante a realização da seleção de modelo, o conjunto de dados de teste não deve ser utilizado. Depois de selecionar a melhor configuração de hiperparâmetros, o conjunto de dados de teste deve ser utilizado para realizar a avaliação de desempenho do modelo treinado com os melhores valores de hiperparâmetros (GÉRON, 2019).

De acordo com Albon (2018) o método *holdout* apresenta duas grandes desvantagens. A primeira é devido a estimativa de desempenho obtida com a avaliação do modelo ser muito sensível à forma como é realizada a divisão do conjunto de dados inicial, ou seja, a estimativa de desempenho pode variar de acordo com as amostras de dados existentes nos conjuntos de dados de treinamento, validação e teste. A segunda desvantagem está relacionada ao treinamento do modelo, que não é realizado utilizando todas as amostras de dados disponíveis e, consequentemente, também não é testado utilizando todas as amostras de dados. Um método mais robusto que busca contornar essas desvantagens é conhecido como validação cruzada por *k-fold* (RASCHKA; MIRJALILI, 2017).

Validação cruzada por *k-fold*

A validação cruzada é um método estatístico que oferece mais robustez e precisão do que o método *holdout* ao ser utilizado para avaliar o desempenho de generalização de modelos de aprendizado de máquina (MÜLLER; GUIDO, 2016). Na validação cruzada por *k-fold*, o conjunto de dados de treinamento é dividido aleatoriamente em k partes, também chamadas de *folds*. Após a divisão, $k-1$ partes devem ser utilizadas para realizar o treinamento de um modelo e a parte restante deve ser utilizada para realizar a avaliação de desempenho. Este procedimento deve ser repetido k vezes, resultando no treinamento de k modelos, e consequentemente, na obtenção de k estimativas de desempenho (uma para cada modelo treinado). Em seguida, para obter um único valor de estimativa de desempenho é realizado o cálculo do desempenho médio dos k modelos que foram treinados (RASCHKA; MIRJALILI, 2017).

Evidências empíricas mostram que o número 10 é um valor adequado para k (RASCHKA; MIRJALILI, 2017). Porém, caso o conjunto de dados seja muito grande, pode ser escolhido um valor menor. A Figura 8 apresenta uma visão geral do funcionamento do método de validação cruzada por *10-folds*. Conforme pode ser observado, o conjunto de dados de treinamento deve ser dividido em 10 partes e, durante 10 iterações, nove partes devem ser utilizadas para realizar o treinamento de um modelo e a parte restante deve ser considerada como o conjunto de testes que deve ser utilizado para realizar a avaliação desse modelo. Além disso, o desempenho estimado E_i de cada modelo deve ser utilizado para calcular o desempenho médio estimado E.

A utilização da validação cruzada possibilita identificar de forma mais precisa se um modelo apresenta um bom desempenho de generalização para todas as amostras de

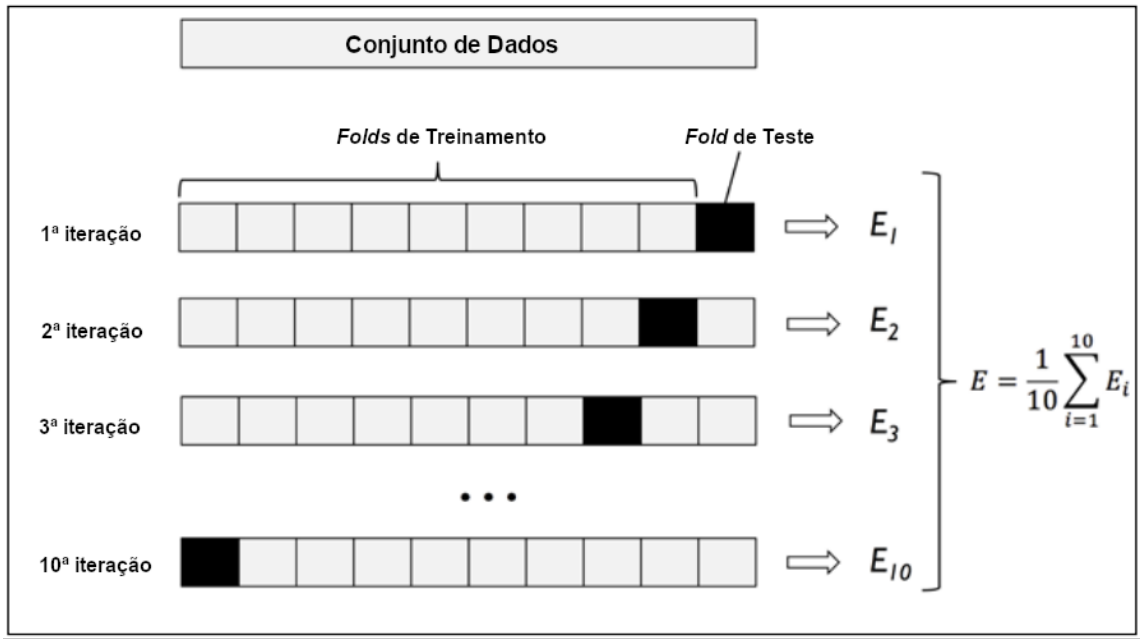

Figura 8 – Método de validação cruzada por *k-fold*, com k = 10. Adaptado de (RASCHKA; MIRJALILI, 2017).

dados existentes em determinado conjunto de dados. Como cada amostra de dados faz parte de um *fold* e cada *fold* em algum momento será considerado como um conjunto de teste. Inevitavelmente, todas as amostras do conjunto de dados serão utilizadas para avaliar o desempenho de um modelo. Consequentemente, para obter um bom desempenho médio de generalização é preciso obter um bom desempenho individual em cada modelo treinado durante a realização da validação cruzada (MÜLLER; GUIDO, 2016).

A validação cruzada também oferece como vantagem a possibilidade de obter algumas informações sobre a sensibilidade do modelo em relação ao conjunto de dados, pois é possível ter uma ideia sobre qual será o pior e o melhor desempenho de generalização do modelo quando ele for utilizado para realizar predições de dados ainda desconhecidos. Müller e Guido (2016) afirmam que a principal desvantagem da validação cruzada está relacionada ao aumento do custo computacional. Como é preciso realizar o treinamento de k modelos e cada modelo precisa ser avaliado com uma divisão dos dados. Naturalmente, a utilização da validação cruzada pode ter um custo computacional até k vezes maior do que utilização do método *holdout* que realiza uma única divisão nos dados.

Validação cruzada estratificada por *k-fold*

O método de validação cruzada por *k-fold* pode apresentar uma limitação ao ser utilizado para avaliar o desempenho de classificadores treinados com conjuntos de dados desbalanceados. O método de validação cruzada estratificada por *k-fold* possui uma ligeira melhoria capaz de contornar essa limitação. Ao realizar a validação cruzada estratificada, as proporções das amostras pertencentes a cada classe do conjunto de dados inicial são preservadas em cada *fold* gerado. Isso faz com que cada *fold* seja suficientemente repre-

sentativo das proporções de amostras pertencentes a cada classe do conjunto de dados de treinamento (RASCHKA; MIRJALILI, 2017). Por esse motivo, Müller e Guido (2016) recomendam utilizar o método de validação cruzada estratificada por *k-fold* para avaliar o desempenho de generalização de classificadores, visto que esse método fornece uma estimativa de desempenho mais precisa para classificadores treinados com conjuntos de dados desbalanceados.

3.3.2 Métricas de Avaliação de Desempenho de Classificadores

Na subseção anterior foram discutidos alguns métodos que podem ser utilizados para obter uma estimativa do desempenho de generalização de modelos de aprendizado de máquina supervisionado. A discussão foi limitada apenas em apresentar o funcionamento desses métodos que normalmente são conhecidos como métodos de amostragem. Porém, não foram discutidas as possíveis métricas de desempenho que são utilizadas em conjunto com esses métodos. Diante disso, esta subseção apresenta algumas métricas de desempenho que podem ser utilizadas para avaliar classificadores.

Métricas de avaliação de classificadores binários

Uma das formas mais utilizadas para representar o resultado da avaliação de um classificador binário é por meio da utilização de uma matriz de confusão (MÜLLER; GUIDO, 2016). Como o processo de treinamento de um classificador binário utiliza um conjunto de dados que possui apenas duas classes. A matriz de confusão para esse tipo de classificador pode ser construída apenas contando o número de vezes que as amostras de dados de uma determinada classe são classificadas como pentencentes a uma ou a outra classe existente no conjunto (GÉRON, 2019).

		Classe Predita	
		+	**-**
Classe Verdadeira	**+**	Verdadeiro Positivo **(VP)**	Falso Negativo **(FN)**
	-	Falso Positivo **(FP)**	Verdadeiro Negativo **(VN)**

Figura 9 – Matriz de confusão utilizada para avaliar o desempenho de classificadores binários. Adaptado de (RASCHKA; MIRJALILI, 2017).

A Figura 9 apresenta a estrutura de uma matriz de confusão construída para a

avaliação de desempenho de um classificador binário. Foi construída uma matriz quadrada, onde as linhas correspondem às classes verdadeiras das amostras de dados e as colunas correspondem às classes que foram preditas pelo classificador. Cada posição existente na matriz possui um valor numérico que representa a contagem de quantas vezes uma amostra de dados que pertence à classe correspondente à linha foram classificadas como se pertencessem à classe corresponde à coluna (MÜLLER; GUIDO, 2016). Como existem apenas duas classes normalmente é adotada a convenção de nomear essas classes como sendo uma classe positiva e a outra negativa.

Conforme pode ser visualizado na Figura 9, os valores que pertencem à diagonal principal da matriz de confusão correspondem a quantas amostras foram classificadas de maneira correta pelo classificador binário (RASCHKA; MIRJALILI, 2017). Os valores que estão fora da diagonal principal correspondem a quantas amostras de uma determinada classe foram erroneamente classificadas como se pertencessem a outra classe. Na Figura 9 esses valores foram representados conforme descrito a seguir.

- **VP:** quantidade de amostras pertencentes a classe positiva que foram corretamente classificadas como sendo da classe positiva.
- **FN:** quantidade de amostras pertencentes a classe positiva que foram erroneamente classificadas como sendo da classe negativa.
- **FP:** quantidade de amostras pertencentes a classe negativa que foram erroneamente classificadas como sendo da classe positiva.
- **VN:** quantidade de amostras pertencentes a classe negativa que foram corretamente classificadas como sendo da classe negativa.

Embora a matriz de confusão seja útil para identificar informações precisas a respeito do desempenho de um classificador binário, a inspeção de uma matriz de confusão é um processo manual e qualitativo que pode ser trabalhoso (MÜLLER; GUIDO, 2016). Diante disso, diversas métricas que permitem avaliar o desempenho de generalização de um classificador binário por meio da utilização das informações contidas na matriz de confusão do classificador foram propostas na literatura. A seguir, algumas das métricas mais utilizadas são apresentadas.

A acurácia é uma métrica de avaliação de desempenho de classificadores que permite medir a proporção existente entre o total de predições realizadas corretamente e a quantidade total de amostras de dados utilizadas durante a avaliação (ASTHANA; HAZELA, 2019). Albon (2018) afirma que a métrica acurácia é muito comum, e possui como ponto forte o fato de possuir uma explicação intuitiva e simples. Porém, o autor destaca que essa métrica possui uma limitação: nem sempre um valor de acurácia reflete a eficácia de um classificador que foi treinado utilizando um conjunto de dados desbalanceado.

Essencialmente, resultados de tal métrica podem fornecer uma estimativa inadequada ao serem utilizados para avaliar um classificador que foi treinado com um conjunto de dados desbalanceado (MRABET et al., 2021). A Equação 3.4 define como é realizado o cálculo de tal métrica.

$$Acurácia = \frac{VP + VN}{VP + FP + VN + FN} \tag{3.4}$$

Precisão (do termo em inglês, *precision*) é uma métrica de avaliação de classificadores que permite medir a proporção das amostras de dados preditas como positivas que realmente pertencem a classe positiva (ALBON, 2018). Por isso, precisão é uma métrica de desempenho muito utilizada em domínios de aplicação que possuem a necessidade de limitar o número de falsos positivos gerados pelo classificador (MÜLLER; GUIDO, 2016). A Equação 3.5 define como é realizado o cálculo da precisão de um classificador.

$$Precisão = \frac{VP}{VP + FP} \tag{3.5}$$

Revocação, do termo em inglês, *recall* é uma métrica de avaliação de classificadores que permite medir a proporção de amostras de dados que pertencem a classe positiva que foram corretamente classificadas como positivas pelo classificador (ALBON, 2018). Por isso, a revocação é uma métrica de desempenho muito utilizada em domínios de aplicação que possuem a necessidade de identificar todas as amostras de dados positivas, ou seja, domínios em que é importante evitar a geração de falsos negativos (MÜLLER; GUIDO, 2016). A Equação 3.6 define como é realizado o cálculo da revocação de um classificador.

$$Revocação = \frac{VP}{VP + FN} \tag{3.6}$$

Embora as métricas de desempenho, precisão e revocação sejam importantes e muito adotadas. A utilização de apenas uma dessas métricas pode não ser suficiente para avaliar adequadamente um classificador (MÜLLER; GUIDO, 2016). Existe uma outra métrica de avaliação denominada de *f-score* ou *f-measure* que possui uma variante específica conhecida como F_1, que pode ser utilizada como um meio termo entre a precisão e revocação (ALBON, 2018). Por levar em consideração os valores de precisão e revocação, a métrica F_1 oferece uma estimativa de desempenho mais precisa do que a métrica acurácia durante a avaliação de classificadores treinados com conjuntos de dados desbalanceados. A cálculo da métrica F_1 é realizado por meio de uma média harmônica entre os valores de precisão e revocação, conforme mostrado na Equação 3.7.

$$F_1 = 2 \times \frac{Precisão \times Revocação}{Precisão + Revocação} \tag{3.7}$$

Métricas de avaliação de classificadores multiclasse

As métricas de avaliação de classificadores multiclasse são derivadas das métricas utilizadas para os classificadores binários, porém, é necessário realizar o cálculo da média de todas as classes envolvidas na avaliação (MÜLLER; GUIDO, 2016). Por exemplo, a acurácia para classificadores multiclasse (Equação 3.8) é novamente definida como a fração das amostras de dados classificadas corretamente.

$$Acurácia^{\star} = \frac{\sum_{i=1}^{n} \frac{VP_i + VN_i}{VP_i + FP_i + VN_i + FN_i}}{n} \tag{3.8}$$

As Equações 3.9, 3.10 e 3.11, respectivamente, representam as versões multiclasse das métricas precisão, revocação e F_1. Müller e Guido (2016) afirmam que a métrica mais utilizada para avaliação de classificadores multiclasse treinados com conjunto de dados desbalanceados é a F_1. Porém, os autores ressaltam que normalmente os resultados das métricas de avaliação de classificadores multiclasse são mais difíceis de compreender do que os resultados das métricas de classificadores binários.

$$Precisão^{\star} = \frac{\sum_{i=1}^{n} \frac{VP_i}{VP_i + FP_i}}{n} \tag{3.9}$$

$$Revocação^{\star} = \frac{\sum_{i=1}^{n} \frac{VP_i}{VP_i + FN_i}}{n} \tag{3.10}$$

$$F_1^{\star} = 2 \times \frac{Precisão^{\star} \times Revocação^{\star}}{Precisão^{\star} + Revocação^{\star}} \tag{3.11}$$

3.4 Bibliotecas e *Frameworks* de Aprendizado de Máquina

O aprendizado de máquina é campo de estudo que está em constante evolução (MRABET et al., 2021). Consequentemente, o número de algoritmos de aprendizado de máquina existentes, bem como suas diferentes implementações, é consideravelmente grande (NGUYEN et al., 2019). Os algoritmos de aprendizado de máquina são frequentemente propostos em formatos de pseudocódigo que incluem fórmulas científicas, regras e conceitos algorítmicos. Esses algoritmos também podem apresentar cálculos numéricos sofisticados que podem ser difíceis de implementar em arquiteturas de hardware modernas que oferecem poder computacional de alto desempenho. Esses fatores somados, resultam em dificuldades para que os desenvolvedores consigam entender e implementar em arquiteturas modernas, as fórmulas, as regras e os conceitos presentes nos algoritmos que são disponibilizados em textos científicos (BRAIEK; KHOMH, 2020).

Devido a essas dificuldades, os desenvolvedores geralmente dependem da utilização de bibliotecas e *frameworks* de terceiros para implementar os seus programas de aprendizado de máquina. Bibliotecas e *frameworks* de aprendizado de máquina fornecem implementações otimizadas de diferentes algoritmos que são compatíveis com arquiteturas de hardware modernas (BRAIEK; KHOMH, 2020). Essas ferramentas são concebidas para apoiar o desenvolvimento de diversos tipos de aplicações, como, plataformas analíticas, sistemas preditivos, sistemas de recomendação, reconhecimento de imagens e processamento de linguagem natural. Possuindo como objetivo em comum, a disponibilização de um ambiente integrado, construído sobre linguagens de programação de propósito geral, que facilita a utilização de diferentes algoritmos do estado da arte de aprendizado de máquina (NGUYEN et al., 2019). A seguir será discutido sobre o *framework* de aprendizado adotado neste trabalho: o `scikit-learn`.

3.4.1 O *Framework* de Aprendizado de Máquina `scikit-learn`

O `scikit-learn` é um projeto de software de código aberto que possui como principal objetivo tornar o aprendizado de máquina acessível a todos, seja na academia ou na indústria. O `scikit-learn` nasceu da observação de que a maioria dos algoritmos de aprendizado de máquina não eram acessíveis aos profissionais que mais poderiam se beneficiar deles, por exemplo, biólogos, cientistas climáticos, físicos experimentais e desenvolvedores de aplicações (VAROQUAUX et al., 2015).

O `scikit-learn` tira proveito do rico ambiente oferecido pela linguagem de programação Python que possui uma natureza interativa de alto nível e apresenta um ecossistema maduro com diversas bibliotecas científicas para fornecer implementações do estado da arte de muitos algoritmos de aprendizado de máquina conhecidos (PEDREGOSA et al., 2011). O projeto possui a qualidade e a facilidade de utilização como pilares o que implica no emprego de esforços dos colaboradores em questões de instalação, documentação e projeto de API (BUITINCK et al., 2013). Resultando em um *framework* de aprendizado de máquina que pode ser utilizado em ambientes do mundo real por meio de uma interface fácil de usar e totalmente integrada à linguagem Python (VAROQUAUX et al., 2015).

O `scikit-learn` disponibiliza implementações de diversos algoritmos de aprendizado de máquina conhecidos. A seguir, é apresentada uma listagem que descreve brevemente alguns tipos de algoritmos que são disponibilizados em determinados módulos do projeto:

- O módulo *`sklearn.linear_model`* implementa uma variedade de algoritmos para o treinamento de modelos lineares.
- O módulo *`sklearn.naive_bayes`* implementa algoritmos de aprendizado supervisionado baseados no algoritmo Naive Bayes.

- O módulo ***sklearn.neighbors*** implementa algoritmos de aprendizado supervisionado baseados no algoritmo K-NN.
- O módulo ***sklearn.gaussian_process*** implementa algoritmos de aprendizado supervisionado baseados em Processos Gaussianos.
- O módulo ***sklearn.svm*** implementa algoritmos de aprendizado supervisionado baseados em Máquina de Vetores de Suporte, do termo em inglês, *Support Vector Machine*.
- O módulo ***sklearn.tree*** implementa algoritmos de aprendizado supervisionado para o treinamento de modelos baseados em árvores de decisão.
- O módulo ***sklearn.neural_network*** implementa algoritmos para o treinamento de modelos baseados em redes neurais artificiais.

Para facilitar a utilização dos objetos existentes no `scikit-learn`, os colaboradores do projeto tomaram a decisão de viabilizar a utilização desses objetos por meio de um conjunto de interfaces públicas que foram definidas (PEDREGOSA et al., 2011). Todos os objetos existentes no `scikit-learn` compartilham uma API básica comum que consiste em três interfaces complementares: a interface ***estimator*** que deve ser utilizada para construir estimadores e para realizar o treinamento de modelos, a interface ***predictor*** que deve ser utilizada para realizar predições e a interface ***transformer*** que deve ser utilizada para realizar conversões de dados (BUITINCK et al., 2013).

A interface ***estimator*** é o componente central do `scikit-learn`. Ela define os mecanismos necessários para criação dos objetos que representam os algoritmos de aprendizado de máquina disponibilizados no *framework*. Além disso, ela também fornece um método específico para realização do treinamento dos modelos. A inicialização de um ***estimator*** e o treinamento de um modelo são realizados em dois passos distintos. Um ***estimator*** é inicializado com um conjunto de valores de hiperparâmetros que são fornecidos como entrada para o construtor do ***estimator***. Diante disso, um ***estimator*** pode ser considerado como uma função que mapeia os valores de hiperparâmetros para o algoritmo de aprendizado de máquina que será utilizado. O construtor do ***estimator*** não recebe nenhum dado de treinamento, e consequentemente, não realiza o treinamento do modelo. Tudo que o construtor faz é atribuir os valores recebidos ao objeto que representa o ***estimator*** utilizado (BUITINCK et al., 2013).

O treinamento do modelo é realizado utilizando o método ***fit*** do objeto que representa o ***estimator*** utilizado. A responsabilidade do método ***fit*** é realizar o treinamento de um modelo executando o algoritmo de aprendizado com os dados de treinamento que devem ser fornecidos durante a invocação do próprio método. Diante disso, o método ***fit*** pode ser considerado como uma função que mapeia os dados de treinamento em um

modelo de aprendizado de máquina. Após realizar o treinamento, o método ***fit*** sempre retorna o objeto ***estimator*** no qual foi invocado. Com isso, o objeto ***estimator*** também pode ser considerado como o modelo de aprendizado de máquina que acabou de ser treinado. Portanto, pode ser utilizado para realizar predições (BUITINCK et al., 2013).

Além dos algoritmos disponibilizados para treinar diferentes modelos de aprendizado de máquina, o `scikit-learn` também oferece diversas funcionalidades para apoiar a realização das outras atividades envolvidas no processo de treinamento de um modelo. Por exemplo, o módulo ***sklearn.metrics*** inclui diversas funções que implementam diferentes métricas para realização da avaliação de desempenho de modelos de aprendizado de máquina. Por exemplo, para realização da avaliação de classificadores, são disponibilizadas implementações para o cálculo de acurácia, precisão, revocação e F_1.

3.5 Considerações Finais

Neste capítulo foi realizada uma discussão sobre o aprendizado de máquina. Este capítulo foi escrito com o objetivo de realizar uma breve introdução ao aprendizado de máquina e apresentar os conceitos, as técnicas e os algoritmos que são relevantes para o contexto deste trabalho. Dessa forma, a discussão realizada ao longo do capítulo foi centrada na técnica de aprendizado de máquina supervisionado, com ênfase nos algoritmos de árvore de decisão e no algoritmo K-NN.

Após versar sobre esses dois algoritmos, foi apresentado o processo de aprendizado de máquina que foi introduzido por Amershi et al. (2019). Foi optado por abordar neste capítulo, o processo introduzido por Amershi et al., pelo fato dos autores conseguirem descrever de maneira clara e objetiva as diferentes etapas e atividades que podem constituir um processo de treinamento de modelos de aprendizado de máquina. Relacionado ao processo de treinamento de modelos, também foi discutido sobre diferentes métodos e métricas de avaliação de desempenho de classificadores. Especificamente, foi discutido sobre os seguintes métodos: *houldout*, validação cruzada por *k-fold* e validação cruzada estratificada por *k-fold*. Em relação às métricas de avaliação de desempenho, foram abordadas: acurácia, precisão, revocação e F_1. Essas quatro métricas de avaliação de desempenho e o método de validação cruzada estratificada por *k-fold* foram essenciais para a produção deste trabalho.

Adicionalmente, apresentou-se discussão relacionada à necessidade de utilizar bibliotecas ou *frameworks* de aprendizado de máquina para construção de modelos de aprendizado de máquina. Por fim, o *framework* de aprendizado de máquina `scikit-learn`, que foi amplamente utilizado no desenvolvimento deste trabalho, foi descrito brevemente. Dessa forma, foram apresentadas os algoritmos, métodos, métricas e ferramentas de aprendizado de máquina que são necessárias para o entendimento da proposta deste trabalho.

4 Teste de Sistemas Baseados em Aprendizado de Máquina

Riccio et al. (2020) afirmaram que atualmente é muito comum integrar um ou mais componentes de aprendizado de máquina em um sistema de software. Diante disso, os autores definiram o termo *sistema baseado em aprendizado* para denotar um tipo de sistema de software que possui pelo menos um componente que depende da utilização de técnicas de aprendizado de máquina. De maneira semelhante, os autores utilizaram o termo *sistemas de software tradicionais*, ou simplesmente *sistemas tradicionais* para fazer referência aos sistemas que não utilizam técnicas de aprendizado de máquina.

Os sistemas baseados em aprendizado de máquina ampliaram as dificuldades e introduziram novos desafios para realização da atividade de teste (RICCIO et al., 2020). Enquanto, os sistemas tradicionais são pré-programados para executar um conjunto específico de regras, os sistemas baseados em aprendizado de máquina possuem um comportamento probabilístico e não determinístico (MARIJAN et al., 2019). Como afirmado por Marijan et al. (2019), um sistema baseado em aprendizado de máquina é capaz de produzir saídas diferentes para execuções realizadas com as mesmas entradas e pré-condições.

Pesquisadores em engenharia de software observaram que as abordagens utilizadas no teste de sistemas tradicionais, como as discutidas no Capítulo 2, geralmente falham e não podem ser diretamente aplicadas no teste de sistemas baseados em aprendizado de máquina (MARIJAN et al., 2019). Conforme ressaltado por Zhang et al. (2022), é necessário adaptar as abordagens e técnicas consolidadas no teste de sistemas tradicionais para o contexto dos sistemas baseados em aprendizado de máquina. Dessa forma, neste capítulo, apresenta-se uma discussão sobre como pesquisadores vêm adaptando técnicas existentes ou desenvolvendo novas abordagens para conduzir o teste de sistemas baseados em aprendizado de máquina.

O restante deste capítulo está organizado conforme descrito a seguir. Na Seção 4.1 é realizada uma introdução ao teste de sistemas baseados em aprendizado de máquina. Os processos de teste online e offline são discutidos na Seção 4.2. Os principais componentes de teste são discutidos na Seção 4.3. Na Seção 4.4 são abordados os níveis de teste que são específicos desse tipo de sistema. A Seção 4.5 apresenta as propriedades de teste. Os principais problemas e desafios de teste são discutidos na Seção 4.6. Na Seção 4.7 é discutido brevemente sobre as técnicas que estão sendo adaptadas e aplicadas no teste de sistemas baseados em aprendizado de máquina. Na Seção 4.8 é discutido sobre dois critérios de teste específicos para sistemas baseados em aprendizado de máquina. Por fim,

na Seção 4.9 são apresentadas as considerações finais do capítulo.

4.1 Introdução ao Teste de Sistemas Baseados em Aprendizado de Máquina

Antes de aprofundar a discussão sobre o teste de sistemas baseados em aprendizado de máquina, é preciso discorrer sobre as diferentes interpretações que a palavra "teste" possui no contexto de aprendizado de máquina. O conceito de testar sistemas baseados em aprendizado de máquina é frequentemente discutido em duas comunidades científicas distintas, a saber, a comunidade de aprendizado de máquina e a comunidade de teste de software (MARIJAN et al., 2019). Cada uma dessas comunidades fornece uma interpretação específica para o teste de um sistema baseado em aprendizado de máquina. Diante disso, será explicado brevemente as diferenças que podem existir nessas interpretações.

No contexto da comunidade de aprendizado de máquina, o teste de um modelo é normalmente realizado durante o processo de treinamento do modelo. Momento no qual, dados de teste (distintos dos dados de treinamento) são submetidos ao modelo e, em seguida, os resultados obtidos são comparados com os resultados esperados (que normalmente são previamente conhecidos) (MARIJAN et al., 2019). Do ponto de vista da comunidade de aprendizado de máquina, a atividade de teste de um modelo é realizada utilizando os métodos de avaliação de desempenho que foram discutidos na Seção 3.3.1.

Por outro lado, na comunidade de teste de software, testar um sistema baseado em aprendizado de máquina implica na verificação do comportamento do sistema para quaisquer dados de teste, incluindo aqueles para os quais os resultados esperados são desconhecidos (MARIJAN et al., 2019). Além disso, durante a realização da atividade de teste, a comunidade de teste de software sugere que os componentes de aprendizado de máquina sejam testados considerando uma variedade de atributos de qualidade como, por exemplo, corretude, robustez e segurança. Este capítulo, assim como a proposta principal deste trabalho, faz uso desta última interpretação de teste, que é empregada pela comunidade de teste de software.

Zhang et al. (2022) definiram o teste de sistemas baseados em aprendizado de máquina como a condução de qualquer atividade cujo principal objetivo é revelar defeitos em componentes de aprendizado de máquina. A Figura 10 apresenta um visão de alto nível do ciclo de vida de desenvolvimento, implantação e testes de um modelo de aprendizado de máquina. O ciclo de vida inicia com a utilização dos dados pré-existentes para realização do treinamento de um modelo protótipo. Com o objetivo de verificar se o modelo atende aos pré-requisitos previamente estabelecidos, após a geração do modelo protótipo, deve ser realizado o teste offline. Ao concluir o teste offline, deve ser realizada a implantação do modelo. A implantação consiste em disponibilizar o sistema que engloba o modelo para

utilização dos usuários finais. Após a implantação, os usuários começam a solicitar que o modelo realize predições. As predições realizadas pelo modelo geram novos dados que podem ser analisados por meio da realização do teste online do modelo (ZHANG et al., 2022).

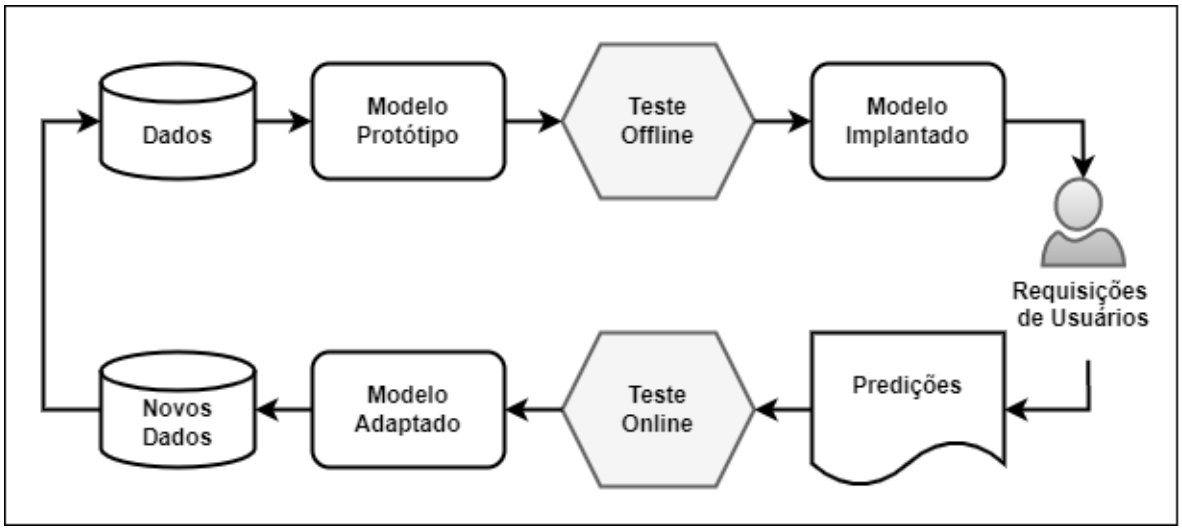

Figura 10 – Visão de alto nível do ciclo de vida de desenvolvimento, implantação e testes de um modelo de aprendizado de máquina. Retirado de (ZHANG et al., 2022).

Como mostrado na Figura 10, o processo de teste de um modelo de aprendizado de máquina pode ser dividido em dois processos distintos: *(i)* o processo de teste offline e *(ii)* o processo de teste online. O teste offline envolve atividades de teste que são realizadas antes do modelo ser implantado e disponibilizado para utilização dos usuários finais. Normalmente, para a condução das atividades de teste offline são utilizados os métodos de avaliação de desempenho que foram discutidos na Seção 3.3.1. Como o teste offline pode depender de dados de teste pré-existentes, os dados de teste disponíveis podem ser insuficientes para representar adequadamente os dados futuros, aos quais o modelo poderá ser exposto. Com isso, após a implantação e disponibilização do modelo para os usuários, é recomendado a realização do teste online. O teste online permite analisar os novos dados que são gerados pelo modelo em teste com o objetivo de identificar como o modelo responde aos diferentes comportamentos dos usuários (ZHANG et al., 2022).

De acordo com Zhang et al. (2022), a literatura de teste de sistemas baseados em aprendizado de máquina é organizada em termos de quatro aspectos: *(i)* os processos de teste; *(ii)* os componentes de teste; *(iii)* as propriedades de teste; e *(iv)* os diferentes domínios de aplicação. Os processos, os componentes e as propriedades de teste serão discutidos nas Subseções 4.2, 4.3 e 4.5, respectivamente.

4.2 Processos de Teste de Sistemas Baseados em Aprendizado de Máquina

Um processo de teste guia a realização do teste de sistemas baseados em aprendizado de máquina: especificando quais atividades de teste devem ou podem ser realizadas (ZHANG et al., 2022). A Figura 11 mostra um processo de teste idealizado proposto por Zhang et al. (2022). Como pode ser observado, o processo proposto pelos autores é dividido em dois processos distintos: *(i)* um que especifica as atividades que constituem o processo de teste offline e *(ii)* o outro que especifica as atividades que integram o teste online.

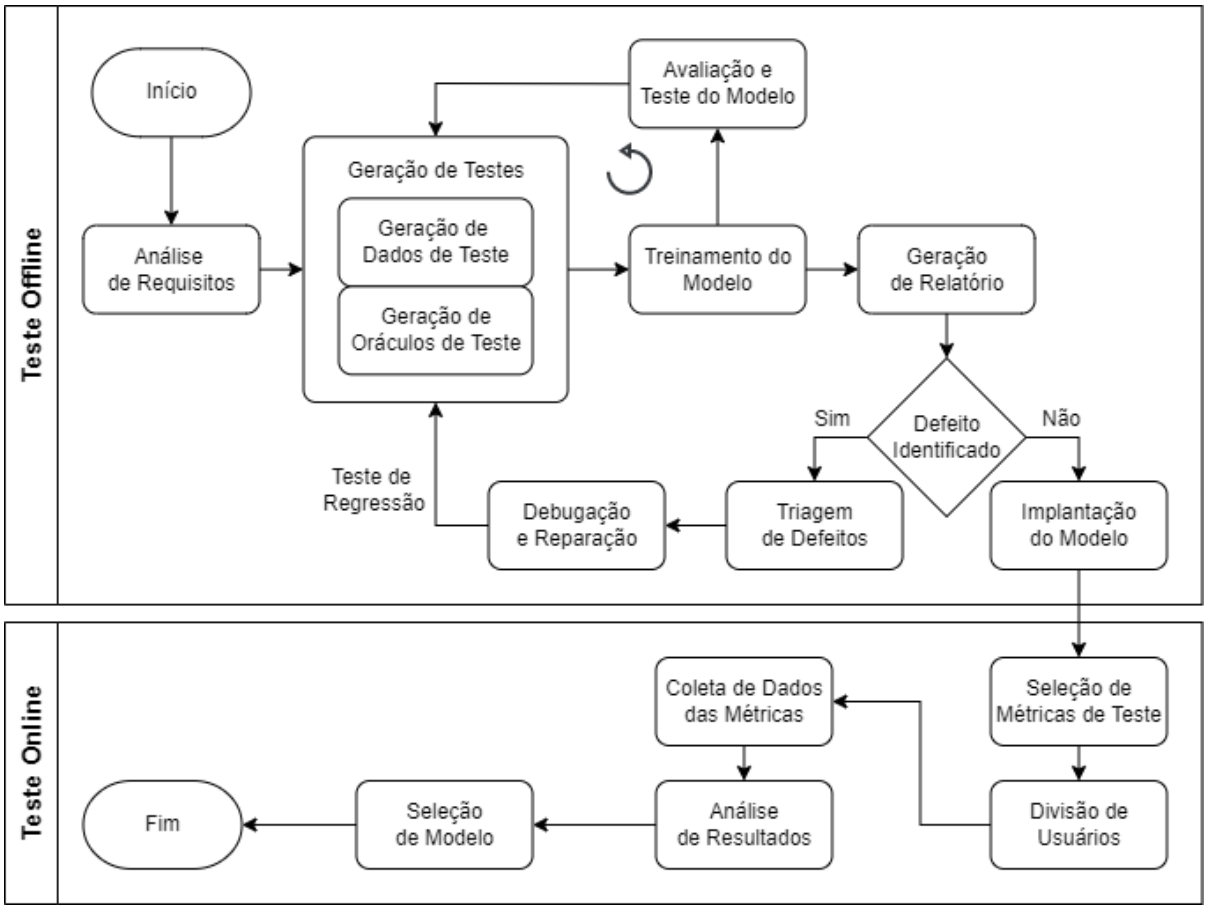

Figura 11 – Processo idealizado para o teste de sistemas baseados em aprendizado de máquina. Adaptado de (ZHANG et al., 2022).

4.2.1 Processo de Teste Offline

As atividades de teste que constituem o processo de teste offline são apresentadas na parte superior da Figura 11. O teste offline inicia por meio da análise de requisitos do sistema baseado em aprendizado de máquina. As informações da especificação de requisitos devem ser utilizadas no planejamento do procedimento de teste do sistema. Em seguida, os dados de teste do sistema baseado em aprendizado de máquina devem ser

gerados de acordo com um propósito específico ou selecionados do conjunto de dados pré-existentes. Os oráculos de teste também precisam ser identificados ou gerados de alguma forma. Como será discutido com mais detalhes na Seção 4.6, a identificação ou geração de dados e oráculos de teste constituem dois dos maiores desafios encontrados durante a realização do teste de sistemas baseados em aprendizado de máquina. Com a obtenção dos dados e oráculos de teste, o conjunto de casos de teste deve ser construído e executado contra o sistema baseado em aprendizado de máquina em teste (ZHANG et al., 2022).

Zhang et al. (2022) afirmam que a execução dos testes consiste na execução do conjunto de casos de teste contra o modelo e na verificação de possíveis violações dos resultados esperados. Após a execução dos testes, métricas de avaliação podem ser empregadas para examinar o desempenho do modelo existente no sistema em teste. Idealmente, após a execução dos testes, deve ser produzido um relatório de defeitos para possibilitar que os desenvolvedores e testadores consigam reproduzir e corrigir os defeitos encontrados. Depois de aplicar as devidas correções, devem ser realizados os testes de regressão para garantir que as correções realizadas não introduziram novos defeitos. Por fim, quando nenhum defeito for encontrado, o processo de teste offline termina e o modelo pode ser implantado e disponibilizado para utilização.

4.2.2 Processo de Teste Online

Na parte inferior da Figura 11 é mostrado o processo de teste online. De acordo com Zhang et al. (2022), o teste online pode ser realizado por meio de diferentes abordagens. O monitoramento contínuo da execução do sistema baseado em aprendizado de máquina, com o objetivo de verificar se o sistema continua atendendo os requisitos previamente definidos, constituiu um método comum de teste online. Outra abordagem normalmente utilizada consiste na realização de testes do tipo A/B, que comparam se um novo modelo é superior a um modelo antigo em um contexto de aplicação específico. No caso de realização de testes A/B, os usuários do sistema são divididos em dois grupos distintos. Um grupo deve enviar requisições para o modelo antigo e o outro grupo, para o modelo novo. Em tal cenário, as respostas geradas por ambos os modelos devem ser verificadas com o objetivo de identificar qual modelo apresenta o melhor desempenho.

4.3 Componentes de Teste de Sistemas Baseados em Aprendizado de Máquina

De acordo com Zhang et al. (2022), o desenvolvimento de sistemas baseados em aprendizado de máquina envolve a interação com diferentes componentes, e esses componentes indicam ao testador o que deve ser testado. Especificamente, os autores destacam

três componentes que devem ser testados: os dados, o programa de treinamento[1] e o *framework* de aprendizado de máquina. De forma semelhante, Braiek e Khomh (2020) também destacaram o modelo de aprendizado de máquina como um dos principais componentes que devem ser testados nesse tipo de sistema.

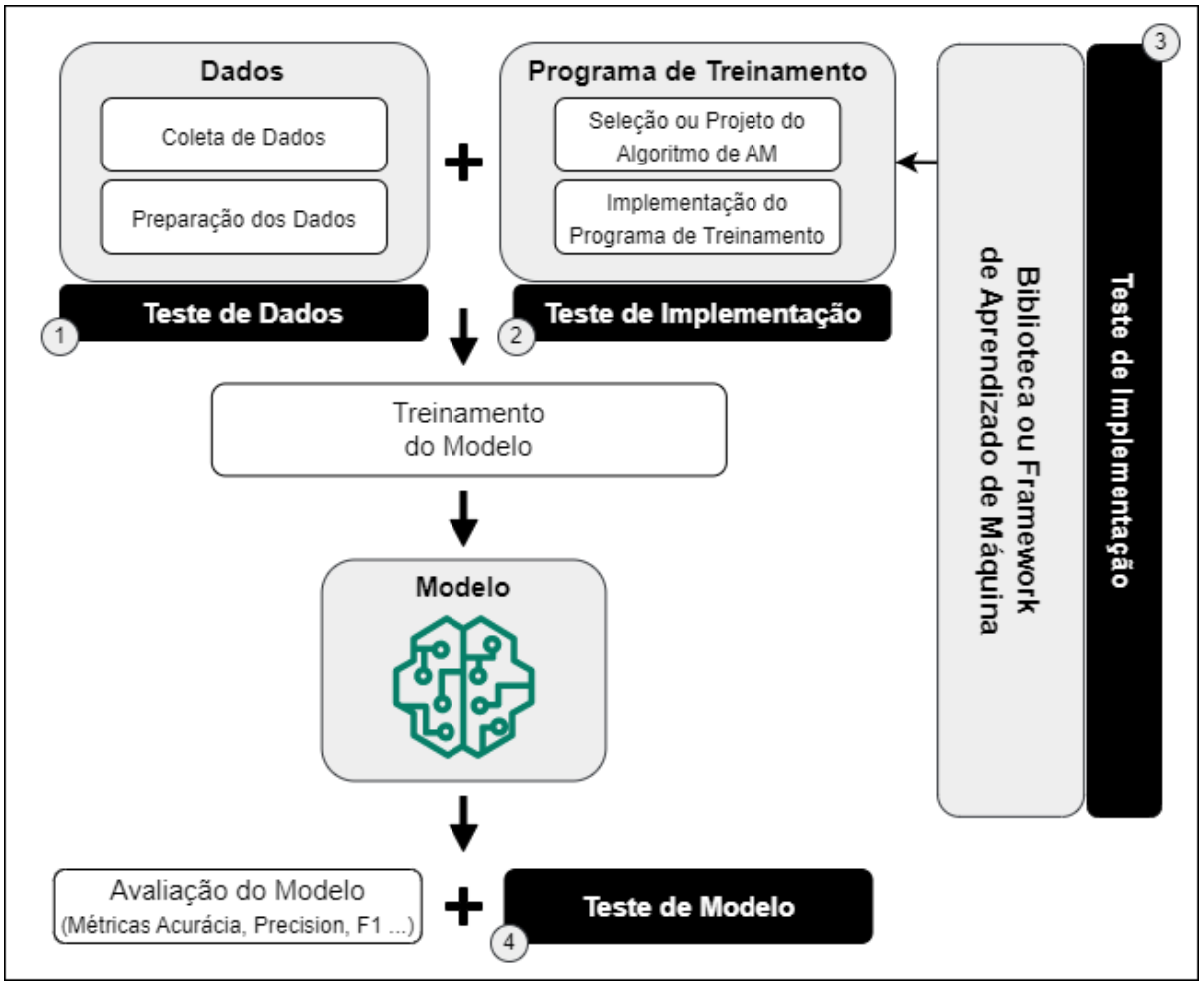

Figura 12 – Componentes (mostrados nos retângulos de cor cinza) envolvidos na construção de um modelo de aprendizado de máquina e os respectivos tipos de teste (mostrados nos retângulos de cor preta). Adaptado de (BRAIEK; KHOMH, 2020) e (ZHANG et al., 2022).

Os componentes mencionados anteriormente possuem a característica de serem fortemente acoplados: o que faz com que a propagação de erros seja um problema ainda mais difícil de ser tratado em sistemas baseados em aprendizado de máquina do que em sistemas tradicionais (AMERSHI et al., 2019). Portanto, enfatiza-se a necessidade de testar cada componente isoladamente (ZHANG et al., 2022). A Figura 12 apresenta os três componentes de teste de um sistema baseado em aprendizado de máquina que foram mencionados por Zhang et al. (2022) e o componente modelo que foi descrito por Braiek e Khomh (2020). Cada componente é apresentado nos retângulos de cor cinza e o tipo de

[1] Zhang et al. (2022) utilizaram o termo "programa de aprendizado" para fazer referência ao programa de treinamento. Porém, neste trabalho, foi optado por utilizar o termo "programa de treinamento", que foi utilizado por Braiek e Khomh (2020).

teste que deve ser realizado é apresentado nos retângulos de cor preta. Conforme pode ser observado, pode ser necessário realizar o teste de dados, teste de modelo, teste de implementação do programa de treinamento e o teste de implementação do *framework* de aprendizado de máquina utilizado. As próximas subseções listam os motivos relacionados à condução de atividades de teste envolvendo os componentes mencionados anteriormente.

4.3.1 Teste dos Dados

A existência de defeitos no conjunto de dados de treinamento pode afetar a qualidade do modelo construído (AMERSHI et al., 2019). Como o comportamento de um sistema baseado em aprendizado de máquina depende dos dados utilizados no processo de treinamento do modelo, um defeito não detectado nos dados pode contribuir para a geração de problemas durante a utilização do sistema (BRECK et al., 2019). De acordo com Zhang et al. (2022), o teste de dados pode ser realizado por vários motivos como, por exemplo, verificar se os dados disponíveis são suficientes para treinar ou testar o modelo, verificar se os dados do conjunto de treinamento e do conjunto de teste apresentam problemas, e certificar que os dados não apresentam ruídos capazes de comprometer o desempenho do modelo.

4.3.2 Teste do Programa de Treinamento

De acordo com Zhang et al. (2022), um programa de treinamento pode ser classificado em duas partes: *(i)* a parte conceitual que representa o algoritmo de aprendizado de máquina que o desenvolvedor desenvolve ou decide utilizar a partir de um *framework* de aprendizado de máquina; e *(ii)* a implementação, ou seja, o código que o desenvolvedor escreve para implementar o algoritmo ou para configurar o algoritmo que ele decidiu utilizar de um *framework*. Defeitos no programa de treinamento podem surgir de ambas as partes (ZHANG et al., 2022). Especificamente, o desenvolvedor pode realizar o projeto de um algoritmo de aprendizado de máquina de maneira inadequada ou pode escolher um algoritmo não adequado para o problema que ele precisa resolver. De maneira semelhante, o desenvolvedor pode projetar ou escolher um algoritmo correto, porém, pode inserir defeitos no código que implementa o algoritmo ou que configura o algoritmo utilizado do *framework*.

4.3.3 Teste do *Framework* de Aprendizado de Máquina

O teste do *framework* de aprendizado de máquina tem o propósito de verificar se as implementações dos algoritmos disponibilizados possuem defeitos. Naturalmente, conforme ressaltado por Zhang et al. (2022), a existência de defeitos no *framework* pode fazer com que um sistema final, que possua modelos que foram treinados com os algoritmos disponibilizados no *framework*, apresente falhas durante a sua utilização.

4.3.4 Teste do Modelo de Aprendizado de Máquina

O teste de um modelo de aprendizado de máquina pode ser realizado para verificar diferentes atributos de qualidade do modelo em teste (ZHANG et al., 2022): a corretude (capacidade do modelo de produzir respostas corretas) é um desses atributos. Um tipo de defeito conceitual que pode ser introduzido por desenvolvedores é aquele relacionado à escolha equivocada de valores de hiperparâmetros durante o processo de treinamento do modelo (BRAIEK; KHOMH, 2020). Conforme afirmado por Zhang et al. (2022), a escolha equivocada de valores de hiperparâmetros geralmente resulta em modelos com baixo desempenho. O teste de modelo pode ser realizado para, por exemplo, viabilizar a identificação de defeitos dessa natureza.

4.4 Níveis de Teste de Sistemas Baseados em Aprendizado de Máquina

Riccio et al. (2020) afirmam que os níveis de teste empregados no teste de sistemas tradicionais (que foram discutidos na Seção 2.2) também são aplicáveis ao teste de sistemas baseados em aprendizado de máquina. Todavia, os autores ressaltam que é necessário adaptar a definição conhecida de alguns níveis de teste ao contexto de aprendizado de máquina. Por exemplo, os autores definem o teste de integração como os testes que possuem como objetivo avaliar as interações existentes entre os diferentes componentes que constituem um sistema baseado em aprendizado de máquina, incluindo, as interações que podem ocorrer entre diferentes modelos. Na visão dos autores, o teste de integração busca revelar defeitos que surgem com a integração de modelos de aprendizado de máquina aos outros componentes do sistema.

4.4.1 Teste de Modelo

Dada a necessidade prática de testar modelos de aprendizado de máquina isoladamente e considerando o papel fundamental que um modelo possui dentro de um sistema baseado em aprendizado de máquina, Riccio et al. (2020) introduziram um novo nível de teste chamado de teste de modelo:

- **Teste de Modelo (*Model Testing*):** os testes de modelo consideram o modelo de aprendizado de máquina isoladamente, ou seja, durante a realização da atividade de teste de nível de modelo, nenhum outro componente do sistema baseado em aprendizado de máquina deve ser considerado (RICCIO et al., 2020).

O teste de nível de modelo tem o propósito de determinar se o modelo sendo testado apresenta um desempenho diferente do esperado. Normalmente, são identificados

alguns dados de entrada para os quais o modelo realiza uma predição errada. Em tal contexto, por meio da utilização de métricas de avaliação de desempenho, os testadores são capazes de obter uma estimativa do desempenho que o modelo irá apresentar quando for disponibilizado para os usuários finais. Riccio et al. (2020) afirmam que o teste de modelo pode ser considerado equivalente ao teste de nível de unidade: modelos podem ser vistos como unidades que dependem da realização de um processo de treinamento.

4.4.2 Teste de Entrada

Para construir um modelo é necessário realizar um processo de treinamento fornecendo um conjunto de dados como entrada para um algoritmo de aprendizado de máquina. Os dados exercem um papel fundamental no processo de treinamento do modelo. Uma vez que as regras que o modelo aprende são inferidas por meio das características existentes nos dados fornecidos. Riccio et al. (2020) também introduziram um nível de teste específico para o teste de dados denominado de teste de entrada:

- **Teste de Entrada (*Input Testing*):** o teste de entrada é realizado com o objetivo de analisar o conjunto de dados de treinamento que será utilizado para treinar o modelo de aprendizado de máquina. Além disso, o teste de entrada também pode ser realizado para avaliar os dados que serão submetidos ao modelo para predição após a disponibilização do sistema para utilização dos usuários finais (RICCIO et al., 2020).

O teste de nível de entrada possui o objetivo de analisar os dados de entrada que serão utilizados no processo de treinamento, buscando identificar fatores que podem resultar na criação de um modelo inadequado. Por exemplo, o conjunto de dados pode ser avaliado a fim de determinar se o mesmo encontra-se desbalanceado, o que pode resultar em um modelo com capacidade de generalização comprometida. Adicionalmente, esse tipo de teste pode ser utilizado para analisar os dados submetidos ao modelo pelos usuários finais com o objetivo de identificar se os novos dados estão devidamente representados pelos dados utilizados no processo de treinamento (RICCIO et al., 2020).

4.5 Propriedades de Teste de Sistemas Baseados em Aprendizado de Máquina

Propriedades de teste informam desenvolvedores e testadores em relação ao que deve ser testado em um sistema baseado em aprendizado de máquina (ZHANG et al., 2022). Especificamente, propriedades de teste indicam quais condições de um componente precisam ser garantidas por meio da realização das atividades de teste. É importante

ressaltar que as propriedades descritas nessa seção não são as propriedades utilizadas na realização do teste baseado em propriedades de sistemas tradicionais que foram discutidas na Seção 2.6. Nesta seção são discutidas algumas propriedades fundamentais dos sistemas baseados em aprendizado de máquina que foram apresentadas na literatura, também denominadas de atributos de qualidade (MARIJAN et al., 2019).

Zhang et al. (2022) classificam algumas das propriedades dos sistemas baseados em aprendizado de máquina como requisitos funcionais e requisitos não funcionais. Por exemplo, propriedades como a corretude e a relevância de um modelo de aprendizado de máquina são classificadas como requisitos funcionais. As propriedades eficiência, robustez, justiça e interpretabilidade são classificadas como não funcionais. De acordo com os autores, essas propriedades definem diferentes manifestações externas do comportamento de um sistema baseado em aprendizado de máquina e, portanto, precisam ser consideradas durante o teste desse tipo de sistema. No contexto da abordagem proposta, a propriedade mais relevante é *corretude*.

De acordo com Zhang et al. (2022) corretude refere-se à probabilidade com que um sistema baseado em aprendizado de máquina consegue acertar as predições que ele realiza. Adicionalmente, Zhang et al. ressaltam que possuir um nível de corretude aceitável é um requisito fundamental de qualquer sistema baseado em aprendizado de máquina. Para tal, o desempenho do modelo de aprendizado de máquina que o sistema utiliza deve ser avaliado utilizando amostras de dados que não foram utilizadas durante o treinamento do modelo, ou seja, devem ser utilizadas amostras de dados desconhecidas. Corretude pode ser mais formalmente descrita como a seguir.

- **Corretude (*Correctness*)**: Seja $\mathcal{D}$ a distribuição dos dados ainda desconhecidos, seja x uma amostra de dados pertencente a $\mathcal{D}$, h o modelo de aprendizado de máquina em teste, $h(x)$ o rótulo predito para x e $c(x)$ o rótulo real de x. A corretude do modelo $E(h)$ é dada pela probabilidade que existe de $h(x)$ e $c(x)$ serem idênticos (ZHANG et al., 2022). $E(h)$ pode ser calculado por meio da Equação 4.1.

$$E(h) = Pr_{x \sim \mathcal{D}}[h(x) = c(x)] \tag{4.1}$$

Outra propriedade relevante no contexto da abordagem proposta neste trabalho é *robustez*. A robustez de um sistema de software é normalmente definida como uma medida da capacidade em que um sistema ou componente consegue funcionar corretamente na presença de entradas inválidas ou condições de ambiente estressantes (SHAHROKNI; FELDT, 2013). Zhang et al. (ZHANG et al., 2022), entretanto, adaptaram essa definição para o contexto de sistemas baseado em aprendizado de máquina da seguinte maneira:

- **Robustez (*Robustness*)**: Seja S um sistema baseado em aprendizado de máquina. Seja $E(S)$ a corretude de S. Seja $\delta(S)$ o sistema de aprendizado de máquina com perturbações em quaisquer componentes de aprendizado de máquina, como os dados, o programa de aprendizado, o *framework* ou o modelo. A robustez de um sistema baseado em aprendizado de máquina é uma medida da diferença entre $E(S)$ e $E(\delta(S))$

$$r = E(S) - E(\delta(S)). \tag{4.2}$$

De acordo com a Equação 4.2, a robustez, portanto, mede a resiliência da corretude de um sistema baseado em aprendizado de máquina na presença de perturbações (ZHANG et al., 2022). Zhang et al. afirmam que existe uma subcategoria popular de robustez chamada de robustez adversária, que envolve perturbações que são projetadas para serem difíceis de detectar. Os autores classificaram a robustez adversária em robustez adversária local e robustez adversária global. A robustez adversária local diz respeito à robustez apresentada em relação a um dado de teste específico, enquanto a robustez adversária global mede a robustez contra todos os dados de teste.

A abordagem proposta neste trabalho é, de certa forma, fundamentada na *interpretabilidade* de alguns modelos (conforme será descrito no Capítulo 6). Portanto, interpretabilidade é outra propriedade relevante no contexto deste trabalho. Para que os usuários, e até mesmo os desenvolvedores e testadores consigam adquirir confiança sobre as decisões tomadas por um sistema baseado em aprendizado de máquina, pode ser necessário obter um entendimento da lógica de decisão empregada pelo sistema (LIPTON, 2018). Zhang et al. (2022) definem a interpretabilidade de um sistema baseado em aprendizado de máquina como o nível de entendimento que um observador consegue adquirir das decisões realizadas pelo sistema (e modelo em questão). Porém, as definições de interpretabilidade de modelos de aprendizado de máquina, encontradas na literatura, ainda são conflitantes (LIPTON, 2018). De acordo com Doshi-Velez e Kim (2017), uma definição matemática da interpretabilidade de sistemas baseados em aprendizado de máquina permanece como um problema não resolvido.

4.6 Desafios e Problemas Envolvidos no Teste de Sistemas Baseados em Aprendizado de Máquina

Além dos desafios conhecidos que são encontrados durante a realização do teste de sistemas tradicionais, o teste de sistemas baseados em aprendizado de máquina introduz uma série de problemas (RICCIO et al., 2020). O principal motivo do aumento da dificuldade para testar sistemas baseados em aprendizado de máquina é devido a mudança no

paradigma de desenvolvimento de sistemas introduzido pela utilização de aprendizado de máquina (BRAIEK; KHOMH, 2020). Um sistema baseado em aprendizado de máquina é construído seguindo um paradigma de programação orientado a dados, onde a lógica de decisão é obtida por meio de um procedimento de treinamento que utiliza um conjunto de dados que é submetido ao algoritmo de aprendizado (AMERSHI et al., 2019). Como resultado do procedimento de treinamento é obtido um modelo: um componente de aprendizado de máquina que é construído para produzir respostas para perguntas cujas respostas ainda não existem (MURPHY et al., 2007).

A natureza estatística associada a capacidade de realizar decisões autônomas dos sistemas baseados em aprendizado de máquina introduziu diversos problemas para a realização do teste desse tipo de sistema (ZHANG et al., 2022). Os três principais problemas que envolvem o teste de sistemas baseados em aprendizado de máquina são: *(i)* o espaço de entrada abrangente; *(ii)* a inexistência de oráculos de teste; e *(iii)* a escassez de critérios de teste. Tais problemas são discutidos nas próximas subseções.

4.6.1 Espaço de Entrada Abrangente

O espaço de entrada abrangente é uma das principais causas de falhas em sistemas baseados em aprendizado de máquina (RICCIO et al., 2020). Os sistemas baseados em aprendizado de máquina são normalmente utilizados em domínios de aplicação que lidam com um grande volume de dados. Tal característica contribui para a criação de um espaço de entrada abrangente e diverso, complicando o processo de seleção sistemática de um conjunto de dados de teste que seja eficaz em revelar falhas (MARIJAN et al., 2019).

Como afirmado por Riccio et al. (2020) gerar dados de teste capazes de representar apropriadamente o espaço de entrada abrangente dos sistemas baseados em aprendizado de máquina é uma tarefa importante, porém, complexa. Os dados de teste gerados devem ser diversos e realistas. Adicionalmente, os dados de teste devem ser acompanhados dos devidos resultados esperados. Os autores ainda afirmam que a definição de dados de teste tende a ser mais complicada para sistemas baseados em aprendizado de máquina pois o domínio válido (isto é, o subconjunto do espaço de entrada que é formado apenas por dados de entrada válidos) desses sistemas é, normalmente, ambíguo e não possui partições muito bem definidas.

4.6.2 Inexistência de Oráculos de Teste

O teste de sistemas de software tradicionais normalmente assume a existência de oráculos de teste: normalmente, assume-se a existência de um mecanismo, conhecido como oráculo de teste, por meio do qual pode ser avaliada a corretude da resposta produzida pelo sistema em teste em relação a entrada fornecida pelo desenvolvedor (MARIJAN

et al., 2019). Diante desse cenário, os oráculos de teste de sistemas tradicionais podem ser definidos previamente pelos desenvolvedores. Porém, o fato de muitos algoritmos de aprendizado de máquina serem programas probabilísticos faz com que a identificação de oráculos de teste seja um dos principais problemas envolvendo o teste de sistemas baseados em aprendizado de máquina (ZHANG et al., 2022).

Conceitualmente os oráculos de teste possuem a mesma função tanto no teste de sistemas tradicionais quanto no teste de sistemas baseados em aprendizado de máquina (ZHANG et al., 2022): em ambos os casos, oráculos *"codificam"* o comportamento esperado do sistema. O comportamento não determinístico associado aos domínios complexos em que os sistemas baseados em aprendizado de máquina são utilizados faz com que até mesmo os desenvolvedores encontrem dificuldades para definir o comportamento esperado de um sistema baseado em aprendizado de máquina (RICCIO et al., 2020). Essencialmente, as respostas fornecidas por um modelo de aprendizado de máquina são derivadas de um procedimento de treinamento envolvendo um subconjunto dos dados do domínio e não da especificação prévia de um conjunto de requisitos. Como resultado, durante a realização do teste de um sistema baseado em aprendizado de máquina, os resultados obtidos não podem ser facilmente comparados com os resultados esperados (MARIJAN et al., 2019): dado que os resultados esperados não são bem definidos ou facilmente identificáveis.

Zhang et al. (2022) afirmaram que a identificação de oráculos de teste de sistemas baseados em aprendizado de máquina permanece sendo um problema devido ao fato de muitas propriedades desse tipo de sistema serem difíceis de especificar. Na visão dos autores, a identificação de oráculos de teste para esse tipo de sistema é uma atividade demorada e trabalhosa dado que muitas vezes é necessário conhecimento específico do domínio do problema sendo abordado. Como um sistema baseado em aprendizado de máquina é utilizado para gerar respostas para um conjunto de entradas que são fornecidas pelos usuários somente após a disponibilização do sistema online, a corretude das respostas fornecidas aos usuários precisa ser confirmada manualmente (ZHANG et al., 2022).

4.6.3 Escassez de Critérios de Teste

Critérios de teste baseados em cobertura de código normalmente utilizados no teste de sistemas tradicionais (apresentados na Seção 2.5.1) não são eficazes quando aplicados ao teste de um sistema baseado em aprendizado de máquina (RICCIO et al., 2020). A lógica de decisão de um modelo de aprendizado de máquina não é escrita manualmente no código fonte do sistema sendo testado, ao invés disso, a lógica desses sistemas é aprendida (extraída a partir) por meio dos dados de treinamento (ZHANG et al., 2022). O código fonte do modelo de aprendizado de máquina normalmente é apenas uma sequência de chamadas de funções que são disponibilizadas no *framework* de aprendizado de máquina

utilizado. Portanto, como mostrado no estudo de Pei et al. (2017), com apenas um caso de teste aleatoriamente selecionado é possível alcançar 100% de cobertura de código de um componente de aprendizado de máquina.

Devido às diferenças fundamentais existentes no paradigma de desenvolvimento e no formato de representação lógica de sistema tradicionais e sistemas baseados em aprendizado de máquina, novos critérios de teste são necessários. Especificamente, critérios voltados para o teste de sistemas baseados em aprendizado de máquina devem levar em consideração as características específicas desses sistemas de software (ZHANG et al., 2022). Com essa finalidade, pesquisadores em teste de software vêm propondo novos critérios de teste que vão além da cobertura de código e que são específicos para o teste de modelos e sistemas baseados em aprendizado de máquina (BRAIEK; KHOMH, 2020). As próximas seções descrevem algumas técnicas que foram adaptadas ou desenvolvidas com o propósito de apoiar o teste de sistemas baseados em aprendizado de máquina.

4.7 Técnicas Aplicadas no Teste de Sistemas Baseados em Aprendizado de Máquina

Abordagens de teste adaptadas e inspiradas nas técnicas de teste de sistemas tradicionais estão sendo propostas na literatura com a finalidade de buscar soluções para os problemas e desafios envolvidos no teste de sistemas baseados em aprendizado de máquina que foram discutidos na seção anterior (BRAIEK; KHOMH, 2020). Nesta seção são brevemente discutidas algumas técnicas de teste que vêm sendo adaptadas para o teste de sistemas baseados em aprendizado de máquina.

4.7.1 Teste de Adversário

Os classificadores de aprendizado de máquina são vulneráveis a um tipo específico de ataque que envolve a utilização de pequenas modificações que são realizadas nos dados de entrada que são submetidos ao modelo em teste (BRAIEK; KHOMH, 2020). Os dados de entrada modificados são denominados de exemplos adversários. Os exemplos adversários são gerados por meio da realização de pequenas perturbações nos dados originais. As perturbações são projetadas para gerar exemplos adversários muito semelhantes aos dados originais, possibilitando "enganar" o modelo sendo testado e levar a erros de classificação (MARIJAN; GOTLIEB, 2020).

A utilização de exemplos adversários para identificação de falhas em modelos de aprendizado de máquinas foi estudada em diferentes domínios de aplicação, como por exemplo, em modelos utilizados para reconhecimento de imagem, texto, e linguagem natural (MARIJAN; GOTLIEB, 2020). De acordo com Marijan e Gotlieb (2020), parte

da literatura sobre teste adversário apresenta propostas de métodos defensivos que possibilitem combater os ataques de exemplos adversários aos modelos de aprendizado de máquina.

4.7.2 Teste Combinatório

O teste combinatório é uma técnica de teste que explora sistematicamente as combinações de diferentes parâmetros de entrada do programa em teste (NIE; LEUNG, 2011). Essa abordagem de teste é baseada na suposição de que diferentes parâmetros de entrada do programa em teste interagem uns com os outros e certas combinações de valores de entrada possuem alta probabilidade de revelar defeitos. A utilização das combinações de valores de parâmetros, em vez da realização de uma busca exaustiva em todo o espaço de entrada, permite realizar a atividade de teste de forma mais estruturada e também mais eficiente. Devido a essas vantagens, o teste combinatório tem sido amplamente utilizado no teste de sistemas baseados em aprendizado de máquina (AHUJA et al., 2022).

4.7.3 Teste Diferencial

O teste diferencial é uma abordagem que vem sendo aplicada com sucesso para o teste de sistemas tradicionais que não possuem oráculos de teste facilmente identificáveis (MCKEEMAN, 1998). Essa abordagem de teste utiliza pseudo-oráculos para mitigar a dificuldade de identificação dos oráculos de teste. No caso dos sistemas baseados em aprendizado de máquina, a atividade de teste diferencial é realizada por meio da utilização de vários modelos distintos que são treinados com um único conjunto de dados (AHUJA et al., 2022). Dado que diferentes modelos são treinados para realizar a mesma tarefa, todos os modelos também devem ser testados com os mesmos dados de teste. Consequentemente, os diferentes modelos testados são considerados oráculos de referência cruzada (BRAIEK; KHOMH, 2020), e as diferenças identificadas nos resultados gerados por diferentes modelos devem ser inspecionadas usando os vários modelos resultantes a fim de viabilizar a identificação de possíveis defeitos.

4.7.4 Teste Metamórfico

O teste metamórfico é uma técnica de teste que foi originalmente aplicada no teste de sistemas tradicionais que possuíam oráculos de teste difíceis ou até mesmo impossíveis de definir (AHUJA et al., 2022). Essa técnica de teste é caracterizada por realizar a detecção de defeitos por meio da utilização de relações de entradas e saídas que são identificadas no próprio domínio do programa em teste (SEGURA et al., 2020). Essas relações, que normalmente são chamadas de relações metamórficas, são utilizadas como pseudo-oráculos para realização da atividade de teste. Uma vez que, a identificação dos oráculos de teste pode não ser trivial (BRAIEK; KHOMH, 2020).

Diferentes abordagens de teste metamórfico foram propostas na literatura para viabilizar o teste de sistemas baseados em aprendizado de máquina (ZHANG et al., 2022). Essas abordagens, normalmente, possuem a detecção de defeitos existentes em modelos e em implementações de algoritmos e *frameworks* de aprendizado de máquina como foco principal (AHUJA et al., 2022). De acordo com Ahuja et al. (2022), o teste metamórfico é considerado uma alternativa interessante para o teste de modelos e algoritmos de aprendizado de máquina devido a complexidade e a natureza estocástica desses componentes. Como a identificação de oráculos de teste geralmente é impossível, são utilizadas relações metamórficas para possibilitar a verificação das respostas que são geradas pelos modelos de aprendizado de máquina que são testados.

4.7.5 Teste de Mutação

O teste de mutação é uma técnica de teste de sistemas tradicionais que é utilizada para avaliar a qualidade de um conjunto de testes e também para possibilitar a geração de um conjunto de casos de teste que com alta capacidade de revelar falhas (Jia; Harman, 2011). No contexto do teste de mutação, os mutantes são versões modificadas do programa sendo testado e o objetivo é avaliar um conjunto de testes por meio da sua capacidade de distinguir o programa original de seus mutantes. Um mutante é gerado por meio da aplicação de operadores de mutação que modificam o programa original introduzindo pequenas alterações no código-fonte. Essas alterações tem o objetivo de simular defeitos reais que poderiam ser cometidos por desenvolvedores (PAPADAKIS et al., 2019). Quando um caso de teste observa uma diferença no comportamento do programa original e de seu mutante, é dito que o caso de teste *mata* o mutante. A proporção de defeitos detectados (mutantes mortos) em relação a todos os defeitos injetados (mutantes gerados) é chamada de *escore de mutação*, do termo em inglês, *mutation score*.

Algumas propostas de abordagens de teste de mutação para o teste de sistemas baseados em aprendizado de máquina são encontradas na literatura (BRAIEK; KHOMH, 2020). Porém, de acordo com Marijan e Gotlieb (2020), a pesquisa de teste de mutação para sistemas baseados em aprendizado de máquina ainda está no seu estágio inicial. Para aplicar adequadamente o teste de mutação a esse tipo de sistema é necessário desenvolver operadores de mutação específicos para diferentes tipos de modelos e para domínios de aplicação distintos (MARIJAN; GOTLIEB, 2020): visto que operadores de mutação genéricos podem gerar defeitos que não são representativos dos defeitos reais aos quais os sistemas baseados em aprendizado de máquina podem estar sujeitos. Diante disso, parte da pesquisa de teste de mutação para esse tipo de sistema está concentrada no desenvolvimento de novos operadores de mutação para conjuntos de dados, algoritmos e modelos de aprendizado de máquina (ZHANG et al., 2022).

4.8 Critérios de Teste: Cobertura de Árvore de Decisão e Análise do Valor Limite de Árvore de Decisão

Conforme mencionado anteriormente, vários esforços têm sido conduzidos com o propósito de desenvolver critérios voltados para o teste de sistemas de software baseados em aprendizado de máquina. O trabalho de Santos et al. (2021), por exemplo, descreve dois critérios de cobertura para o teste de classificadores de aprendizado de máquina, a saber: cobertura de árvore de decisão, do termo em inglês, *decision tree coverage*; e análise do valor limite de árvore de decisão[2], do termo em inglês, *boundary value analysis*. A ideia principal dos dois critérios é utilizar a estrutura e as informações dos nós internos de um modelo de árvore de decisão para guiar a seleção de dados de teste. Os autores apresentaram a conjectura de que o aumento da cobertura de nós folha e de nós internos de um modelo de árvore de decisão pode promover a diversidade dos dados de entrada que serão selecionados para teste e como consequência pode resultar em casos de teste mais eficazes. Conforme descrito nos próximos capítulos, o trabalho de Santos et al. (2021) é um dos trabalhos cuja abordagem proposta neste documento foi baseada.

As definições dos dois critérios de cobertura propostos por Santos et al. (2021), são dadas da seguinte forma:

- **Cobertura de Árvore de Decisão (CAD)**: dado um modelo de árvore de decisão $\mathcal{M}$, um conjunto de casos de teste $\mathcal{T}$ é considerado adequado em relação ao critério CAD se existe em $\mathcal{T}$, casos de teste que percorrem a árvore da raiz até todos os nós folhas pelo menos uma vez (SANTOS et al., 2021).
- **Análise do Valor Limite de Árvore de Decisão (AVLAD)**: os casos de teste são projetados a fim de cobrir valores limites válidos dos nós internos. Especificamente, este critério requer a seleção de valores que exploram o limite inferior ou o limite superior de cada nó interno da árvore (SANTOS et al., 2021).

Ao aplicar o critério CAD em um modelo de árvore decisão, cada caminho da raiz à cada folha da árvore representa um requisito de teste. Dado que existe apenas um caminho da raiz para cada nó folha, o número de requisitos de teste do critério CAD será igual ao número de caminhos da raiz à cada folha da árvore de decisão (SANTOS et al., 2021).

O critério AVLAD é inspirado no critério análise do valor limite utilizado no teste de sistemas tradicionais. Tal critério faz uso da suposição de que casos de teste que

[2] Os autores denominaram o segundo critério de cobertura apenas como *Boundary Value Analysis* (SANTOS et al., 2021). Porém, com objetivo de evitar confusão com o critério análise do valor limite utilizado no teste de sistemas tradicionais, que foi discutido na Seção 2.4.2. Neste trabalho, o segundo critério será referenciado como análise do valor limite de árvore de decisão.

exploram condições limites são necessários para o teste eficaz de um sistema (MYERS et al., 2011). Ao aplicar o critério AVLAD, o intervalo de valores codificado em nós internos são analisados e utilizados para derivar os casos de teste (SANTOS et al., 2021). Os valores limites desse intervalo são importantes dado que eles são derivados das combinações de regras inferidas das características existentes no conjunto de dados de treinamento.

A principal diferença entre os dois critérios reside na ênfase especial aos valores limites dos nós internos que é requerida pelo critério AVLAD ao percorrer a árvore da raiz até cada nó folha. Da perspectiva de requisitos de teste, ambos os critérios possuem o objetivo de satisfazer o mesmo número de requisitos de teste: que é equivalente ao número de caminhos da raiz à cada folha da árvore de decisão sendo testada (SANTOS et al., 2021).

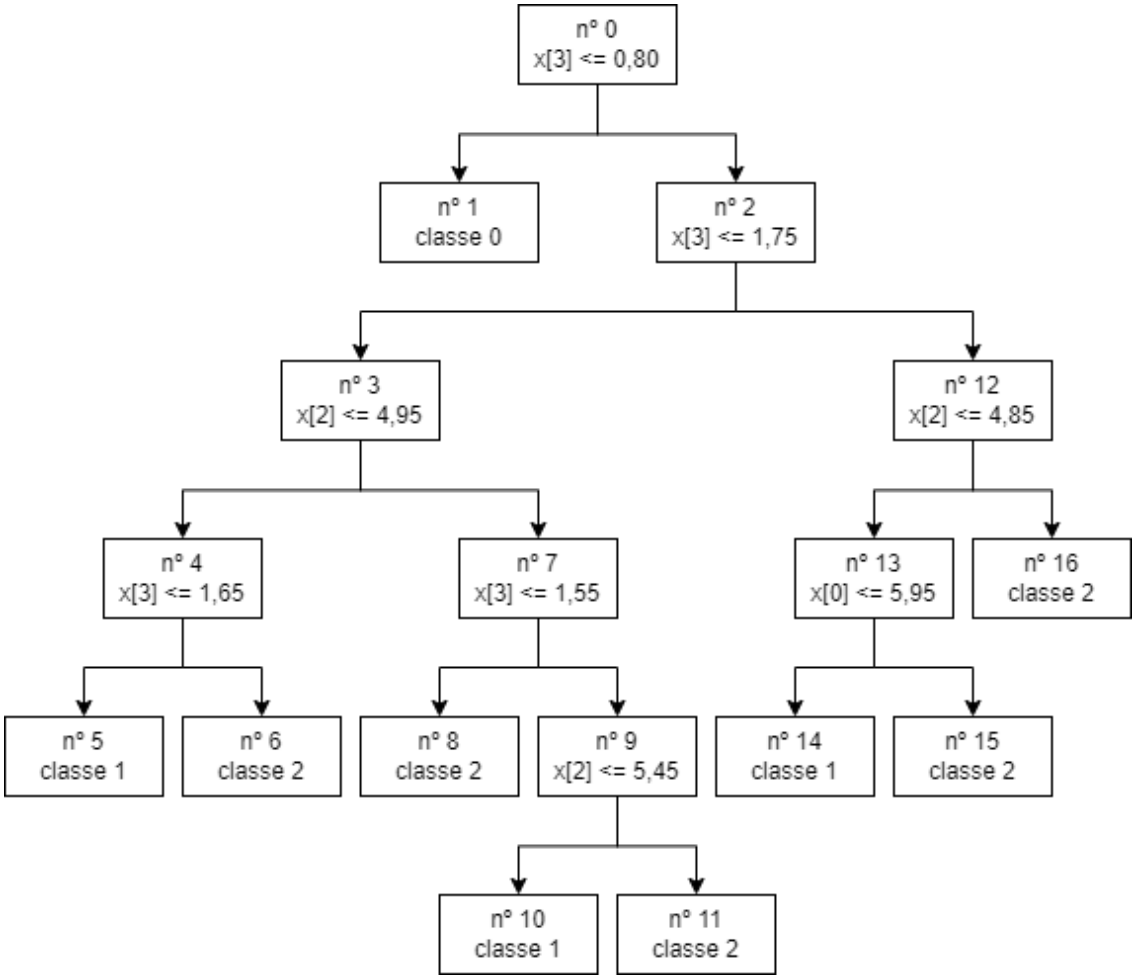

Figura 13 – Modelo de árvore de decisão construído com a utilização do conjunto de dados Iris. As características representadas na figura correspondem a: x[0] = comprimento da sépala, x[1] = largura da sépala, x[2] = comprimento da pétala, x[3] = largura da pétala. Retirado de (SANTOS et al., 2021).

Com o intuito de demonstrar a utilização dos critérios de teste CAD e AVLAD será apresentado a seguir um exemplo que foi retirado do trabalho de Santos et al. (2021). A Figura 13 apresenta uma árvore de decisão que representa um modelo que foi construído ao fornecer o conjunto de dados Iris para um algoritmo de árvore de decisão. A árvore

apresentada possui 17 nós, dos quais: o nó 0 é a raiz da árvore, os nós {2, 3, 4, 7, 9, 12, 13} são nós internos e os nós {1, 5, 6, 8, 10, 11, 14, 15, 16} são nós folhas.

De acordo com a árvore apresentada na Figura 13, as amostras de dados capazes de percorrer o caminho [0, 2, 12, 16] da raíz (nó 0) até a folha (nó 16) devem pertencer a classe 2 (que representa as flores Iris do tipo Virginica). Ou seja, o modelo construído aprendeu que as flores Iris que possuírem a característica x[2] (comprimento da pétala) maior que *4,85* e a característica x[3] (largura da pétala) maior que *1,75* são do tipo Virginica (classe 2). Com base nessas informações, os critérios de teste CAD e AVLAD podem ser utilizados para possibilitar a construção de casos de teste (SANTOS et al., 2021).

Ao aplicar os critérios de teste CAD e AVLAD no caminho [0, 2, 12, 16] será produzido um requisito de teste para cada critério (SANTOS et al., 2021). Consequentemente, deverá ser construído um caso de teste para satisfazer cada requisito. Para satisfazer o requisito produzido pelo critério CAD poderá ser selecionada a amostra de dado de teste (6.3, 3.3, 6.0, 2.5). Como resultado esperado, deverá ser utilizado o valor *2*, uma vez que, a classe determinada pelo nó folha de nº 16 é a classe *2*. Dessa forma, ao aplicar o critério CAD no caminho [0, 2, 12, 16] poderá ser construído o caso de teste descrito na Tabela 1.

Tabela 1 – Exemplo de caso de teste construído com base no critério CAD para satisfazer o requisito de teste gerado pelo caminho [0, 2, 12, 16]. Retirado de (SANTOS et al., 2021).

Comprimento da Sépala - x[0]	**Largura da Sépala - x[1]**	**Comprimento da Pétala - x[2]**	**Largura da Pétala - x[3]**	**Resultado Esperado**
6,3	3,3	6,0	2,5	2

De modo semelhante, deverá ser construído um caso de teste para satisfazer o requisito de teste produzido com aplicação do critério AVLAD no caminho [0, 2, 12, 16]. Como discutido anteriormente, a principal diferença existente entre os critérios CAD e AVLAD refere-se à exigência adicional do critério AVLAD para a seleção de dados de teste cujos valores sejam próximos dos valores limites das características que aparecem em nós internos do caminho que produziu o requisito de teste (SANTOS et al., 2021). Os valores limites das características que aparecem em nós internos do caminho [0, 2, 12, 16] são $x[2] = 4,85$ e $x[3] = 1,75$. Como a característica $x[2]$ é um limite inferior, será selecionado um valor apenas 10% maior do que *4,85*, resultando em: $x[2] = 5,335$. Da mesma forma, como $x[3]$ também é um limite inferior, será selecionado o valor: $x[3] = 1,925$. O caso de teste que poderá ser construído para satisfazer o requisito de teste produzido com aplicação do critério AVLAD no caminho [0, 2, 12, 16] é descrito na Tabela 2.

Tabela 2 – Exemplo de caso de teste construído com base no critério AVLAD para satisfazer o requisito de teste gerado pelo caminho [0, 2, 12, 16]. Retirado de (SANTOS et al., 2021).

Comprimento da Sépala - x[0]	**Largura da Sépala - x[1]**	**Comprimento da Pétala - x[2]**	**Largura da Pétala - x[3]**	**Resultado Esperado**
5,00	3,00	5,335	1,925	2

4.9 Considerações Finais

Neste capítulo foi abordado o tema de teste de sistemas baseados em aprendizado de máquina que está diretamente relacionado à proposta deste trabalho. A discussão realizada no capítulo iniciou apresentando o termo *sistema baseado em aprendizado de máquina* que é utilizado para denotar um tipo de sistema de software que possui componentes que dependem da utilização de técnicas de aprendizado de máquina (RICCIO et al., 2020). Foi destacado que o desenvolvimento e utilização desse tipo de sistema está cada vez mais comum e que muitas das técnicas de teste consolidadas no teste de sistemas tradicionais não são diretamente aplicáveis ao teste de sistemas baseados em aprendizado de máquina (ZHANG et al., 2022). Fatos que motivaram os pesquisadores a desenvolverem abordagens de teste adaptadas das técnicas de teste de sistemas tradicionais para viabilizar o teste de sistemas baseados em aprendizado de máquina (BRAIEK; KHOMH, 2020).

Foi chamada a atenção para o fato de que o termo teste tem interpretações distintas nas comunidades científicas de aprendizado de máquina e de teste de software. Enquanto a comunidade de aprendizado de máquina está interessada nos métodos de avaliação de desempenho que foram discutidos na Subseção 3.3.1. A comunidade de teste de software está interessada em testar os diferentes componentes que integram um sistema baseado em aprendizado de máquina em relação a diferentes atributos de qualidade. Como por exemplo, testar a corretude de um sistema até mesmo para os dados de teste cujos resultados esperados são desconhecidos (MARIJAN et al., 2019).

Toda a discussão realizada neste capítulo foi centrada na interpretação de teste de sistemas baseados em aprendizado de máquina que é dada pela comunidade científica de teste de software. Diante disso, foram apresentados os processos de teste online e offline que foram abordados por Zhang et al. (2022). Foram discutidos sobre os diferentes componentes de teste que integram um sistema baseado em aprendizado de máquina, a saber: os dados; o programa de treinamento; o *framework* de aprendizado de máquina; e o modelo de aprendizado de máquina. Também foram apresentados dois níveis de teste específicos para o teste de sistemas baseados de máquina que foram introduzidos por Riccio et al. (2020), a saber: o nível de teste de modelo e o nível de teste de entrada.

Algumas propriedades (atributos de qualidade) de teste de sistemas baseados em aprendizado de máquina também foram discutidas neste capítulo, sendo a corretude a pro-

priedade de maior interesse para este trabalho. Não menos importantes, foram discutidos três dos principais problemas que envolvem o teste de sistemas baseados em aprendizado de máquina e que são responsáveis por tornar o teste desse tipo de sistema tão desafiador, a saber: o espaço de entrada abrangente; a inexistência de oráculos de teste; e a escassez de critérios de teste (MARIJAN et al., 2019). Em sequência, foi realizada uma discussão sobre as principais técnicas de teste de sistemas tradicionais que estão sendo adaptadas para serem aplicadas no teste de sistemas baseados em aprendizado de máquina. Por fim, foram apresentados os critérios de teste CAD e AVLAD que foram propostos por Santos et al. (2021) e cuja aplicação foi adaptada para o desenvolvimento da abordagem proposta neste trabalho.

5 Trabalhos Relacionados

O teste de sistemas de software baseados em aprendizado de máquina apresenta diversos desafios quando comparado ao teste de sistemas tradicionais (MARIJAN et al., 2019). Pesquisadores em engenharia e teste de software vêm propondo diferentes abordagens com o objetivo de viabilizar a realização do teste desse tipo de sistema de maneira sistemática (BRAIEK; KHOMH, 2020) e (RICCIO et al., 2020). Como destacado na Seção 4.7, as técnicas consolidadas no teste de sistemas tradicionais estão sendo adaptadas para o contexto de sistemas baseados em aprendizado de máquina. Diante disso, neste capítulo são discutidas algumas abordagens de teste que foram propostas na literatura e que são úteis para ampliar a discussão sobre o teste de sistemas baseados em aprendizado de máquina que é realizada neste trabalho.

O restante deste capítulo está organizado conforme a seguir. Na Seção 5.1 são apresentadas algumas abordagens que foram propostas para realização do teste de dados (conjuntos de dados) utilizados para conduzir o processo de treinamento ou avaliação de modelos de aprendizado de máquina. Na Seção 5.2 são discutidas abordagens que foram propostas para viabilizar o teste de implementações de algoritmos de aprendizado de máquina. Na Seção 5.3 são apresentadas as abordagens para o teste de modelos de aprendizado de máquina ou aprendizado profundo. Por fim, a Seção 5.4 apresenta as considerações finais do capítulo.

5.1 Teste de Conjuntos de Dados

Analistas de dados, normalmente, realizam a limpeza de conjuntos de dados de forma iterativa: *(i)* um subconjunto dos dados disponíveis é selecionado; *(ii)* em seguida, é realizada a limpeza dos dados contidos no subconjunto selecionado; *(iii)* os dados selecionados (e limpos) são analisados; e, ao final do processo, *(iv)* com base nos resultados da análise realizada, outro subconjunto de dados é selecionado e o processo é repetido. Krishnan et al. (2016) exploraram essa natureza iterativa do processo de limpar conjuntos de dados no contexto de aprendizado de máquina. Os autores propuseram o `ActiveClean`, um *framework* que permite limpar conjuntos de dados de treinamento de modelos de aprendizado de máquina de forma iterativa, possibilitando melhorar o desempenho dos modelos treinados progressivamente, à medida que os dados são limpos. O `ActiveClean` utiliza um conjunto de otimizações para selecionar do conjunto de dados de treinamento, um subconjunto de amostras de dados que apresentam considerável probabilidade de serem amostras de dados sujos *(dirty data)*. Em seguida, o *framework* solicita que o analista transforme ou remova cada amostra de dados sujos existentes no subconjunto selecionado.

Por fim, o *framework* atualiza os parâmetros do modelo e continua o processo de treinamento utilizando apenas os dados parcialmente limpos. Esse processo é repetido até que nenhuma amostra de dados sujos possa ser identificada.

Modelos de aprendizado de máquina podem ser sensíveis a defeitos e inconsistências existentes no conjunto de dados utilizado para treinar o modelo (JAMES et al., 2021). Um tipo de inconsistência de dados que pode existir em um conjunto de dados é a chamada inconsistência de domínio, que ocorre quando o valor de um atributo de uma amostra de dados está fora do domínio permitido para aquele atributo. Krishnan et al. (2017) exploraram a detecção e o reparo automatizado desse tipo de inconsistência. Os autores propuseram o *framework* `BoostClean`: tal *framework* implementa um *pipeline* de operações de transformações de dados com o objetivo de limpar o conjunto de dados que será utilizado para treinar o modelo de aprendizado de máquina. O *framework* proposto por Krishnan et al. utiliza uma técnica de *boosting* para identificar a melhor sequência de operações de reparos de dados que ao ser aplicada ao conjunto de dados de treinamento resulta em um modelo aprimorado em termos de acurácia. Krishnan et al. avaliaram o *framework* proposto em 8 conjuntos de dados que continham amostras de dados com defeitos reais. Os resultados indicam que o `BoostClean` é capaz de aumentar a acurácia de predição de modelos de aprendizado de máquina em até 7%.

Linters de código são ferramentas de qualidade de software utilizadas para notificar o desenvolvedor sobre a existência de determinados tipos de problemas no código (TóMASDóTTIR et al., 2017). Inspirados nos *linters* de código, Hynes et al. (2017) introduziram o conceito de *linters* de dados. Os autores definiram um *linter* de dados como uma ferramenta que analisa as características de um conjunto de dados de treinamento em relação a um tipo de modelo específico e fornece recomendações de transformações de dados que podem ser realizadas com o objetivo de aumentar a probabilidade de que o modelo seja capaz de aprender com as características existentes no conjunto de dados. Hynes et al. também implementaram um *linter* de dados que ajuda os desenvolvedores de aprendizado de máquina a limpar, transformar e extrair características de conjuntos de dados de treinamento. O *linter* de dados proposto pelos autores identifica determinados tipos de defeitos existentes em conjunto de dados, como por exemplo, características com valores numéricos em escalas diferentes, características com valores faltantes e características com valores incorretos e produz avisos contendo recomendações de como corrigir cada característica identificada com os possíveis defeitos. Segundo Hynes et al. uma das principais vantagens oferecidas pela utilização do *linter* de dados proposto reside na redução do esforço humano necessário para realizar a limpeza e a engenharia de características dos conjuntos de dados de treinamento que serão utilizados para construir modelos de aprendizado de máquina.

Dados sujos (*dirty data*) podem ser classificados em diferentes tipos como, por

exemplo, dados faltantes, dados inconsistentes e dados conflitantes. Qi et al. (2018) investigaram os efeitos desses três tipos de dados sujos no desempenho de dezesseis algoritmos clássicos de aprendizado de máquina. Os autores injetaram dados sujos em treze conjuntos de dados conhecidos e em seguida avaliaram o desempenho de modelos treinados com esses dados e com diferentes algoritmos de classificação e regressão. Utilizando as métricas precisão, revocação e F_1 para avaliar o desempenho dos algoritmos de classificação e métricas específicas para os algoritmos de regressão, Qi et al. constataram que os dados sujos são capazes de gerar um efeito negativo no desempenho de praticamente todos os modelos avaliados. Adicionalmente, os autores observaram que a magnitude do impacto negativo depende do tipo e da quantidade de dados sujos existentes nos conjuntos de dados utilizados. Entretanto, os resultados indicam que o aumento da dimensão do conjunto de dados utilizado para treinar os modelos tende a reduzir o efeito negativo dos dados sujos.

Qi et al. (2021) atualizaram o trabalho de Qi et al. (2018) ao conduzir uma avaliação experimental para avaliar o impacto que dados sujos causam em modelos de aprendizado de máquina. Essa atualização apresenta duas novas métricas para possibilitar medir o nível de tolerabilidade que diferentes modelos de aprendizado de máquina apresentam em relação a conjuntos de dados que possuem dados sujos. Utilizando as métricas propostas, os autores conduziram uma avaliação experimental com doze tipos de modelos de classificação e clusterização. Para treinar os modelos foram utilizados nove conjuntos de dados que tiveram dados sujos injetados. Com base nos resultados, novamente foi constatado que três fatores relacionados aos conjuntos de dados de treinamento podem afetar o desempenho de um modelo de aprendizado de máquina: *(i)* o tipo de defeito existente nos dados; *(ii)* a quantidade de dados que apresentam defeitos; e *(iii)* a quantidade total de dados de treinamento.

5.2 Teste de Implementação de Algoritmos de Aprendizado de Máquina

O trabalho de Murphy et al. (2007), foi um dos primeiros a mencionar a ideia de testar sistemas baseados em aprendizado de máquina. Dada a dificuldade encontrada para identificar oráculos de teste para esse tipo de sistema, os autores classificaram os sistemas baseados em aprendizado de máquina como sistemas não testáveis (ZHANG et al., 2022). Murphy et al. apresentaram uma abordagem que foi utilizada para criar e executar casos de teste contra dois algoritmos de ranking, o SVM-Light e o MartiRank. Com a abordagem proposta os autores descobriram defeitos de implementação e discrepâncias de comportamento existentes entre os dois algoritmos testados. Posteriormente, Murphy et al. (2008) apresentaram um conjunto de propriedades dos sistemas baseados em apren-

dizado de máquina que poderiam ser adaptadas e utilizadas como relações metamórficas para detectar defeitos de implementação existentes nesse tipo de sistema.

Xie et al. (2011) propuseram um conjunto de relações metamórficas para o teste de implementações de algoritmos de aprendizado de máquina supervisionado. As relações metamórficas propostas pelos autores são baseadas nos seguintes tipos de transformações: *(i)* consistência com transformações de *affine*; *(ii)* permutação de rótulos (tipos de classes); *(iii)* permutação de atributos; *(iv)* adição de atributos não informativos; *(v)* adição de atributos informativos; *(vi)* consistência com re-predição; *(vii)* adição de amostras de dados de treinamento; *(viii)* adição de rótulos (tipos de classes); *(ix)* remoção de rótulos (tipos de classes); e *(x)* remoção de amostras de dados de treinamento. Os autores realizaram um estudo de caso com o objetivo de avaliar a eficácia dessas relações metamórficas para detecção de possíveis defeitos nas implementações dos algoritmos K-NN e Naive Bayes disponibilizados no pacote `WEKA` (WITTEN et al., 2009). A utilização das relações metamórficas durante a realização do estudo de caso possibilitou a identificação de defeitos existentes na implementação do algoritmo Naive Bayes. Os autores também realizaram uma avaliação do conjunto de relações metamórficas utilizando a ferramenta de teste de mutação `MuJava` (MA et al., 2006). Como resultado da avaliação foi descoberto que as relações metamórficas propostas são capazes de detectar 90% dos defeitos introduzidos pela ferramenta `MuJava` em alguns algoritmos de aprendizado de máquina supervisionado disponibilizados no pacote `WEKA`.

Dwarakanath et al. (2018) também propuseram conjuntos de relações metamórficas para o teste de algoritmos de aprendizado de máquina. Porém, as relações propostas pelos autores foram específicas para dois algoritmos de aprendizado de máquina mais complexos: máquinas de vetores de suporte, do termo em inglês, *support vector machine* (SVM) e um classificador de imagens baseado em aprendizado profundo denominado *residual neural network* (ResNet). As relações metamórficas propostas para realização do teste do SVM foram: *(i)* permutação de atributos (características) de amostras de dados de treinamento e amostras de dados de teste; *(ii)* permutação da ordem das amostras de dados de treinamento; *(iii)* deslocamento de características de amostras de dados de treinamento e amostras de dados de teste; e *(iv)* escalonamento linear dos valores de características de amostras de dados de teste. Para o teste do ResNet os autores propuseram as seguintes relações metamórficas: *(i)* permutação de canais de entrada RGB de amostras de dados de treinamento e amostras de dados de teste; *(ii)* permutação da ordem de operação de convolução de amostras de dados de treinamento e amostras de dados de teste; *(iii)* normalização dos valores de características das amostras de dados de teste; e *(iv)* escalonamento dos valores de características das amostras de dados de teste. Os autores utilizaram a ferramenta de teste de mutação `MutPy`[1] para realizar uma avaliação

[1] https://github.com/mutpy/mutpy

experimental sobre a eficácia das relações metamórficas propostas na detecção de defeitos de implementação. Os resultados mostraram que, em média, as relações metamórficas propostas foram capazes de detectar 71% dos defeitos de implementação simulados pelos mutantes gerados com a `MutPy`.

Braiek e Khomh (2019b) propuseram um *framework* de teste baseado em propriedades chamado de `TFCheck`. O `TFCheck` possibilita aos desenvolvedores testar implementações de programas de treinamento de redes neurais profundas. Além do *framework*, Braiek e Khomh também forneceram um guia prático que descreve determinados problemas e as respectivas rotinas de verificação que os desenvolvedores podem utilizar para detectar e corrigir defeitos em suas implementações de programas de aprendizado profundo. O *framework* implementa as rotinas de verificação que foram propostas pelos autores, possibilitando assim que os desenvolvedores possam monitorar e automaticamente depurar programas de aprendizado profundo que forem desenvolvidos com base no *framework* `TensorFlow` (ABADI et al., 2016). Braiek e Khomh também conduziram um estudo de caso para avaliar a eficácia do `TFCheck` na detecção de defeitos de implementação: os resultados sugerem que o `TFCheck` é capaz de detectar com sucesso defeitos existentes em implementações de programas de treinamento de redes neurais profundas.

Odena et al. (2019) adaptaram uma técnica existente no teste de software tradicional conhecida como técnica de teste *fuzzing* guiado por cobertura e a combinaram com a técnica de teste baseado em propriedades para testar a implementação de redes neurais. Os autores introduziram a noção de teste *fuzzing* guiado por cobertura para redes neurais e descreveram como algoritmos de vizinho mais próximo podem ser utilizados para verificar a cobertura de uma rede neural. Os autores desenvolveram uma biblioteca de código aberto chamada de `TensorFuzz` que implementa a abordagem de teste de redes neurais proposta por eles. A biblioteca desenvolvida combina as técnicas de teste baseado em propriedades com as técnicas de teste *fuzzing* guiado por cobertura formando um sistema pronto para utilização que possibilita testar a utilização de redes neurais em uma variedade de aplicações. Por meio da realização de uma avaliação experimental, Odena et al. buscaram demonstrar que a biblioteca `TensorFuzz` consegue encontrar diversos tipos de defeitos numéricos em redes neurais treinadas com os algoritmos disponibilizados no *framework* `TensorFlow` (ABADI et al., 2016).

5.3 Teste de Modelos de Aprendizado de Máquina e Aprendizado Profundo

Um modelo generativo pode ser utilizado para *perturbar* um conjunto de dados de treinamento com o objetivo de gerar dados sintéticos ligeiramente diferentes, porém, que possuem muitas das propriedades dos dados originais (BRAIEK; KHOMH, 2020).

Goodfellow et al. (2014) introduziram um tipo de modelo generativo denominado de redes adversárias generativas, do termo em inglês, *generative adversarial networks*. Uma rede adversária generativa é uma abordagem de aprendizado generativo que utiliza duas redes neurais distintas, sendo uma rede geradora e outra discriminadora. A rede geradora é treinada com o objetivo de gerar dados sintéticos realistas para tentar *enganar* a rede discriminadora, a rede discriminadora é treinada com o objetivo de distinguir as amostras de dados reais das amostras de dados sintéticos. As redes adversárias generativas têm sido utilizadas com sucesso na realização de transformações avançadas de imagens que possibilitam gerar imagens sintéticas para realização do teste de sistemas de veículos autônomos (ZHANG et al., 2022).

Pei et al. (2017) propuseram o `DeepXplore`: o primeiro *framework* de teste de caixa branca para realização do teste sistemático de modelos de aprendizado profundo. A fim de contornar o problema da inexistência de oráculos de teste, Pei et al. adaptaram a técnica de teste diferencial para o contexto de aprendizado profundo. `DeepXplore` utiliza um conjunto de modelos de redes neurais profundas semelhantes (construídas com a finalidade de resolver o mesmo problema) para possibilitar a detecção de defeitos por meio da exposição de diferenças no comportamento dos modelos quando testados com as mesmas entradas. Pei et al. também adaptaram a ideia de cobertura de testes consolidada no teste de sistemas tradicionais e propuseram um critério de teste denominado *cobertura de neurônios* que utiliza a cobertura de neurônios de redes neurais profundas para guiar a geração de dados de teste. O *framework* `DeepXplore` utiliza um algoritmo de otimização conjunta para gerar dados de teste que são criados com o objetivo de maximizar a cobertura de neurônios e a exposição de comportamentos distintos dos modelos de redes neurais profundas sendo testados. A avaliação experimental realizada indica que o *framework* é capaz de gerar dados de teste que cobrem uma quantidade de neurônios até 34,4% maior do que os dados de teste selecionados aleatoriamente. Além disso, a utilização do `DeepXplore` possibilitou identificar milhares de comportamentos incorretos em sistemas de carros autônomos.

Tian et al. (2018) propuseram a ferramenta `DeepTest` para auxiliar no teste de sistemas de carros autônomos. Para criar casos de teste eficazes a `DeepTest` utiliza uma técnica de *busca gulosa* que identifica as combinações de diferentes transformações de imagens que possibilitam aumentar a ativação de neurônios do modelo de rede neural profunda em teste. Os autores acreditam que as combinações de transformações de imagens capazes de ativar o maior número de neurônios são aquelas que melhor simulam os casos excepcionais que as câmeras dos carros autônomos estão sujeitas no mundo real. Para lidar com o problema do oráculo de teste, a ferramenta faz uso de relações metamórficas que possibilitam identificar os comportamentos errôneos exibidos pelo modelo de aprendizado profundo em teste.

Com o objetivo de sintetizar imagens realistas para o teste de sistemas de veículos autônomos, Zhang et al. (2018) propuseram o *framework* `DeepRoad`. A primeira abordagem de aprendizado profundo não supervisionado que utilizou teste metamórfico e redes adversárias generativas para gerar imagens de condução de veículos em diferentes condições climáticas como, por exemplo, em climas de chuva ou neve. Zhang et al. realizaram um experimento utilizando três modelos Udacity de carros autônomos para avaliar a eficácia do `DeepRoad`. Os resultados da avaliação sugerem que o modelo generativo utilizado pelo `DeepRoad` permite gerar cenas realistas em climas de chuva e neve que são capazes de detectar milhares de inconsistências no comportamento de sistemas de veículos autônomos.

Guo et al. (2018) propuseram o `DLFuzz`: o primeiro *framework* para realização de *fuzzing* em modelos de aprendizado profundo. O `DLFuzz` expandiu a ideia de cobertura de neurônios proposta no `DeepXplore` introduzindo a utilização de estratégias específicas como, por exemplo, a seleção de neurônios que são frequentemente ativados durante a execução do modelo em teste e a priorização de neurônios que possuem maiores pesos. Guo et al. realizaram um experimento com o objetivo de avaliar a eficácia do `DLFuzz` em cenários de testes semelhantes aos que foram utilizados para avaliar a eficácia do `DeepExplore`. Comparado com o `DeepXplore`, `DLFuzz` foi capaz de gerar uma quantidade de amostras de dados de teste até 584,62% maior com um custo computacional até 20,11% menor.

Ma et al. (2018) afirmam que buscar apenas por valores altos de acurácia não é suficiente para medir a qualidade de um modelo de aprendizado profundo. Segundo os autores, medir a qualidade de um sistema que utiliza modelos de aprendizado profundo apenas por meio das respostas fornecidas pelo modelo é superficial, no sentido de que o entendimento da rede neural artificial e as atividades dos neurônios internos da rede não são levadas em consideração. Dessa forma, Ma et al. introduziram um conjunto de critérios de teste para sistemas baseados em aprendizado profundo chamado de `DeepGauge`. Os critérios propostos pelos autores possuem diferentes granularidades e, portanto, podem ser aplicados em diferentes níveis, possibilitando explorar características distintas do modelo de aprendizado profundo em teste.

Engstrom et al. (2019) realizaram um estudo com o objetivo de avaliar a robustez de classificadores de imagens baseados em redes neurais convolucionais. Os autores mostraram que mesmo a utilização de transformações espaciais simples como translações e rotações de imagens são suficientes para reduzir consideravelmente o desempenho desse tipo de classificador. Diante dessas constatações, os autores enfatizam a necessidade da robustez adversária de redes neurais artificiais ser levada em consideração ao realizar o treinamento desse tipo de modelo, visto que perturbações de dados de entrada caracterizadas por transformações simples como rotações e translações de imagens podem ocorrer

naturalmente até mesmo no ambiente normal de utilização do modelo.

Braiek e Khomh (2019a) introduziram a `DeepEvolution`, uma abordagem de teste de software baseado em busca adaptada para o teste de modelos de aprendizado profundo. A `DeepEvolution` possibilita a detecção de inconsistências e possíveis defeitos existentes nos modelos de redes neurais profundas em teste por meio da utilização de transformações metamórficas e de metaheurísticas baseadas em população. Braiek e Khomh avaliaram a eficácia da `DeepEvolution` utilizando dois modelos de redes neurais profundas aplicados no reconhecimento de imagens. Os resultados indicam que a `DeepEvolution` consegue identificar defeitos e aumentar a cobertura de neurônios dos modelos de redes neurais em teste. Os resultados também indicaram que a `DeepEvolution` consegue superar o `TensorFuzz` em relação a sua capacidade de detecção de defeitos.

Ma et al. (2019) investigaram se o teste combinatório pode ser utilizado para testar efetivamente sistemas que utilizam modelos de aprendizado profundo. Um motivador para a pesquisa realizada refere-se ao fato de que durante a realização da atividade de teste pode ser necessário considerar as interações que ocorrem entre diferentes neurônios que constituem a rede neural profunda em teste. Diante disso, Ma et al. propuseram um conjunto de critérios de cobertura baseados em teste combinatório para realização do teste de modelos de aprendizado profundo. Os autores também introduziram o `DeepCT`, um *framework* de teste combinatório para modelos de aprendizado profundo que possibilita a geração automatizada de dados de testes para redes neurais profundas construídas com base nos *frameworks* `Keras` ou `TensorFlow` (ABADI et al., 2016).

Kim et al. (2019) propuseram um critério de teste denominado *surprise adequacy*, que quantifica o quanto um determinado dado de teste é surpreendente para um modelo de aprendizado profundo. A ideia central empregada por esse critério é que para um dado de teste ser eficaz ele deve ser suficientemente *surpreendente* em relação aos dados de treinamento. O *nível de surpresa* oferecido por um dado de teste é medido com base na diferença de comportamento que o modelo em teste exibe ao ser executado com o dado de teste em relação ao comportamento exibido com dados de treinamento.

Gopinath et al. (2019) introduziram o `DeepCheck`, uma abordagem baseada em execução simbólica para testar modelos de aprendizado profundo que traduz as ativações dos neurônios de uma rede neural profunda em estruturas de decisão do tipo *IF-Else*. O `DeepCheck` implementa técnicas de análise simbólica e as aplica no contexto de classificação de imagens para abordar dois problemas: *(i)* identificação de pixels importantes existentes em imagens; e *(ii)* criação de ataques adversários. Gopinath et al. conduziram uma avaliação experimental utilizando o conjunto de dados MNIST que demonstrou que a execução simbólica pode efetivamente identificar amostras de dados de teste adversário resultantes da perturbação de pixels ou pares de pixels em imagens.

Conforme discutido na Seção 4.8, Santos et al. (2021) introduziram dois critérios

de teste para modelos de aprendizado de máquina: o critério CAD e o critério AVLAD. Para avaliar a eficácia dos critérios propostos, os autores realizaram um experimento utilizando modelos K-NN treinados com 12 conjuntos de dados distintos. Considerando os resultados, os autores afirmam que os critérios propostos podem ser utilizados para orientar a geração de dados de teste eficazes para modelos de aprendizado de máquina.

5.4 Considerações Finais

Neste capítulo foram apresentados alguns trabalhos identificados na literatura que podem estar diretamente ou marginalmente relacionados à proposta deste trabalho. Os trabalhos apresentados foram organizados em três seções: a Seção 5.1 descreveu alguns trabalhos relacionados ao teste de conjuntos de dados; a Seção 5.2 abordou trabalhos relacionados ao teste de implementações de algoritmos de aprendizado de máquina; e a Seção 5.3 apresentou diversos trabalhos relacionados ao teste de modelos de aprendizado de máquina e aprendizado profundo.

É importante destacar que apesar da proposta deste trabalho ser diretamente relacionada ao teste de modelos de aprendizado de máquina, optou-se por incluir neste capítulo alguns trabalhos relacionados ao teste de conjuntos de dados e ao teste de implementações de algoritmos de aprendizado de máquina por questões de completude. Especificamente, essa decisão foi tomada com a finalidade de ampliar a discussão sobre o teste de sistemas baseados em aprendizado de máquina, reforçando uma ideia apresentada por Zhang et al. (2022), de que o teste de sistemas baseados em aprendizado de máquina é ainda mais diverso do que o teste de sistemas tradicionais e que defeitos podem existir em diferentes componentes que constituem esse tipo de sistema, assim cada componente precisa ser testado individualmente.

Dessa forma, é válido enfatizar que os trabalhos sobre o teste de conjuntos de dados e sobre o teste de implementações de algoritmos de aprendizado de máquina que foram discutidos nas Seções 5.1 e 5.2 não são diretamente relacionados à proposta deste trabalho pois tais trabalhos não abordam diretamente o teste de modelos de aprendizado de máquina. Por exemplo, o trabalho de Braiek e Khomh (2019b) introduziu um *framework* denominado de `TFCheck` que possibilita a realização do teste baseado em propriedades de implementações de programas de treinamento de redes neurais profundas construídas com base no *framework* `TensorFlow`. Apesar do *framework* `TFCheck` utilizar diretamente a técnica de teste baseado em propriedades que também é utilizada na abordagem proposta neste trabalho, foi considerado que o *framework* `TFCheck` não relaciona-se diretamente com a abordagem proposta visto que o artefato que é testado com a proposta de Braiek e Khomh é a implementação de programas de treinamento de redes neurais profundas e não os modelos de aprendizado de máquina ou de aprendizado profundo. Porém, por

considerar que a proposta do *framework* `TFCheck` contribuiu para o enriquecimento da discussão sobre o teste de sistemas baseados em aprendizado de máquina que é realizada neste trabalho foi optado por incluir o trabalho de Braiek e Khomh neste capítulo.

Os trabalhos discutidos neste capítulo que podem ser considerados como relacionados mais diretamente com a proposta deste trabalho são aqueles que abordam o teste de modelos de aprendizado de máquina ou aprendizado profundo (que foram discutidos na Seção 5.3). Porém, a ideia de "diretamente relacionado" deve ser interpretada com cautela até mesmo para os trabalhos que foram discutidos nesta seção. Especificamente, com exceção do trabalho de Santos et al. (2021) que introduziu os critérios de teste CAD e AVLAD, todos os demais trabalhos discutidos na Seção 5.3 são propostas que possuem como finalidade possibilitar o teste de modelos de aprendizado profundo, ou seja, possibilitar o teste de diferentes tipos de redes neurais profundas.

O trabalho de Pei et al. (2017) foi pioneiro ao propor um critério de cobertura para o teste de modelos baseados em redes neurais profundas (ZHANG et al., 2022). O *framework* `DeepXplore` proposto por Pei et al. inspirou o desenvolvimento de várias abordagens que posteriormente foram apresentadas na literatura com o objetivo de viabilizar o teste de modelos de aprendizado profundo, incluindo, o `DeepTest` (TIAN et al., 2018) e o `DeepRoad` (ZHANG et al., 2018). Até mesmo os critérios de teste CAD e AVLAD propostos por Santos et al. (2021) que são fundamentados na noção de cobertura de um modelo de árvore de decisão foram inspirados pela noção de cobertura de neurônios que foi introduzida no trabalho de Pei et al. com o *framework* `DeepXplore`.

Com exceção das propostas de abordagens de teste para modelos de aprendizado profundo que em sua maioria foram inspiradas pelo trabalho de Pei et al. (2017), não foram identificadas propostas da comunidade científica de teste de software que sejam específicas para o teste de modelos construídos com os algoritmos de aprendizado de máquina clássico (que não são baseados na utilização de redes neurais profundas). Inspirado nessas abordagens que estão sendo propostas para o teste de modelos de aprendizado profundo foi identificada uma oportunidade para propor uma abordagem de teste para modelos clássicos de aprendizado de máquina, que é discutida a seguir.

6 Abordagem de Teste Baseado em Propriedades para Modelos Clássicos de Aprendizado de Máquina

A crescente utilização de sistemas baseados em aprendizado de máquina nos mais diversos domínios de aplicação (SARKER, 2021) aumentou consideravelmente a necessidade de avaliar a qualidade e a confiabilidade desse tipo de sistema (ZHANG et al., 2022). Conforme discutido na Seção 4.6, o teste de sistemas baseados em aprendizado de máquina apresenta diversos desafios e possui vários problemas carentes de uma solução definitiva. Pesquisadores em engenharia e teste de software têm dedicado sua atenção para esse tipo de sistema e, como resultado, diferentes abordagens de teste baseadas em diferentes técnicas foram propostas na literatura (RICCIO et al., 2020). Diante dos problemas envolvidos no teste desse tipo de sistema, este capítulo apresenta uma abordagem de teste baseado em propriedades que foi desenvolvida com o objetivo de possibilitar o teste automatizado de modelos clássicos de aprendizado de máquina.

O restante deste capítulo está organizado conforme a seguir. A Seção 6.1 define o escopo de aplicação da abordagem de teste baseado em propriedades para modelos clássicos de aprendizado de máquina que é proposta neste trabalho. Na Seção 6.2 são apresentadas as principais motivações da abordagem proposta. A Seção 6.3 apresenta uma visão geral do funcionamento da abordagem em questão. Na Seção 6.4 o processo de geração das propriedades é apresentado e descrito em detalhes. Na Seção 6.5 é realizada uma demonstração do funcionamento do processo de geração das propriedades. A Seção 6.6 apresenta a ferramenta que foi desenvolvida com o propósito de possibilitar a automatização do processo de geração das propriedades e, por fim, na Seção 6.7 são apresentadas as considerações finais do capítulo.

6.1 Escopo de Aplicação da Abordagem Proposta

Antes de iniciar uma discussão sobre a abordagem proposta, é necessário esclarecer o entendimento da palavra “clássico” que é utilizada nos termos algoritmos clássicos de aprendizado de máquina e modelos clássicos de aprendizado de máquina, que são recorrentemente utilizados neste trabalho. A palavra clássico é utilizada para fazer uma distinção entre os algoritmos e modelos que pertencem ao aprendizado de máquina clássico daqueles que pertencem ao aprendizado profundo (do termo em inglês *deep learning*). Algoritmos de árvore de decisão, k-vizinhos mais próximos, regressão linear e naive bayes pertencem

ao aprendizado de máquina clássico (ZHANG et al., 2022). O aprendizado profundo, por outro lado, é caracterizado pela utilização de redes neurais profundas de múltiplas camadas de unidades de processamento (LECUN et al., 2015), como por exemplo, redes neurais convolucionais (KIM, 2014) e redes neurais recorrentes (GRAVES et al., 2013).

Foi adotada essa distinção entre duas categorias de aprendizado de máquina com o objetivo de deixar claro que a abordagem de teste de modelos proposta neste trabalho foi projetada para testar modelos construídos com os algoritmos de aprendizado de máquina clássico. É importante ressaltar também que essa distinção entre duas categorias de aprendizado de máquina é adotada em outros trabalhos encontrados na literatura (ZHANG et al., 2022).

6.2 Motivação

No capítulo anterior foram discutidos alguns trabalhos que foram propostos na literatura com o objetivo de possibilitar o teste de diferentes componentes relacionados ao desenvolvimento de um sistema baseado em aprendizado de máquina. Com base nos trabalhos descritos, é possível constatar que existe um interesse considerável por parte dos pesquisadores no desenvolvimento de novas abordagens e técnicas para o teste de modelos baseados em aprendizado profundo. Dos trabalhos discutidos no capítulo anterior, as principais propostas, relacionadas ao teste de nível de modelo são exclusivamente direcionadas para o teste de modelos de aprendizado profundo. Exemplos desse tipo de proposta são: `DeepXplore` (PEI et al., 2017), `DeepTest` (TIAN et al., 2018), `DeepGauge` (MA et al., 2018), `DeepRoad` (ZHANG et al., 2018), `DeepEvolution` (BRAIEK; KHOMH, 2019a) e o `DeepCheck` (GOPINATH et al., 2019). Constatação semelhante também é realizada no trabalho de Zhang et al. (2022). Conforme afirmado pelos autores, dos 144 trabalhos que eles investigaram, 56 são propostas de abordagens e técnicas que foram projetadas exclusivamente para o teste de componentes de aprendizado profundo. Enquanto os 88 trabalhos restantes são relacionados a diferentes abordagens e técnicas para o teste de componentes de aprendizado de máquina de forma geral. Os autores ainda complementam, ao afirmar que até o ano de 2017, os trabalhos eram, em sua maioria, propostas para o teste de aprendizado de máquina de forma geral. Porém, a partir do ano de 2018, houve um considerável aumento no número de publicações projetadas exclusivamente para abordar aplicações baseadas em aprendizado profundo. Apesar das publicações relacionadas ao teste de aprendizado de máquina de forma geral também terem aumentado a partir de 2018.

Essas constatações indicam que o teste em nível de modelo de aprendizado profundo tem sido um tema de considerável interesse por parte dos pesquisadores. Fato que pode ser justificado pelos significativos avanços recentes que a utilização dos modelos de

aprendizado profundo tem possibilitado em diversas áreas de aplicação (POUYANFAR et al., 2018). Resultando no aumento da necessidade de desenvolvimento de novas técnicas que são direcionadas para o teste desse tipo específico de modelo. Porém, essas constatações também podem indicar uma oportunidade para propor melhorias às técnicas de teste de modelos clássicos existentes.

A abordagem proposta no contexto deste trabalho adapta a abordagem de teste baseado em propriedades e a aplicação dos critérios de teste CAD e AVLAD com o propósito de abordar os problemas e desafios relacionados ao teste de sistemas baseados em aprendizado de máquina que foram discutidos na Seção 4.6. A abordagem de teste baseado em propriedades para modelos clássicos de aprendizado de máquina que foi desenvolvida neste trabalho é descrita na seção seguinte.

6.3 Visão Geral da Abordagem Proposta

Conforme mencionado, o principal objetivo da abordagem proposta neste trabalho é possibilitar a realização do teste de modelos clássicos de aprendizado de máquina. Especificamente, modelos clássicos de aprendizado supervisionado construídos para solucionar problemas de classificação, ou seja, classificadores. Para alcançar esse objetivo é preciso fazer um refinamento no escopo de aplicação da abordagem proposta, definindo precisamente não apenas o componente de teste, mas também a propriedade (atributo de qualidade) que será testada, o tipo de processo de teste (online ou offline) e o nível de teste que serão realizados. A Tabela 3 descreve brevemente as dimensões de interesse da abordagem de teste proposta neste trabalho. Conforme é destacado na tabela, a abordagem proposta utiliza a técnica de teste baseado em propriedades juntamente com os critérios de teste CAD e AVLAD para possibilitar a realização do teste offline da corretude de modelos clássicos de aprendizado de máquina.

Tabela 3 – Dimensões de interesse da abordagem de teste proposta.

Dimensão de Interesse	**Descrição**
Processo de Teste	Processo de teste offline
Componente de Teste	Modelos de aprendizado de máquina (classificadores)
Propriedade de Teste	Corretude
Nível de Teste	Nível de modelo
Técnica de Teste	Teste baseado em propriedades
Critérios de Teste	CAD e AVLAD

Ao propor uma abordagem de teste para modelos de aprendizado de máquina, é necessário levar em consideração os principais problemas que afetam o teste de sistemas baseados em aprendizado de máquina que foram discutidos na Seção 4.6:

- **Problema 1:** espaço de entrada abrangente;
- **Problema 2:** inexistência de oráculos de teste;
- **Problema 3:** escassez de critérios de teste adaptados para tal contexto.

Para desenvolver a abordagem proposta neste trabalho levando em consideração a existência desses três problemas, optou-se pela utilização da técnica de teste baseado em propriedades e dos critérios de cobertura de árvore de decisão que foram discutidos na Seção 4.8. Com a utilização da técnica de teste baseado em propriedades é possível gerar uma grande quantidade de dados de teste de maneira automatizada. Porém, dada a abrangência do espaço de entrada dos modelos de aprendizado de máquina, apenas gerar dados de teste de maneira aleatória, pode não ser suficiente para encontrar os dados de teste que realmente podem ser eficazes em revelar falhas. Dessa forma, a abordagem de teste proposta utiliza os critérios de cobertura de árvore de decisão propostos por Santos et al. (2021) a fim de derivar um conjunto de propriedades que verificam determinadas características dos modelos de aprendizado de máquina.

A abordagem proposta utiliza os critérios de teste CAD e AVLAD para derivar a partir da estrutura de um modelo de árvore de decisão os intervalos de valores que supostamente contêm os dados de teste mais eficazes. Essencialmente, os critérios de cobertura propostos por Santos et al. (2021) são utilizados para guiar a geração de dados de teste baseado em propriedades. Dessa forma, é possível tirar vantagem da lógica de predição codificada na estrutura de uma árvore de decisão para definir a distribuição dos dados de teste que devem ser gerados, possibiliando assim explorar o espaço de entrada do modelo de uma forma mais eficaz.

A abordagem proposta é fundamentada na hipótese de que as propriedades derivadas por meio da utilização dos critérios CAD e AVLAD possibilitam explorar o espaço de entrada dos modelos de aprendizado de máquina de forma mais apropriada. Como resultado, a geração de dados de teste é mais eficaz em relação a uma abordagem aleatória. Adicionalmente, visto que as propriedades resultantes incorporam informações sobre a estrutura interna dos modelos sendo testados, é possível que os dados de teste gerados por meio da abordagem proposta sejam mais eficazes do que os dados presentes no conjunto de treinamento dos modelos (que normalmente são extraídos do conjunto de dados por meio da utilização de técnicas de avaliação de desempenho como, por exemplo, o método de validação cruzada por *k-fold*). A utilização da técnica de teste baseado em propriedades juntamente com os critérios de teste CAD e AVLAD, garante que ao executar as propriedades geradas obtem-se uma suíte de testes adequada em relação a esses dois critérios: o que representa outra vantagem em relação à falta de critérios de adequação das técnicas tradicionais de avaliação de desempenho de modelos de aprendizado de máquina.

O problema da inexistência de oráculos de teste não é tratado diretamente na abordagem proposta neste trabalho. Todavia, a utilização dos critérios CAD e AVLAD para obter informações possibilita construir asserções de teste contendo resultados esperados que são considerados como aproximações dos resultados esperados verdadeiros.

O problema da escassez de critérios de teste também não é tratado diretamente com a abordagem proposta neste trabalho. São utilizados os critérios CAD e AVLAD na abordagem proposta, porém, o objetivo não é propor um novo critério de teste, mas sim, propor uma nova abordagem de teste que adapta a técnica de teste baseado em propriedades para o contexto de teste de sistemas baseados em aprendizado de máquina. A abordagem proposta é constituída por dois processos distintos: *(i)* o processo de geração de propriedades e *(ii)* o processo de execução de propriedades. Na próxima subseção será discutido o processo de geração de propriedades.

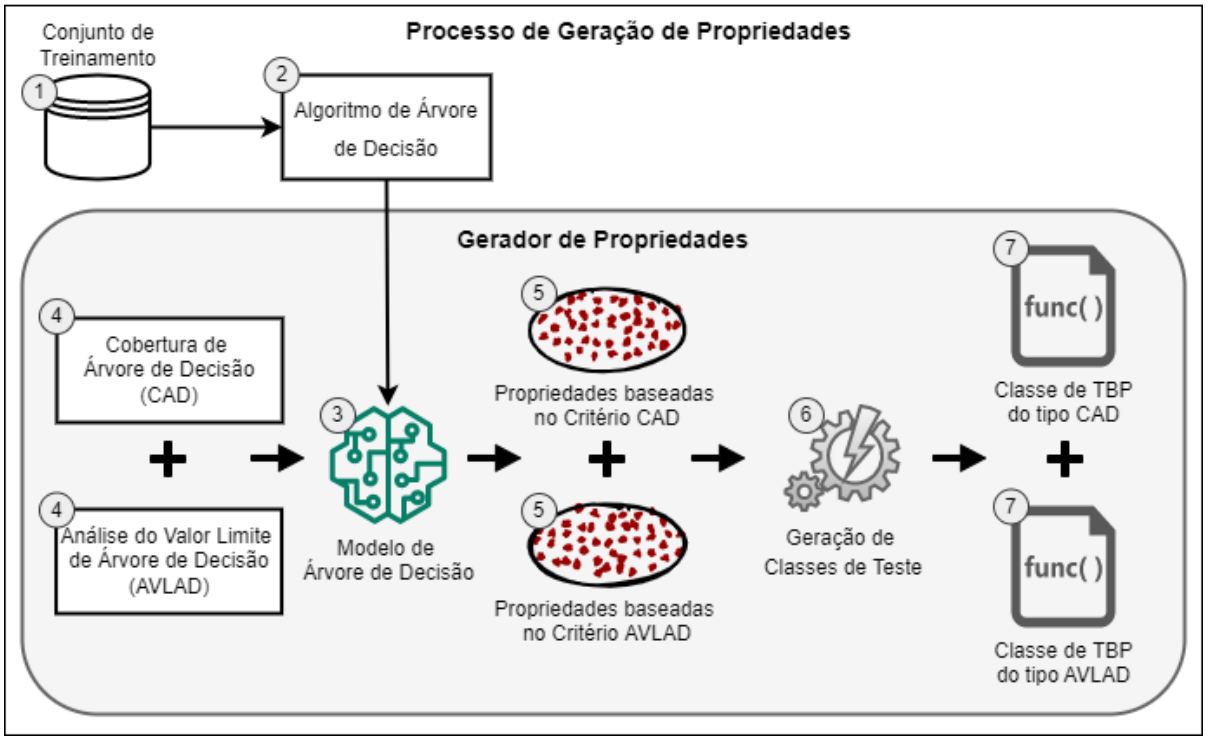

Figura 14 – Visão geral do processo de geração de propriedades.

6.3.1 Processo de Geração de Propriedades

A Figura 14 mostra uma visão geral do funcionamento do processo de geração de propriedades. Conforme pode ser visualizado, o processo de geração de propriedades começa com o fornecimento do conjunto de dados de treinamento para um algoritmo de árvore de decisão. Em seguida, o algoritmo de árvore de decisão realiza o treinamento (criação) de um modelo de árvore de decisão utilizando o conjunto de dados de treinamento fornecido. Além disso, é importante ressaltar que o modelo de árvore de decisão treinado

neste processo é utilizado apenas para gerar as propriedades. As propriedades começam a ser geradas ao ser fornecido o modelo de árvore de decisão para o gerador de propriedades.

O gerador de propriedades é o elemento central do processo ilustrado na Figura 14. Esse elemento possui duas responsabilidades principais: *(i)* aplicar os critérios CAD e AVLAD para extrair da estrutura da árvore, as informações que serão necessárias para gerar as propriedades de teste; e *(ii)* gerar classes de teste (`arquivos.py`) executáveis contendo as propriedades que serão utilizadas na realização do teste baseado em propriedades de modelos de aprendizado de máquina.

Nesta etapa a abordagem proposta difere do trabalho de Santos et al. (2021): Santos et al. propuseram os critérios de teste CAD e AVLAD para possibilitar a seleção (amostragem) de amostras de dados de teste de conjuntos de dados pré-existentes, a proposta deste trabalho utiliza os critérios de cobertura propostos pelos autores para gerar conjuntos de propriedades que são então "codificadas" em classes de teste executáveis compatíveis com uma ferramenta de teste baseado em propriedades. Dessa forma, a abordagem proposta neste trabalho possibilita que testadores deleguem a responsabilidade de selecionar ou gerar dados de teste para a ferramenta de teste baseado em propriedades. Na próxima subseção será discutido como é realizado o processo de execução das propriedades geradas.

6.3.2 Processo de Execução de Propriedades

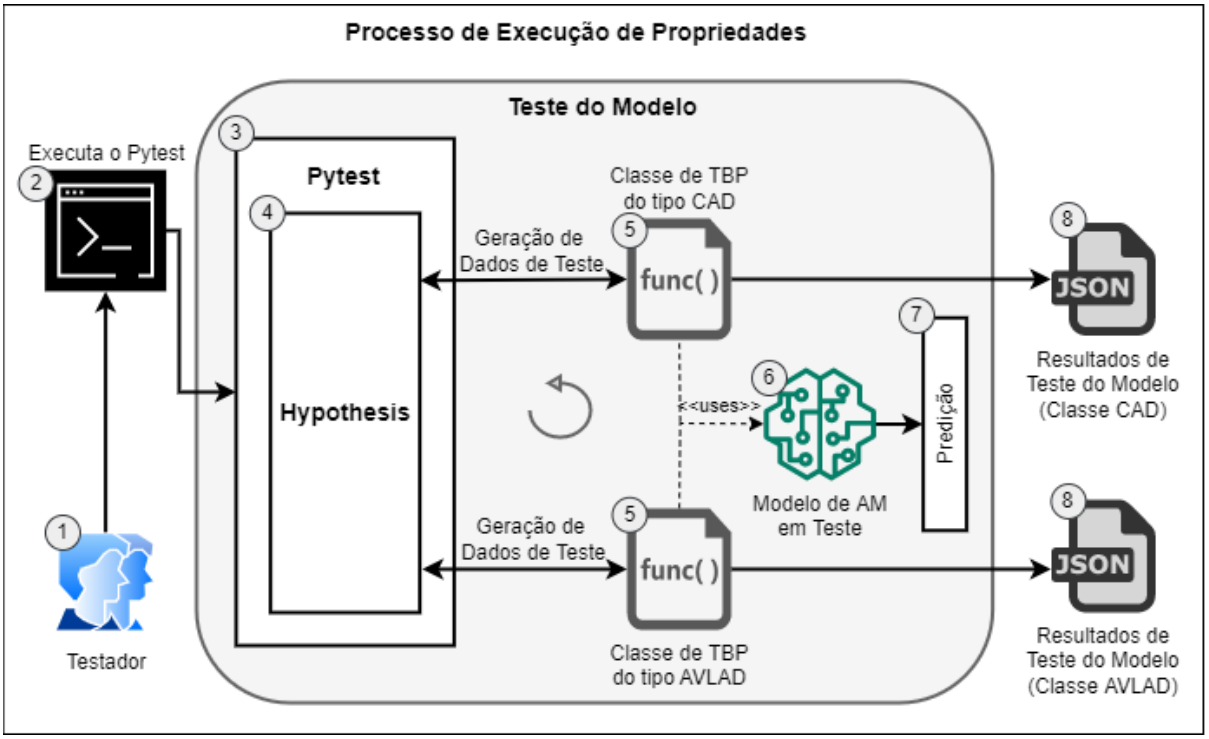

Figura 15 – Visão geral do processo de execução de propriedades.

Duas classes de teste são obtidas após a realização do processo de geração de

propriedades da abordagem proposta. Uma classe de teste baseado em propriedades do tipo CAD e outra classe de teste baseado em propriedades do tipo AVLAD. Nesta subseção é apresentado como essas classes devem ser utilizadas para realização do teste baseado em propriedades de modelos clássicos de aprendizado de máquina.

A Figura 15 apresenta uma visão geral de como funciona o processo de execução de propriedades da abordagem proposta. Em um ambiente Python com o *framework* `pytest` e a biblioteca `Hypothesis` instalados, o testador deve executar as duas classes de teste contra o modelo clássico de aprendizado de máquina em teste. Ao executar as classes de teste, o *framework* `pytest` utiliza a `Hypothesis` para gerar amostras de dados de teste para cada propriedade existente nas classes de teste. Cada amostra de dado de teste gerada será submetida ao modelo sendo testado que, por sua vez, realiza a predição e retorna o resultado. Na sequência, o resultado retornado é comparado com o resultado esperado presente na asserção de teste da propriedade em execução.

Durante a execução de cada classe de teste são armazenados em memória todas as amostras de dados de teste gerados pela `Hypothesis`, todos os resultados gerados pelo modelo sendo testado e todos os resultados esperados de cada propriedade. Ao término da execução de cada classe de teste, é retornado para o testador um arquivo no formato JSON contendo todas as informações mencionadas anteriormente, conforme ilustrado na Figura 15, item 8.

Os arquivos JSON contendo as informações resultantes da realização da atividade de teste são fornecidos para o testador para contornar uma limitação decorrente do comportamento padrão do *framework* `pytest` e da biblioteca `Hypothesis`. Especificamente, como essas duas ferramentas foram propostas originalmente para o teste de sistemas tradicionais e não para o teste de modelos de aprendizado de máquina, as informações relacionadas à atividade de teste que elas fornecem não são suficientes para que o testador consiga fazer uma análise completa relacionada ao desempenho e ao comportamento do modelo testado. A disponibilização desses dados em um arquivo estruturado permite que o testador consiga fazer uma análise aprofundada dos dados de teste gerados e dos resultados utilizando as métricas de avaliação que forem necessárias.

6.3.3 Tecnologias Adotadas Como Referência

Embora a abordagem de teste de modelos de aprendizado de máquina seja independente de tecnologias específicas, a implementação de tal abordagem envolveu decisões relacionadas à utilização de determinadas tecnologias. Por exemplo, ao incorporar a técnica de teste baseado em propriedades, inevitavelmente, fez-se necessário tomar decisões relacionadas a qual biblioteca deveria ser adotada para implementação de tal funcionalidade. Conforme destacado por Aniche (2022), um dos grandes diferenciais do teste baseado em propriedades é a utilização de uma ferramenta geradora de dados de teste.

De maneira semelhante, conforme afirmado por Zhang et al. (2022), o desenvolvimento de sistemas baseados em aprendizado de máquina também é consideravelmente dependente da utilização de bibliotecas e *frameworks* que disponibilizam implementações prontas de diferentes algoritmos de aprendizado de máquina. Diante desses fatores, optou-se por implementar a abordagem proposta neste trabalho incorporando as tecnologias específicas que foram adotadas como referência para o seu desenvolvimento.

Ao visualizar o funcionamento do gerador de propriedades que foi apresentado na Figura 14, é possível entender de forma mais precisa a necessidade de adotar um conjunto de tecnologias e ferramentas específicas como referência. Conforme pode ser visualizado, o gerador de propriedades não apenas gera as propriedades, mas também gera classes de teste executáveis contendo (especificando) as propriedades geradas. Para gerar uma classe de teste executável é preciso gerar código-fonte compatível com uma determinada ferramenta de teste baseado em propriedades. Neste contexto, a ferramenta de teste baseado em propriedades pode depender da utilização de um *framework* específico de testes automatizado e, consequentemente, o *framework* de testes automatizado é implementado em uma linguagem alvo. Assim, a escolha da linguagem de programação alvo tem um papel fundamental e determina a escolha das bibliotecas e *frameworks* utilizados para implementação da abordagem proposta.

A linguagem de programação adotada como referência para a abordagem proposta foi a linguagem Python. De acordo com Nguyen et al. (2019), por oferecer um ecossistema de bibliotecas e *frameworks* de aprendizado de máquina diverso e abrangente Python tornou-se uma das linguagens mais utilizada nas áreas de ciência de dados e aprendizado de máquina. O *framework* de aprendizado de máquina adotado como referência foi o `scikit-learn`. Conforme discutido na Seção 3.4.1, o `scikit-learn` é um *framework* de código aberto que fornece implementações do estado da arte de muitos algoritmos clássicos de aprendizado de máquina. Por meio da utilização de uma interface comum, denominada de ***`estimator`***, o *framework* disponibiliza uma interface programável padronizada para todos os modelos treinados com os diferentes algoritmos disponibilizados no *framework*. Esse foi um fator determinante para a escolha do `scikit-learn` como referência para a abordagem proposta.

A biblioteca de teste baseado em propriedades adotada como referência foi a `Hypothesis` (MACIVER et al., 2019). Conforme discutido na Seção 2.6.2, a `Hypothesis` é a biblioteca de teste baseado em propriedades mais utilizada na linguagem Python. O *framework* de teste automatizado adotado como referência foi o `pytest`. É importante mencionar que a `Hypothesis` e o `pytest` possuem compatibilidade e integração, fatores que contribuíram para a decisão de utilizar essas duas ferramentas conjuntamente.

6.4 Especificação do Funcionamento do Gerador de Propriedades

Esta seção descreve o funcionamento do gerador de propriedades da abordagem proposta. O gerador de propriedades, como mostrado na Figura 14, é o componente central do processo de geração de propriedades. Uma das suas funções principais é gerar automaticamente as propriedades de teste. Tais propriedades são geradas por meio da aplicação dos critérios de teste CAD e AVLAD sobre o modelo de árvore de decisão que é construído com o conjunto de dados de treinamento fornecido. Cada caminho da raiz à cada folha da árvore de decisão é utilizado para gerar uma propriedade. O rótulo determinado pelo nó folha de cada caminho é utilizado para construir a asserção presente no teste da propriedade gerada para o caminho. Dessa forma, o número total de propriedades geradas para cada classe de teste (`arquivo.py`) é igual ao número de caminhos da raiz à cada folha da árvore de decisão. Por exemplo, caso um modelo de árvore de decisão possua dez caminhos partindo do nó raiz até os nós folhas, será gerado duas classes de teste (uma para cada critério), contendo dez propriedades em cada classe. A única diferença que existirá entre as duas classes de teste refere-se ao fato de que uma classe terá dez propriedades baseadas no critério CAD e a outra dez propriedades baseadas no critério AVLAD, que por definição, é um critério mais restritivo do que o anterior.

O gerador de propriedades também deve definir um conjunto de geradores de dados da biblioteca `Hypothesis` para cada propriedade. A quantidade de geradores definidos para cada propriedade deve ser igual ao número de características existentes no conjunto de dados de treinamento utilizado para treinar o modelo de árvore de decisão. Os geradores de dados são definidos para instruir a biblioteca `Hypothesis` sobre como deverão ser gerados os valores de cada característica das amostras de dados de teste que serão geradas por cada propriedade.

A configuração correta da distribuição de dados dos geradores disponibilizados em uma ferramenta de teste baseado em propriedades é uma tarefa fundamental. Na abordagem proposta, o gerador de propriedades é o responsável por identificar os valores limites dos nós internos de cada caminho da raiz à cada folha da árvore de decisão com o propósito de derivar os intervalos de valores que serão utilizados para controlar a distribuição de dados dos geradores utilizados em cada propriedade. O processo de derivação dos intervalos de valores dos geradores de cada propriedade é realizado em um processo iterativo composto por quatro passos:

- **Passo 1**: derivação de um intervalo de valores *(mínimo, máximo)* para cada característica que aparece em nós internos dos caminhos que levam a cada uma das folhas.
- **Passo 2**: quando é possível extrair somente o valor mínimo ou máximo de um determinado nó, o limite ausente é substituído pelo menor (quando o limite inferior

é ausente) ou maior (quando o limite superior é ausente) valor da característica que foi identificado no conjunto de dados de treinamento utilizado para treinar a árvore de decisão.

- **Passo 3**: criação de um conjunto de valores com dados obtidos do conjunto de dados de treinamento para as características que não apareceram no caminho da raiz à folha que gerou a propriedade.
- **Passo 4**: limita-se os intervalos de valores *(mínimo, máximo)* para valores próximos do valor mínimo ou do valor máximo, caso a propriedade gerada pertença a uma classe de teste que será construída baseada no critério AVLAD.

Ao gerar uma classe de teste baseado em propriedades que utiliza o critério CAD, o gerador de propriedades deve realizar os passos 1, 2 e 3 descritos acima para gerar cada uma das propriedades. Os quatro passos são realizados somente quando é gerada uma classe de teste que utiliza o critério AVLAD: a restrição aplicada aos intervalos de valores que controlam a distribuição de dados dos geradores de cada propriedade é a principal diferença entre as propriedades geradas com os critérios CAD e AVLAD.

6.5 Demonstração do Funcionamento do Gerador de Propriedades

Esta seção ilustra o funcionamento do gerador de propriedades por meio de um exemplo. Na Figura 16 é apresentado uma árvore de decisão que representa o modelo obtido por meio da aplicação do algoritmo de árvore de decisão ao conjunto de dados Iris. A árvore de decisão resultante possui 15 nós, sendo 7 nós internos e 8 nós folhas. O caminho da raiz (nó 0) até o nó folha (nó 5), [0, 2, 3, 4, 5], é usado para exemplificar como o gerador de propriedades aplica o critério CAD e o critério AVLAD para gerar propriedades contendo as devidas asserções de teste e os intervalos de valores necessários para configurar a distribuição de dados dos geradores de cada propriedade.

Por meio do caminho [0, 2, 3, 4, 5], o gerador gera uma propriedade que classifica todas as amostras de dados capazes de alcançar o nó folha 5 como pertencendo a classe 1 (*versicolor*). Especificamente, a propriedade gerada terá uma asserção que verifica se a predição (resultado) que o modelo sendo testado gera para amostras de dados que alcançam o nó folha 5 é igual a classe 1. Para gerar novas amostras válidas (novos dados de teste), o gerador de propriedades deriva um intervalo de valores para cada característica do conjunto de dados Iris que aparece em nós internos do caminho percorrido da raiz até o nó folha 5. Neste exemplo, as únicas características que apareceram em nós internos do caminho [0, 2, 3, 4, 5] são as características comprimento e largura da pétala.

Após a realização do Passo 1 do processo de derivação dos intervalos de valores que serão utilizados nos geradores da propriedade deste exemplo, o gerador de propriedades

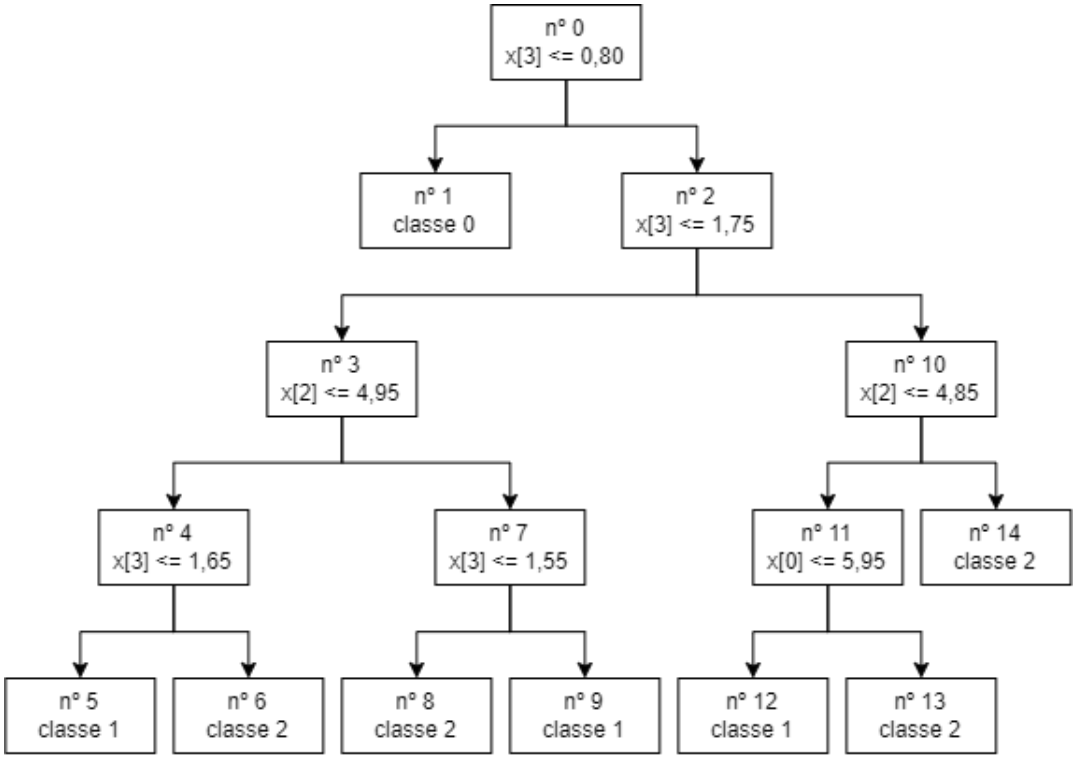

Figura 16 – Modelo de árvore de decisão gerado por meio da utilização do conjunto de dados Iris. As características representadas na figura correspondem a: x[0] = comprimento da sépala, x[1] = largura da sépala, x[2] = comprimento da pétala, x[3] = largura da pétala.

gera o resultado mostrado na Tabela 4. É possível visualizar um par de intervalos de valores *(mínimo, máximo)* derivados dos nós internos existentes no caminho [0, 2, 3, 4, 5] para cada uma das características do conjunto de dados Iris. As características que não aparecem no caminho percorrido da raiz até o nó folha 5 foram representadas com o par *(?, ?)*. A característica que teve apenas o valor máximo derivado foi representada com o par *(?, máximo)*. Nesse caso, o símbolo *?* indica que não foi possível derivar dos nós internos do caminho percorrido um valor limite mínimo ou máximo para a característica em questão.

Tabela 4 – Resultado da realização do Passo 1 do processo de derivação dos intervalos de valores dos geradores da propriedade.

Comprimento da Sépala - x[0]	**Largura da Sépala - x[1]**	**Comprimento da Pétala - x[2]**	**Largura da Pétala - x[3]**	**Asserção de Teste**
(?, ?)	(?, ?)	(?, 4.95)	(0.80, 1.65)	1

No próximo passo da construção dos intervalos de valores é realizada a complementação dos intervalos incompletos da forma *(mínimo, ?)* ou *(?, máximo)*: intervalos que tiveram apenas o valor mínimo ou o valor máximo derivado a partir do caminho que gerou a propriedade. Na abordagem proposta, o gerador de propriedades é responsável por identificar diretamente do conjunto de dados de treinamento os valores únicos que aparecem em cada característica de cada tipo de classe (rótulo). Portanto, ao realizar o Passo

2 do processo de derivação de intervalos de valores, o gerador de propriedades substitui o símbolo *?* existente em cada par *(mínimo, ?)* pelo maior valor da característica que foi identificado no conjunto de dados de treinamento. De forma semelhante, para cada par *(?, máximo)*, o gerador de propriedades substitui o símbolo *?* pelo menor valor identificado para a característica no conjunto de dados de treinamento. Após a realização do Passo 2 são obtidos os intervalos de valores ilustrados na Tabela 5 para a propriedade deste exemplo. Para a característica `x[2]` (comprimento da pétala) foi identificado no conjunto de dados de treinamento que o valor 1.0 (menor valor) deveria substituir o símbolo *?* que existia no par *(?, 4.95)*.

Tabela 5 – Resultado da realização do Passo 2 do processo de derivação dos intervalos de valores dos geradores da propriedade.

Comprimento da Sépala - x[0]	**Largura da Sépala - x[1]**	**Comprimento da Pétala - x[2]**	**Largura da Pétala - x[3]**	**Asserção de Teste**
(?, ?)	(?, ?)	(1.00, 4.95)	(0.80, 1.65)	1

No terceiro passo do processo de derivação dos intervalos de valores, o gerador de proprieades deve realizar o tratamento das características que não apareceram em nós internos de cada caminho. No caso da propriedade gerada no exemplo, as características `x[0]` (comprimento da sépala) e `x[1]` (largura da sépala) são as únicas que não aparecem no caminho [0, 2, 3, 4, 5]. Quando uma característica não aparece em um caminho de uma árvore de decisão isso significa que durante o treinamento do modelo o algoritmo de aprendizado de máquina inferiu que essa característica não é relevante para o resultado que é obtido ao atingir o nó folha do caminho. Na abordagem proposta, entretanto, o gerador de propriedades cria um conjunto de valores com dados obtidos exclusivamente do conjunto de dados de treinamento para cada característica que não é relevante para a propriedade gerada. A Tabela 6 apresenta os conjuntos e os intervalos de valores da propriedade gerada para o exemplo após a realização do Passo 3. Os conjuntos de valores das características *comprimento da sépala* e *largura da sépala* são formados por valores identificados em amostras de dados do conjunto de treinamento que pertencem a mesma classe (possuem o mesmo rótulo) que é utilizado na asserção de teste da propriedade gerada.

Tabela 6 – Resultado da realização do Passo 3 do processo de derivação dos intervalos de valores dos geradores da propriedade.

Comprimento da Sépala - x[0]	**Largura da Sépala - x[1]**	**Comprimento da Pétala - x[2]**	**Largura da Pétala - x[3]**	**Asserção de Teste**
[5.1, 5.4, 6.3]	[2.5, 2.7, 3.4]	(1.00, 4.95)	(0.80, 1.65)	1

Caso a classe de teste a ser gerada seja baseada no critério CAD, o processo de derivação dos intervalos de valores dos geradores de dados deve encerrar após a realização

do Passo 3. No caso da propriedade gerada neste exemplo, todas as informações necessárias para definir os geradores de dados estão disponíveis na Tabela 6. De posse dessas informações, o gerador de propriedades gera o código de teste baseado em propriedades compatível com a biblioteca `Hypothesis` e com o *framework* `pytest`, conforme mostrado na Listagem 6.1.

```
@given(st.sampled_from([5.1, 5.4, 6.3]),
       st.sampled_from([2.5, 2.7, 3.4]),
       st.floats(min_value=1.00, max_value=4.95),
       st.floats(min_value=0.80, max_value=1.65))
@settings(phases=[Phase.generate], max_examples=100)
def test_1(self, feat_1, feat_2, feat_3, feat_4):
    x_test = [feat_1, feat_2, feat_3, feat_4]
    y_expected = [1]
    y_predicted = self.model.predict([x_test]).tolist()

    assert y_expected == y_predicted
```

Listagem 6.1 – Código de teste da propriedade derivada do caminho [0, 2, 3, 4, 5] utilizando o critério de teste CAD.

Para utilizar o código mostrado na Listagem 6.1 a fim de testar um modelo resultante da aplicação de algum algoritmo clássico de aprendizado supervisionado, treinado com os algoritmos de classificação disponibilizados no *framework* `scikit-learn`, basta testar o modelo usando o código gerado pela solução proposta em um ambiente Python que inclui o *framework* de teste `pytest` e a biblioteca `Hypothesis`.

Conforme mencionado, o quarto passo do processo de derivação dos intervalos de valores somente é realizado quando é utilizado o critério AVLAD. Durante o Passo 4 é realizado a restrição dos intervalos de valores *(mínimo, máximo)* para apenas valores próximos do valor mínimo ou próximos do valor máximo. Para as características que possuem os dois valores definidos, a implementação seleciona aleatoriamente o valor mínimo ou máximo para definir o novo intervalo (mais restrito). Para as características que tiveram apenas um valor definido, mínimo ou máximo, é utilizado esse valor para definir o novo intervalo. Para as características que não tiveram valores derivados de nós internos do caminho que gerou a propriedade são mantidos os conjuntos de valores criados no Passo 3.

Na Tabela 7 são apresentados os conjuntos e intervalos de valores da propriedade deste exemplo após a realização do Passo 4. As características comprimento e largura da sépala tiveram os conjuntos de valores mantidos, pois elas não tiveram valores de intervalo derivados de nós internos do caminho que gerou a propriedade. A característica comprimento da pétala teve o novo intervalo (mais restrito) definido para *(4.16, 4.95)* pois foi utilizado o valor 4.95: o valor máximo derivado de nós de decisão para definir o novo intervalo. Para a característica largura da pétala, foi definido o novo intervalo para *(0.80,*

0.97), visto que os dois valores de intervalo da característica em questão são derivados de nós de internos: o valor 0.8 (mínimo do intervalo) é selecionado aleatoriamente para definir o novo intervalo. A Listagem 6.2 apresenta o código de teste da propriedade que é gerado quando o critério AVLAD é utilizado.

Tabela 7 – Resultado da realização do Passo 4 do processo de derivação dos intervalos de valores dos geradores da propriedade.

Comprimento da Sépala - x[0]	**Largura da Sépala - x[1]**	**Comprimento da Pétala - x[2]**	**Largura da Pétala - x[3]**	**Asserção de Teste**
[5.1, 5.4, 6.3]	[2.5, 2.7, 3.4]	(4.16, 4.95)	(0.80, 0.97)	1

```
@given(st.sampled_from([5.1, 5.4, 6.3]),
       st.sampled_from([2.5, 2.7, 3.4]),
       st.floats(min_value=4.16, max_value=4.95),
       st.floats(min_value=0.80, max_value=0.97))
@settings(phases=[Phase.generate], max_examples=100)
def test_1(self, feat_1, feat_2, feat_3, feat_4):
    x_test = [feat_1, feat_2, feat_3, feat_4]
    y_expected = [1]
    y_predicted = self.model.predict([x_test]).tolist()

    assert y_expected == y_predicted
```

Listagem 6.2 – Código de teste da propriedade derivada do caminho [0, 2, 3, 4, 5] utilizando o critério de teste AVLAD.

6.6 Visão Geral da Ferramenta Proposta para Automatizar o Processo de Geração de Propriedades

Esta seção apresenta uma visão geral de uma ferramenta[1] que foi desenvolvida neste trabalho com o objetivo de automatizar o processo de geração de propriedades que foi especificado nas seções anteriores. Na Figura 17 é apresentada uma visão geral do funcionamento da ferramenta desenvolvida. Conforme pode ser observado, o `Gerador de Propriedades` é o elemento central da ferramenta que foi desenvolvida. A implementação foi realizada por meio da modularização do gerador de propriedades que foi especificado na Seção 6.4 em componentes específicos. No restante desta seção, apresenta-se uma visão geral do papel de cada um desses componentes no processo de geração de propriedades utilizando a ferramenta proposta.

Conforme mostrado na Figura 17, o processo de geração de propriedades utilizando a ferramenta proposta inicia com o testador fornecendo o `Conjunto de dados` e os

[1] O projeto da ferramenta proposta para a automatização do processo de geração de propriedades pode ser acessado em: https://github.com/ricardosm/abordagem_ferramenta_mestrado

`Parâmetros de Configuração` para a ferramenta. O `Conjunto de Dados` refere-se aos dados relacionados ao problema de classificação que será tratado com a utilização de um modelo clássico de aprendizado de máquina. Os `Parâmetros de Configuração` referem-se aos parâmetros que a ferramenta proposta disponibiliza para que o testador configure alguns pontos relacionados ao funcionamento da ferramenta e ao processo de geração de propriedades.

De posse dos `Parâmetros de Configuração` e do `Conjunto de dados`, a ferramenta proposta inicializa o componente `Gerenciador de Configuração`. O `Gerenciador de Configuração` verifica se os parâmetros de configuração fornecidos pelos testador estão corretos e encapsula as informações de configuração que serão necessárias para conduzir o processo de geração de propriedades. Na sequência, a ferramenta proposta inicializa o componente `Gerenciador de Dados de Entrada` que realiza a leitura e a criação de uma representação interna do conjunto de dados fornecido como entrada pelo testador.

Por meio da representação interna do conjunto de dados fornecido, o `Gerenciador de Dados de Entrada` fornece um conjunto de dados de treinamento para o `Algoritmo de Árvore de Decisão`. O `Algoritmo de Árvore de Decisão` disponibilizado no *framework* `scikit-learn` é utilizado para treinar um `Modelo de Árvore de Decisão` que será submetido ao `Gerador de Propriedades`. Para treinar o modelo de árvore de decisão, o `scikit-learn` disponibiliza uma implementação otimizada do algoritmo CART que foi brevemente discutido na Seção 3.2.1. Como resultado do processo de treinamento, o algoritmo CART sempre constrói uma árvore de decisão binária. O que facilita percorrer a estrutura da árvore para extrair as informações que são necessárias para o `Gerador de Propriedades`.

Após o `Gerador de Propriedades` receber o `Modelo de Árvore de Decisão`, o componente `Analisador de Requisitos de Teste` é acionado para realizar a análise do `Modelo de Árvore de Decisão`. Especificamente, o `Analisador de Requisitos de Teste` percorre a árvore de decisão para identificar: todos os caminhos da raiz à cada folha que existem na árvore; os nós internos e o nó folha que constitui cada caminho da raiz à cada folha; a característica (atributo), o valor limite, e o tipo de limite (inferior ou superior) que são avaliados em cada nó interno de cada caminho; e o tipo de classe (rótulo) que é determinado pelo nó folha de cada caminho da raiz à cada folha.

Após identificar as informações dos nós de cada caminho, o `Analisador de Requisitos de Teste` aplica os critérios CAD e AVLAD para derivar os intervalos de valores que serão utilizados para definição dos geradores de dados das propriedades. Conforme foi discutido na Seção 6.5, cada caminho identificado permite criar uma propriedade baseada no critério CAD e outra propriedade baseada no critério AVLAD. O critério de teste utilizado determina os intervalos de valores que serão derivados para os geradores de dados de cada propriedade. Dessa forma, para cada propriedade é derivado um conjunto

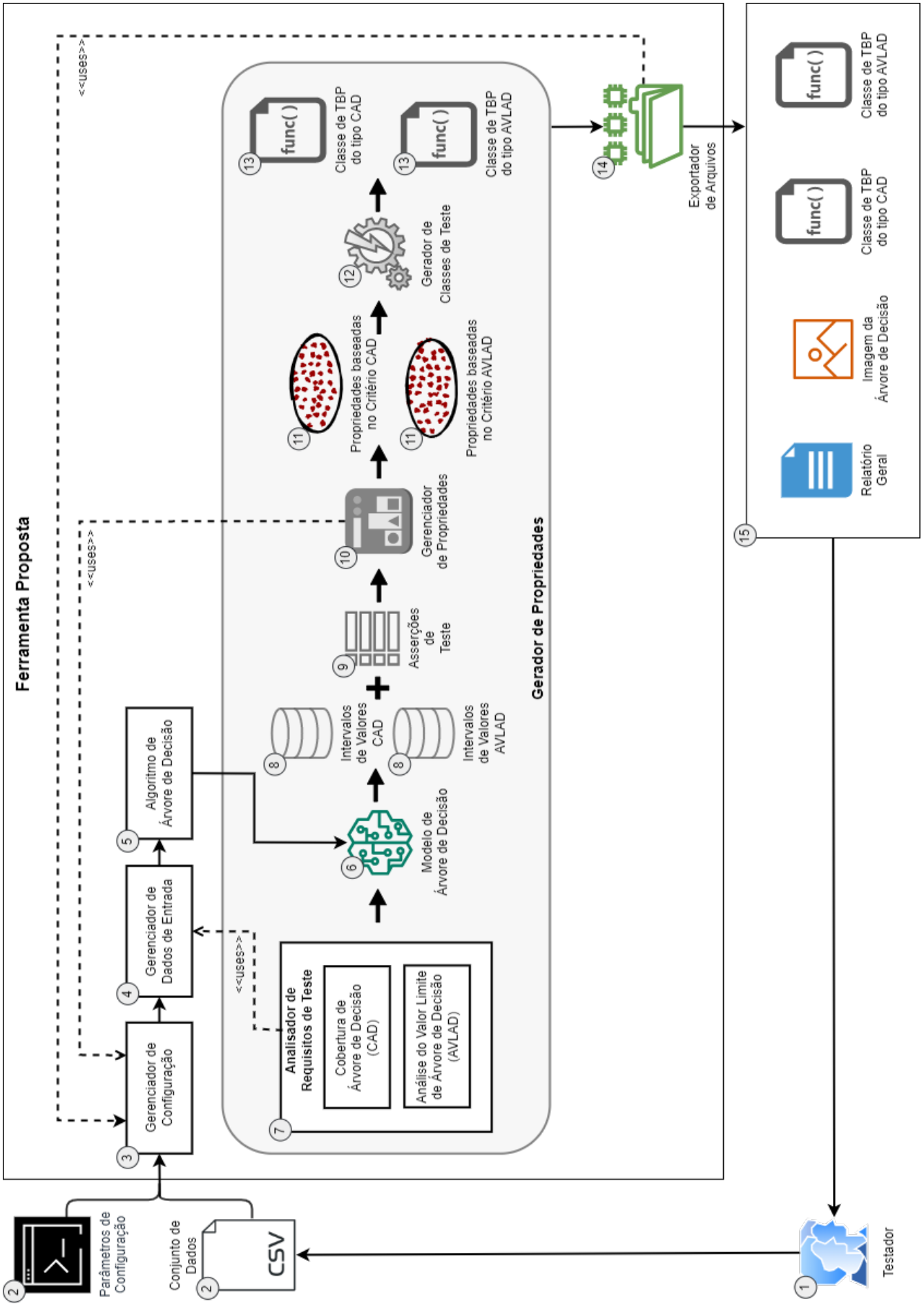

Figura 17 – Visão geral do funcionamento da ferramenta proposta para a automatização do processo de geração de propriedades.

de intervalos de valores baseados no critério CAD e outro conjunto baseado no critério AVLAD, conforme ilustrado pelo componente de número 8 da Figura 17. Caso não seja possível derivar um intervalo de valores para alguma característica de determinada propriedade, o `Analisador de Requisitos de Teste` consulta o componente `Gerenciador de Dados de Entrada` para derivar um intervalo de valores para essa característica por meio da utilização dos dados de treinamento. O componente `Analisador de Requisitos de Teste` encerra a sua execução determinando as asserções de teste que serão definidas para cada propriedade, conforme representado pelo componente de número 9 da Figura 17.

Após a definição dos intervalos de valores e asserções de teste de cada propriedade, o componente `Gerenciador de Propriedades`, representado com o número 10 na Figura 17, é inicializado para criar uma representação interna para cada propriedade de teste. Cada propriedade é representada com a definição completa dos geradores de dados que serão utilizados por cada característica, com a definição da função de teste que possui a devida asserção de teste e com a definição de configurações específicas da `Hypothesis`. Durante a criação da representação de cada propriedade, o `Gerenciador de Propriedades` pode consultar o componente `Gerenciador de Configuração` para obter valores de parâmetros de configuração que foram fornecidos pelo testador, como por exemplo, o tipo de gerador de dados da `Hypothesis` que será utilizado e a quantidade máxima de amostras de dados que serão geradas para cada propriedade. O componente `Gerenciador de Propriedades` encerra a sua execução com a criação de conjunto de propriedades baseadas no critério CAD e outro conjunto de propriedades baseadas no critério AVLAD, conforme ilustrado pelo componente de número 11 na Figura 17.

Com os conjuntos de propriedades baseados no critério CAD e AVLAD construídos, o componente `Gerador de Classes de Teste`, ilustrado com o número 12 na Figura 17, é inicializado para construir duas classes de teste: uma classe que contém todas as propriedades que foram geradas baseadas no critério CAD e outra classe que contém as propriedades que foram geradas baseadas no critério AVLAD. As duas classes de teste estão ilustradas com o número 13 na Figura 17. Ambas as classes são compatíveis com o *framework* `pytest` e com a biblioteca `Hypothesis` de teste baseado em propriedades.

A ferramenta proposta finaliza a sua execução acionando o componente `Exportador de Arquivos`. O `Exportador de Arquivos` exporta para o testador as duas classes de teste que foram geradas, um relatório geral contendo informações sobre o processo de geração das propriedades e uma imagem que representa o modelo de árvore de decisão que foi construído para derivar as propriedades. De posse das classes de teste, em um ambiente Python que possua o *framework* `pytest` e a biblioteca `Hypothesis` instalados, o testador poderá utilizar as classes para realizar o teste baseado em propriedades de modelos clássicos de aprendizado de máquina.

6.7 Considerações Finais

Neste capítulo foi realizada a apresentação da abordagem de teste baseado em propriedades que foi proposta neste trabalho para o teste de modelos clássicos de aprendizado de máquina. O capítulo iniciou delimitando o escopo de aplicação da abordagem proposta. Foi enfatizado que a abordagem proposta neste trabalho possui como objetivo possibilitar o teste de modelos clássicos de aprendizado de máquina e que o teste de modelos de aprendizado profundo não pertence ao escopo deste trabalho. Em seguida, foi discutido a motivação que impulsionou o desenvolvimento da abordagem proposta. Conforme relatado, foi identificado na literatura de teste de sistemas baseados em aprendizado de máquina que parte considerável das abordagens propostas recentemente são destinadas exclusivamente ao teste de modelos de aprendizado profundo (ou seja, modelos baseados em redes neurais profundas). Essa constatação motivou o desenvolvimento de uma abordagem de teste que adapta a técnica de teste baseado em propriedades utilizada no teste de sistemas tradicionais e os critérios de teste CAD e AVLAD que foram propostos por Santos et al. (2021) para o contexto de teste de modelos clássicos de aprendizado de máquina.

Ao prosseguir com a discussão realizada no capítulo foi apresentada uma visão geral da abordagem proposta neste trabalho. Especificamente, foram apresentados os dois processos que constituem a abordagem proposta: o processo de geração de propriedades; e o processo de execução de propriedades. Foi explicado que o elemento central do processo de geração de propriedade é o gerador de propriedades que possui a responsabilidade de aplicar os critérios de teste CAD e AVLAD sobre um modelo de árvore de decisão para possibilitar a construção de conjuntos de propriedades de teste. Foi enfatizado que essa é uma das principais diferenças existentes entre a proposta apresentada neste trabalho e o trabalho de Santos et al. (2021). Enquanto no trabalho de Santos et al. (2021) foram introduzidos os critérios de teste CAD e AVLAD que possibilitam realizar a seleção de amostras de dados de teste de um conjunto de dados pré-existentes. A abordagem proposta neste trabalho possibilita criar conjuntos de propriedades que são codificadas em classes (`arquivos.py`) de teste executáveis para realização do teste baseado em propriedades de modelos clássicos de aprendizado de máquina de maneira automatizada. Também foi explicado que o processo de execução de propriedades da abordagem proposta consiste em executar as classes de teste baseado em propriedades dos tipos CAD e AVLAD que forem construídas com a abordagem proposta contra modelos clássicos de aprendizado de máquina (especificamente, classificadores). Por fim, foi apresentada uma visão geral da ferramenta que foi desenvolvida neste trabalho como uma implementação da abordagem proposta.

7 Experimento

Neste capítulo apresenta-se o experimento que foi conduzido a fim de validar a abordagem proposta e descrita no capítulo anterior. Especificamente, o experimento foi conduzido com o propósito de avaliar a eficácia da abordagem de teste baseado em propriedades para modelos clássicos de aprendizado de máquina. No contexto do experimento, realizou-se uma comparação com o objetivo de avaliar se as amostras de dados de teste geradas com a utilização das classes de teste baseado em propriedades construídas pela ferramenta proposta (discutida na Seção 6.6) são mais eficazes do que as amostras de dados de teste que são obtidas por meio do método de validação cruzada estratificada utlizando *10-fold*, conforme brevemente discutido na Seção 3.3.1.

O restante deste capítulo está organizado conforme a seguir. Na Seção 7.1 apresenta-se a configuração do experimento realizado. Na Seção 7.2 são descritas as etapas de execução do experimento. Na Seção 7.3 são apresentados e discutidos os resultados do experimento. Na Seção 7.4 são destacadas as ameaças à validade do experimento. Por fim, na Seção 7.5 são apresentadas as considerações finais do capítulo.

7.1 Configuração do Experimento

O experimento realizado no contexto deste trabalho foi projetado com o objetivo de responder a seguinte questão de pesquisa (QP):

QP: O quão eficazes são as amostras de dados de teste geradas com a utilização da abordagem de teste baseado em propriedades para modelos clássicos de aprendizado de máquina em comparação com as amostras de dados de teste obtidas por meio da utilização de métodos de validação cruzada?

7.1.1 Escopo do Experimento

O escopo do experimento foi determinado por meio da definição de seus objetivos, que foram resumidos de acordo com o modelo Objetivo/Pergunta/Métrica, do termo em inglês, *Goal/Question/Metric (GQM)* (WOHLIN et al., 2012). Portanto, o escopo deste experimento pode ser formulado conforme a seguir:

Analisar a abordagem de teste baseado em propriedades para modelos clássicos de aprendizado de máquina

com o propósito de comparação

no que diz respeito à sua eficácia

do ponto de vista do pesquisador

no contexto de teste de sistemas baseados em aprendizado de máquina.

7.1.2 Hipóteses Formuladas

A **QP** foi formulada no formato de hipóteses para possibilitar a realização de testes estatísticos com os dados obtidos por meio da realização do experimento. Espera-se que a abordagem proposta resulte em dados de teste mais eficazes, assim a resposta esperada para a **QP** é: a abordagem de teste baseado em propriedades para modelos clássicos de aprendizado de máquina é mais eficaz do que os métodos de validação cruzada. Dessa forma, a **QP** foi transformada nas seguintes hipóteses:

- **Hipótese nula, H_0:** não existe diferença entre a eficácia obtida com a utilização da abordagem de teste baseado em propriedades para modelos clássicos de aprendizado de máquina proposta neste trabalho e a eficácia obtida com a utilização de métodos de validação cruzada.
- **Hipótese alternativa, H_1:** existe uma diferença significativa entre a eficácia obtida com a utilização da abordagem de teste baseado em propriedades para modelos clássicos de aprendizado de máquina proposta neste trabalho e a eficácia obtida com a utilização de métodos de validação cruzada.

Para avaliar essas hipóteses, a implementação da abordagem de teste baseado em propriedades (na forma da ferramenta proposta que foi discutida na Seção 6.6) foi utilizada para gerar classes de teste baseado em propriedades para realização do teste de 21 modelos de aprendizado de máquina. Especificamente, foram selecionados 21 conjuntos de dados que foram submetidos ao algoritmo K-NN para construção de 21 modelos distintos. Cada conjunto de dados também foi submetido à ferramenta proposta para geração de duas classes de teste baseado em propriedades: uma classe do tipo CAD e outra do tipo AVLAD, ambas usadas para testar cada modelo K-NN. Em seguida, as classes de teste geradas com cada conjunto de dados foram executadas contra o modelo K-NN que foi construído com o mesmo conjunto de dados que originou cada par de classes de teste. Por fim, os resultados da realização do teste baseado em propriedades de cada modelo K-NN foram comparados com os resultados da realização da validação cruzada estratificada por *10-fold* de modelos K-NN que também foram construídos com os mesmos conjuntos de dados.

Para comparar os resultados da execução das classes de teste baseado em propriedades geradas com a ferramenta proposta com os resultados da realização da validação cruzada estratificada por *10-fold* de cada modelo K-NN, foram utilizadas as seguintes

métricas de avaliação de desempenho: acurácia, F_1, revocação e precisão. Conforme apresentado na Seção 3.3.2, essas quatro métricas são amplamente utilizadas na avaliação do desempenho de classificadores. Tais métricas possuem em comum o fato de serem calculadas por meio da matriz de confusão do classificador em teste.

A fim de responder a **QP**, a eficácia foi a variável dependente investigada. A eficácia foi medida em termos das métricas de avaliação de desempenho de classificadores mencionadas anteriormente. No contexto deste experimento, foi considerado como a abordagem ou método de teste mais eficaz, aquele que possibilitou a seleção ou a geração de dados de teste (amostras de teste) que resultaram em uma maior diminuição do desempenho de predição do modelo testado. A eficácia da abordagem de teste baseado em propriedades proposta neste trabalho e a eficácia do método de validação cruzada estratificada por *10-fold*, foram avaliadas em termos da capacidade que ambos os métodos apresentaram para possibilitar a geração ou a seleção de amostras de dados de teste que resultaram em menores valores de acurácia, F_1, revocação e precisão. Na seção seguinte é detalhado como o experimento foi conduzido.

7.2 Execução do Experimento

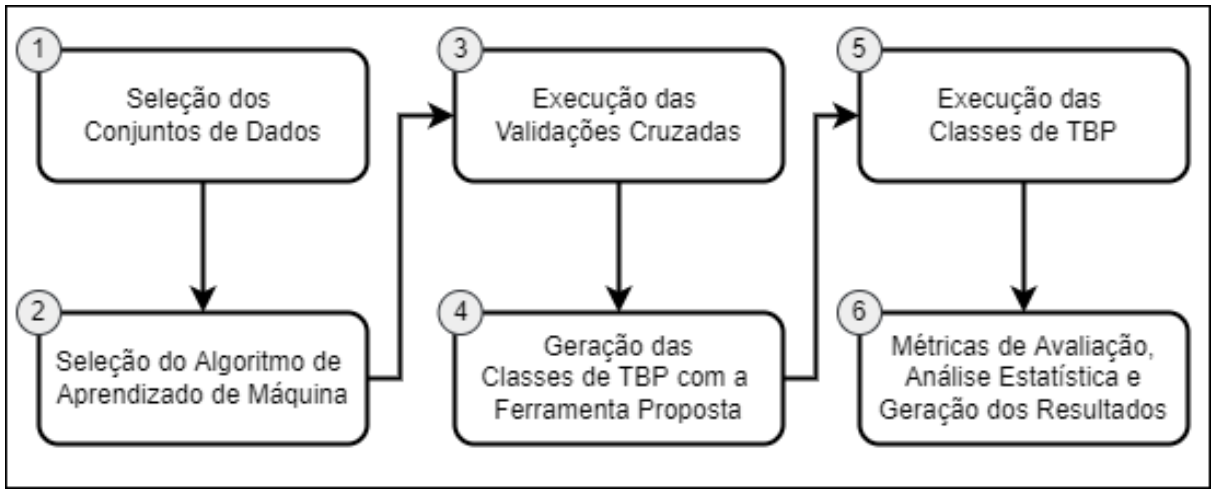

Figura 18 – Etapas envolvidas na execução do experimento.

Conforme apresentado na Figura 18, a execução do experimento foi realizada em seis etapas distintas: *(i)* seleção dos conjunto de dados; *(ii)* seleção do algoritmo de aprendizado de máquina utilizado para construir os modelos que foram testados e avaliados durante o experimento; *(iii)* execução das validações cruzadas; *(iv)* geração das classes de teste baseado em propriedades por meio da utilização da ferramenta proposta; *(v)* execução das classes de teste baseado em propriedades (ou seja, o teste baseado em propriedades dos modelos); e, por fim, *(vi)* a execução das métricas de avaliação, a análise estatística e avaliação dos resultados. Cada uma dessas etapas é descrita em detalhes nas próximas subseções.

7.2.1 Seleção dos Conjuntos de Dados

A execução do experimento iniciou com a coleta dos conjuntos de dados que foram necessários para realizar o treinamento dos modelos de aprendizado de máquina que foram testados e avaliados durante o experimento. Além disso, os mesmos conjuntos de dados também foram fornecidos para a ferramenta proposta, que internamente realizou o treinamento dos modelos de árvore de decisão utilizados pela própria ferramenta no processo de geração de propriedades, possibilitando assim a geração das classes de teste baseado em propriedades.

Tabela 8 – Conjuntos de dados selecionados e utilizados no experimento.

Conjunto de Dados	**Amostras**	**Atributos**	**Classes**	**Amostras por Classe**
Bank Marketing	45211	17	2	[39922, 5289]
Banknote Authentication	1372	5	2	[762, 610]
Blood Transfusion	748	5	2	[570, 178]
Breast Cancer Wisconsin	569	31	2	[212, 357]
Cleveland Heart Disease	297	14	2	[160, 137]
Glass Identification	214	10	6	[70, 76, 17, 13, 9, 29]
Haberman's Survival	306	4	2	[225, 81]
Horse Racing Tipster Bets	38248	10	2	[30565, 7683]
Indian Liver Patient	579	11	2	[414, 165]
Ionosphere	351	35	2	[126, 225]
Iris	150	5	3	[50, 50, 50]
Oil Spill	937	50	2	[896, 41]
Page Blocks Classification	5473	11	5	[4913, 329, 28, 88, 115]
Phoneme	5404	6	2	[3818, 1586]
Pima Indians Diabetes	768	9	2	[500, 268]
Seeds	210	8	3	[70, 70, 70]
Sonar, Mines vs. Rocks	208	61	2	[111, 97]
Students Knowledge Levels	403	6	4	[50, 129, 122, 102]
Vertebral Column	310	7	3	[100, 60, 150]
Wine Recognition	178	14	3	[59, 71, 48]
Woods Mammography	11183	7	2	[10923, 260]

Para seleção dos conjunto de dados foi utilizado o *UCI Machine Learning Repository*[1], que disponibiliza uma coleção de conjuntos de dados amplamente utilizados pela comunidade de aprendizado de máquina para realização de análise empírica de algoritmos de aprendizado de máquina. Como a abordagem proposta é destinada ao teste de modelos de aprendizado de máquina clássicos que solucionam problemas de classificação, ou seja, é destinada ao teste de classificadores, todos os conjuntos de dados selecionados são aplicáveis ao treinamento de classificadores. Para evitar a necessidade de realização de uma complexa etapa de pré-processamento de dados, foram selecionados do *UCI Machine Learning Repository* conjuntos de dados considerados simples e concisos.

[1] https://archive.ics.uci.edu/ml/index.php

A Tabela 8 sumariza as informações referentes aos 21 conjuntos de dados selecionados para realização do experimento. Conforme descrito na tabela, os conjuntos de dados selecionados apresentam uma variação considerável na quantidade de amostras de dados e na quantidade de características que constituem cada conjunto. Por exemplo, o conjunto de dados `Seeds` possui 210 amostras de dados, cada amostra do conjunto possui 8 atributos, dos quais 7 atributos referem-se a características específicas das amostras e 1 atributo é o rótulo que determina a classe que cada amostra do conjunto de dados pertence. Além disso, o conjunto de dados `Seeds` possui 3 tipos de classe distintas e cada tipo de classe é composto por 70 amostras de dados, ou seja, esse é um conjunto de dados balanceado. Cada um dos conjuntos de dados selecionados para realização do experimento é brevemente descrito a seguir.

1. `Bank Marketing:` este conjunto de dados possui 45211 registros relacionados a uma campanha de marketing direto de uma instituição bancária portuguesa. O conjunto de dados é formado por 17 atributos. Os dezesseis primeiros atributos são formados por valores numéricos contendo informações de clientes do banco. O último atributo é um rótulo que permite classificar se determinado cliente vai realizar um investimento de longo prazo com o banco. O conjunto de dados possui 5289 registros de clientes que realizaram um investimento com o banco e 39922 registros de clientes que não realizaram um investimento.

2. `Banknote Authentication:` este conjunto de dados possui 1372 registros de especificações de fotografias que foram tiradas de notas de papel moeda. Foi utilizada a ferramenta `Wavelet` para extrair as características de cada uma das imagens geradas. O conjunto de dados é formado por 5 atributos. Os quatro primeiros atributos do conjunto são valores numéricos que representam as especificações de cada imagem e o último atributo é um rótulo que permite classificar os registros do conjunto em dois subgrupos. Um subgrupo é formado por 762 registros que representam especificações de notas falsificadas e o outro subgrupo é formado por 610 registros que representam especificações de notas verdadeiras.

3. `Blood Transfusion:` este conjunto de dados possui registros de doadores de sangue de um centro de serviços de transfusão de sangue existente na cidade de Hsin-Chu, em Taiwan. O conjunto de dados é formado por 5 atributos e possui registros de 748 doadores. Os quatro primeiros atributos do conjunto são formados por valores numéricos que fornecem informações sobre as doações realizadas por cada doador e o último atributo é um rótulo que divide o grupo de doadores em dois subgrupos. Um subgrupo é formado por 178 registros de doadores que doaram sangue em Março de 2007 e o outro subgrupo é formado por 570 registros de doadores que não doaram sangue nesse mês.

4. `Breast Cancer Wisconsin:` este conjunto de dados possui 569 registros de dados obtidos de um banco de dados de diagnóstico de câncer de mama do estado de Wisconsin, EUA. O conjunto de dados possui 31 atributos. Os primeiros trinta atributos representam valores de características que foram calculadas por meio da utilização de imagens digitalizadas de massas mamárias. O último atributo é um rótulo que permite classificar se as informações existentes em cada registro resultaram em um diagnóstico positivo ou negativo de câncer de mama. O conjunto de dados possui 212 registros que representam um tumor maligno e 357 registros que representam um tumor benigno.

5. `Cleveland Heart Disease:` este conjunto de dados possui 297 registros de dados obtidos do banco de dados Cleveland. O conjunto de dados possui 14 atributos, treze atributos representam valores de características fisiológicas que podem indicar se um paciente médico possui alguma cardiopatia e o último atributo é um rótulo que permite classificar se as informações existentes em cada registro são provenientes de um paciente com alguma cardiopatia ou de um paciente saudável. O conjunto de dados possui 160 registros de pacientes saudáveis e 137 registros de pacientes diagnosticados com alguma cardiopatia.

6. `Glass Identification:` este conjunto de dados possui 214 registros de propriedades físicas encontradas em 6 tipos distintos de vidro. Os dados foram obtidos de uma base de dados utilizada por um serviço de ciência forense dos EUA. O estudo da classificação de diferentes tipos de vidro é importante para ajudar na identificação de possíveis provas deixadas em cenas de crimes. O conjunto de dados é formado por 10 atributos, sendo que, nove atributos são formados por valores numéricos relacionados a propriedades físicas dos materiais encontrados em vidros e o último atributo é um rótulo que permite classificar cada registro como sendo de um tipo específico de vidro.

7. `Haberman's Survival:` este conjunto de dados possui 306 registros médicos de pacientes que foram submetidos a cirurgia para tratamento de câncer de mama. Os dados foram obtidos por meio de um estudo conduzido entre 1958 e 1970 no Billings Hospital da Universidade de Chicago. O conjunto de dados possui 4 atributos, três atributos são formados por valores numéricos relacionados à condição clínica do paciente e o último atributo é um rótulo que permite classificar se o paciente sobreviveu por um período de 5 anos ou mais após a cirurgia. O conjunto de dados possui 225 registros de pacientes que conseguiram sobreviver por um período maior ou igual a cinco anos e 81 registros de pacientes que faleceram em um período menor que cinco anos após a cirurgia.

8. `Horse Racing Tipster Bets:` este conjunto de dados possui 38248 registros de

dicas de apostas em corridas de cavalo. O objetivo é catalogar informações relacionadas às dicas fornecidas por apostadores que conseguem identificar os cavalos ganhadores de determinadas corridas. As informações catalogadas podem ser utilizadas na realização de predições sobre quais serão os próximos cavalos vencedores. O conjunto de dados possui 10 atributos, nove atributos são formados por valores numéricos relacionados às dicas fornecidas por apostadores e o último atributo é um rótulo que permite classificar se uma determinada dica resultou em uma aposta vencedora ou não. O conjunto de dados possui 7683 registros de dicas que resultaram em apostas vencedoras e 30565 registros de dicas não vencedoras.

9. `Indian Liver Patient:` este conjunto de dados possui registros médicos de 579 pacientes residentes no nordeste do estado de Andhra Pradesh, na Índia. O conjunto de dados possui 11 atributos, sendo que, dez atributos são formados por valores numéricos relacionados a indicadores de saúde de cada paciente e o último atributo é um rótulo que permite classificar cada registro como pertencente a um paciente hepático ou um paciente não hepático. O conjunto de dados total está dividido entre 414 registros de pacientes hepáticos e 165 registros de pacientes não hepáticos.

10. `Ionosphere:` este conjunto de dados possui 351 registros de retornos de sinais de radar que foram emitidos na Ionosfera. Por meio dos retornos de sinais é possível detectar a presença de objetos na Ionosfera. O conjunto de dados possui 35 atributos, sendo que, trinta e quatro atributos são formados por valores numéricos que representam características dos retornos de sinais de radar que foram captados e o último atributo é um rótulo que permite classificar se um retorno de sinal detectou ou não a presença de algum objeto. O conjunto de dados possui 225 registros de retornos de sinais que detectaram objetos e 126 registros que não detectaram.

11. `Iris:` este conjunto de dados possui 150 registros de características encontradas em três tipos de flor Iris. O conjunto de dados possui 5 atributos, quatro atributos são formados por valores numéricos que representam características das flores Íris. O último atributo é um rótulo que permite classificar cada registro de flor como sendo de um tipo específico de Íris. O conjunto de dados possui 50 registros de cada um dos três tipos de Íris.

12. `Oil Spill:` este conjunto de dados possui 937 registros de características identificadas em imagens de satélite do oceano. Alguns registros correspondem a imagens que contêm derramamento de óleo no oceano. O conjunto de dados possui 50 atributos, quarenta e nove atributos são formados por valores numéricos que representam características das imagens e o último atributo é um rótulo que permite classificar se o registro corresponde ou não a uma imagem do oceano que possui derramamento de óleo. O conjunto de dados possui 41 registros que correspondem a imagens que

possuem derramamento de óleo e 896 registros que correspondem a imagens que não possuem derramamento.

13. `Page Blocks Classification:` este conjunto de dados possui 5473 registros de blocos de objetos encontrados no layout de páginas de documentos. Todos os 5473 registros foram obtidos de 54 documentos distintos por meio de um processo de segmentação. O conjunto de dados é formado por 11 atributos, dez atributos são valores numéricos correspondentes ao formato dos blocos identificados no layout das páginas e o último atributo é um rótulo que permite classificar cada registro como sendo de um tipo de bloco específico. O conjunto de dados total está dividido em cinco tipos distintos de blocos.

14. `Phoneme:` este conjunto de dados possui 5404 registros de dados que representam a pronúncia de vogais. O objetivo deste conjunto de dados é possibilitar a distinção da pronúncia de vogais nasais e orais. O conjunto de dados possui 6 atributos, cinco atributos são formados por valores numéricos que representam características específicas da pronúncia de vogais e o último atributo é um rótulo que permite classificar cada registro como sendo de uma vogal nasal ou vogal oral. O conjunto de dados possui 3818 registros de vogais nasais e 1586 registros de vogais orais.

15. `Pima Indians Diabetes:` este conjunto de dados possui 768 registros relacionados à condição clínica de pacientes mulheres descendentes dos índios Pima. O objetivo do conjunto de dados é possibilitar a realização de diagnóstico de diabetes por meio das informações coletadas de exames médicos realizados pelas pacientes. O conjunto de dados possui 9 atributos, oito atributos são formados por valores numéricos que representam indicadores de saúde das pacientes e o último atributo é um rótulo que permite classificar com base nos indicadores de cada paciente se ela possui diabetes. O conjunto de dados possui 268 registros de pacientes que possuem diabetes e 500 registros de pacientes que não possuem.

16. `Seeds:` este conjunto de dados possui 210 registros de grãos que pertencem a três diferentes tipos de trigo. O conjunto de dados é formado por 8 atributos, sete atributos são formados por valores numéricos que correspondem a características que foram medidas dos grãos e o último atributo é um rótulo que permite classificar cada registro como pertencente a uma classe específica de tipo de trigo. O conjunto de dados total está dividido de forma balanceada com cada uma das três classes de trigo possuindo 70 registros.

17. `Sonar, Mines vs. Rocks:` este conjunto de dados possui 208 registros de padrões de sinais de sonar que foram refletidos em rochas e em cilindros metálicos sob vários ângulos e condições. O conjunto de dados é formado por 61 atributos. Sessenta atributos correspondem a valores no intervalo de 0,0 até 1,0. Cada valor representa

a quantidade de energia existente dentro de uma determinada faixa de frequência. O último atributo é um rótulo que classifica cada registro como um sinal que foi refletido por uma rocha ou por um cilindro metálico. O conjunto de dados possui 111 sinais refletidos por cilindros metálicos e 97 refletidos por rochas.

18. `Students Knowledge Levels:` este conjunto de dados possui 403 registros de notas em formato de conceitos de alunos da disciplina Máquinas Elétricas de Corrente Contínua ministrada pelo Departamento de Educação Elétrica da Universidade de Gazi da cidade de Ancara na Turquia. O conjunto de dados possui 6 atributos, cinco atributos são valores numéricos relacionados ao tempo de estudo que cada aluno dedicou para diferentes atividades e o último atributo é um rótulo que determina a nota em forma de conceito que o aluno obteve. O conjunto de dados total está dividido em quatro tipos de notas em forma de conceito.

19. `Vertebral Column:` este conjunto de dados possui 310 registros de pacientes ortopédicos. Cada registro define o estado clínico de um paciente, sendo que, três estados clínicos são possíveis: normal, com hérnia de disco e com espondilolistese. O conjunto de dados foi construído por um médico durante uma residência médica em um hospital da cidade de Lyon na França. O conjunto de dados é formado por 7 atributos, sendo que, seis atributos correspondem a valores numéricos relacionados a características biomecânicas dos pacientes e o último atributo é um rótulo que permite classificar cada registro como pertencente a um paciente que possui um estado clínico específico. O conjunto de dados possui 100 registros de pacientes com estado clínico normal, 60 pacientes que possuem hérnia de disco e 150 pacientes que possuem espondilolistese.

20. `Wine Recognition:` este conjunto de dados possui 178 registros de valores de características encontradas em três tipos de vinhos produzidos em uma região da Itália. Os dados foram obtidos por meio da realização de uma análise química que possibilitou identificar 13 constituintes determinantes para identificação dos diferentes tipos de vinhos. O conjunto de dados possui 14 atributos, sendo que, treze atributos são valores numéricos que determinam as características dos vinhos e o último atributo é um rótulo que classifica cada registro como sendo de um tipo específico de vinho.

21. `Woods Mammography:` este conjunto de dados possui 11183 registros de dados obtidos de mamografias realizadas em pacientes médicos com o objetivo de detectar a presença de câncer de mama por meio da identificação de microcalcificações mamárias. O conjunto de dados possui 7 atributos, seis atributos são formados por valores numéricos que representam características identificadas nos exames realizados e o último atributo é um rótulo que permite classificar se cada registro de exame resultou na identificação de microcalcificações mamárias. O conjunto de dados possui

260 registros de exames que identificaram microcalcificações e 10923 registros que não identificaram.

7.2.2 Seleção do Algoritmo de Aprendizado de Máquina

Após a seleção dos conjuntos de dados foi necessário selecionar o algoritmo de aprendizado de máquina utilizado para realizar o treinamento dos modelos que foram testados e avaliados durante o experimento. Para tal, foi selecionado o algoritmo K-NN (discutido na Seção 3.2.2): no contexto do experimento, utilizou-se a implementação do algoritmo disponível no *framework* `scikit-learn`.

Como o algoritmo K-NN permite que o desenvolvedor faça a definição do valor do hiperparâmetro K que determina o tamanho da vizinhança que será utilizada pelo modelo resultante. No contexto do experimento, decidiu-se pela utilização do valor de $K = 5$ para definir o tamanho da vizinhança de todos os modelos treinados com o algoritmo: ou seja, optou-se pela utilização do mesmo valor de K adotado como padrão pela implementação do algoritmo K-NN disponível no *framework* `scikit-learn`.

7.2.3 Execução das Validações Cruzadas Estratificadas dos Modelos 5-NN

Esta subseção descreve como o método de validação cruzada foi utilizado para avaliar o desempenho dos modelos 5-NN construídos com cada um dos conjuntos de dados selecionados. Foi utilizada uma variação do método de validação cruzada por *10-fold* conhecida como validação cruzada estratificada por *10-Fold* (os dois métodos foram discutidos na Seção 3.3.1). O diferencial do método de validação cruzada estratificada é que ele permite realizar a divisão do conjunto de dados original em conjuntos de treino e teste que possuem a mesma proporção de amostras de cada tipo de classe que existe no conjunto de dados original. Possibilitando obter uma estimativa de desempenho mais precisa para os modelos que forem treinados com conjuntos de dados desbalanceados.

Para realizar as validações cruzadas foi implementado um *script* Python que utilizou a biblioteca `pandas`[2] para carregar todos os conjuntos de dados selecionados. Neste mesmo *script* foram importados o algoritmo K-NN e as funções relacionadas à validação cruzada estratificada por *10-fold* que são disponibilizadas no `scikit-learn`. Então, cada conjunto de dados carregado com o `pandas` foi fornecido para a função do `scikit-learn` responsável por realizar a validação cruzada estratificada dos modelos 5-NN que seriam construídos com cada conjunto de dados. Por fim, o *script* implementado foi executado em um ambiente Python disponibilizado pelo Google Colab[3].

[2] https://pandas.pydata.org/
[3] https://colab.research.google.com

Na Figura 19 é ilustrado como as validações cruzadas estratificadas por *10-fold* foram realizadas ao executar o *script* implementado mencionado anteriormente. O processo de divisão do conjunto de dados original em conjunto de treino e teste, o treinamento do modelo 5-NN utilizando o algoritmo KNN, a realização da predição e a coleta dos resultados foram repetidas 10 vezes (uma para cada *fold* de teste) para cada um dos 21 conjuntos de dados existentes na amostra do experimento. Essencialmente, para cada conjunto de dados foram treinados 10 modelos 5-NN que foram avaliados (testados) com conjuntos de teste distintos. Por fim, os resultados das predições realizadas pelos 10 modelos que foram treinados para cada um dos 21 conjuntos de dados foram armazenados para posterior análise dos resultados.

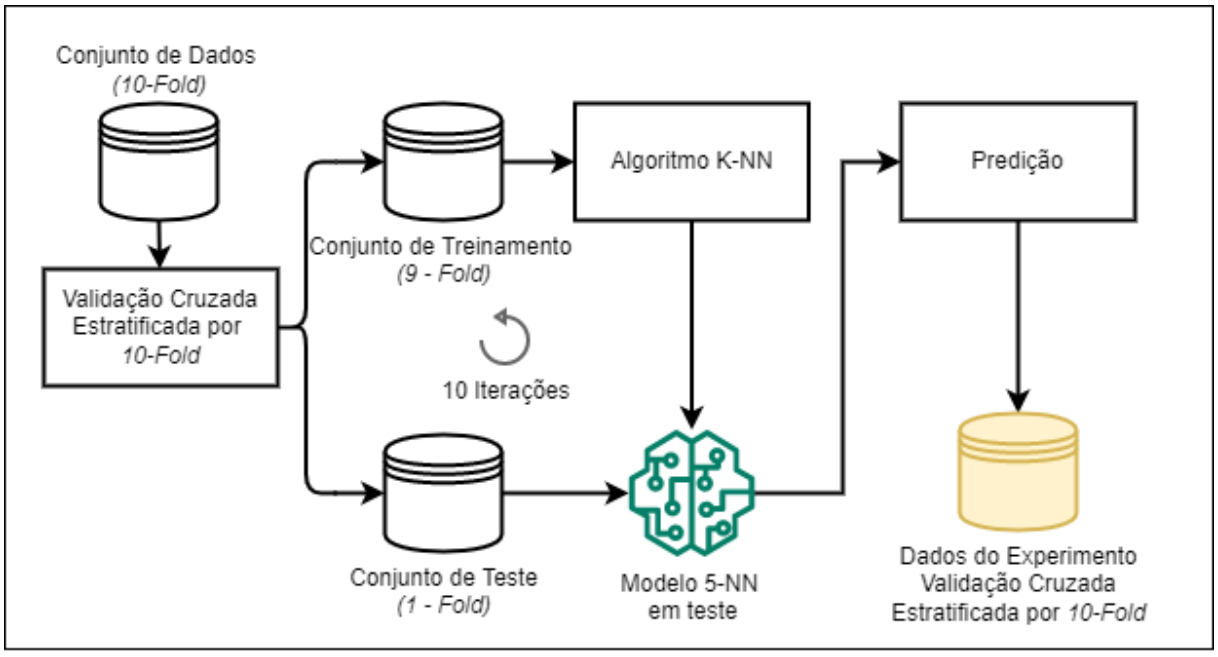

Figura 19 – Execução das validações cruzadas estratificadas dos modelos 5-NN.

7.2.4 Geração das Classes de Teste Baseado em Propriedades Utilizando a Ferramenta Proposta

Como o objetivo do experimento é investigar a eficácia das classes de teste construídas com a abordagem de teste baseado em propriedades. Foi necessário utilizar a ferramenta desenvolvida neste trabalho (que implementa a abordagem proposta) para gerar as classes de teste baseado em propriedades que foram utilizadas durante a realização do experimento. Um visão geral de como a ferramenta foi utilizada é apresentada na Figura 20. Conforme pode ser observado, para cada um dos 21 conjuntos de dados utilizados no experimento, foram criadas duas classes de teste baseado em propriedades, sendo uma baseada no critério de teste CAD e outra no critério AVLAD. Além das classes de teste, a ferramenta proposta também construiu um modelo 5-NN utilizando o conjunto de dados que foi fornecido como entrada.

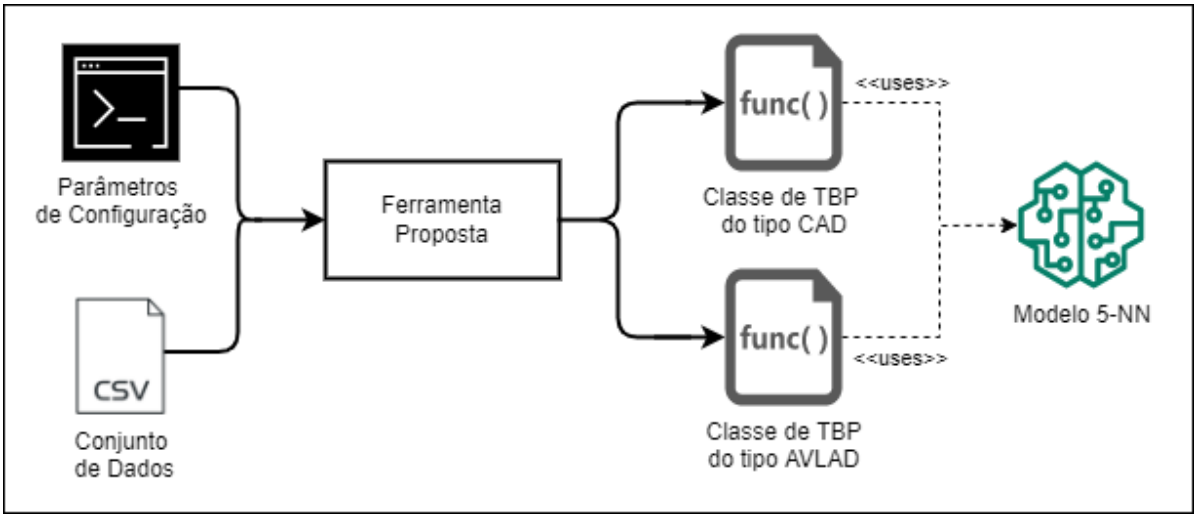

Figura 20 – Geração das classes de teste baseado em propriedades utilizando a ferramenta proposta.

É importante enfatizar que a finalidade da ferramenta proposta é gerar classes de teste baseado em propriedades para realização do teste de modelos, e não a criação de modelos de aprendizado de máquina. Entretanto, a ferramenta proposta possibilita a construção de determinados tipos de modelos utilizando os algoritmos de aprendizado de máquina que são disponibilizados no `scikit-learn`. Essa funcionalidade foi implementada a fim de facilitar a realização de atividades de experimentação utilizando as classes de teste construídas com a ferramenta. A ferramenta pode ser configurada para, além de gerar as classes de teste, utilizar o algoritmo K-NN para realizar o treinamento de um modelo 5-NN utilizando 90% das amostras de dados existentes no conjunto de dados que foi fornecido como entrada. Por padrão, a ferramenta seleciona de forma estratificada 90% das amostras de dados existentes no conjunto de dados original para garantir que os modelos 5-NN construídos nessa etapa da execução do experimento sejam expostos a mesma quantidade de dados de treinamento que os modelos que foram avaliados na etapa de execução das validações cruzadas.

Na Tabela 9 são sumarizadas as informações das classes de teste baseado em propriedades que foram geradas pela ferramenta proposta. A coluna `Propriedades` descreve a quantidade de propriedades que foram geradas pela ferramenta proposta para cada tipo de classe de teste (CAD ou AVLAD) para realizar o teste do modelo 5-NN construído com cada conjunto de dados. Por exemplo, para testar o modelo 5-NN construído com o conjunto de dados `Phoneme` foi gerada uma classe de teste contendo 521 propriedades baseadas no critério CAD e outra classe contendo 521 propriedades baseadas no critério AVLAD. A coluna `Amostras por Propriedade` descreve a configuração (*decorator* `@settings(max_examples=100)`) que cada propriedade recebeu para orientar a biblioteca `Hypothesis` sobre a quantidade máxima de amostras de dados de teste que deveriam ser geradas quando a classe de teste fosse executada. Por exemplo, quando a classe de teste do tipo CAD que foi gerada para testar o modelo 5-NN construído com o conjunto

de dados `Phoneme` foi executada, a biblioteca `Hypothesis` gerou 100 amostras de dados de teste para cada uma das 521 propriedades existentes nessa classe de teste. O mesmo aconteceu quando a classe de teste do tipo AVLAD foi executada.

Tabela 9 – Quantidade de propriedades existentes e quantidade de amostras de dados de teste geradas com cada classe de teste baseado em propriedades.

Conjunto de Dados	**Propriedades**	**Amostras por Propriedade**	**Amostras Geradas**	**Amostras Únicas Geradas**	
	CAD / AVLAD	**CAD / AVLAD**	**CAD / AVLAD**	**CAD**	**AVLAD**
Bank Marketing	3679	100	367900	364747	364422
Banknote Authentication	27	100	2700	2568	2535
Blood Transfusion	198	100	19800	19717	19695
Breast Cancer Wisconsin	22	100	2200	2142	2143
Cleveland Heart Disease	57	100	5700	5641	5652
Glass Identification	50	100	5000	4889	4891
Haberman's Survival	103	100	10300	10277	10279
Horse Racing Tipster Bets	7389	100	738900	734365	734202
Indian Liver Patient	111	100	11100	10968	10976
Ionosphere	23	100	2300	2278	2279
Iris	9	100	900	877	874
Oil Spill	35	100	3500	3458	3456
Page Blocks Classification	165	100	16500	16137	16107
Phoneme	521	100	52100	51667	51647
Pima Indians Diabetes	129	100	12900	12623	12622
Seeds	16	100	1600	1567	1564
Sonar, Mines vs. Rocks	22	100	2200	2171	2174
Students Knowledge Levels	31	100	3100	3040	3023
Vertebral Column	45	100	4500	4396	4390
Wine Recognition	12	100	1200	1162	1164
Woods Mammography	161	100	16100	15912	15914
Estatísticas Descritivas					
Máx	7389,00	100,00	738900,00	734365,00	734202,00
Mín	9,00	100,00	900,00	877,00	874,00
Média	609,76	100,00	60976,19	60504,86	60476,62
Mediana	50,00	100,00	5000,00	4889,00	4891,00
Desvio Padrão	1742,70	0,00	174270,46	173137,77	173078,78
MAD	937,95	0,00	93795,01	93152,60	93111,50

7.2.5 Execução das Classes de Teste Baseado em Propriedades Contra os Modelos 5-NN

A seguir descreve-se a etapa da execução do experimento no qual as classes de teste baseado em propriedades geradas por meio da ferramenta foram executadas contra os modelos 5-NN. Conforme mostrado na Figura 21, em um ambiente Python com o *framework* `pytest` e com a biblioteca `Hypothesis` instalados foram executadas duas classes de teste baseado em propriedades (uma classe do tipo CAT e outra do tipo AVLAD) contra cada um dos 21 modelos 5-NN que foram construídos. Especificamente, cada amostra de dados de teste gerada pela biblioteca `Hypothesis` durante a execução de cada classe de teste baseado em propriedades foi submetida ao modelo 5-NN em teste para realização da predição do rótulo da amostra em questão. Por fim, cada amostra gerada (`x_teste`),

cada resultado de predição (`y_predicted`) e cada resultado esperado (`y_expected`) foram coletados e armazenados para posterior análise dos resultados.

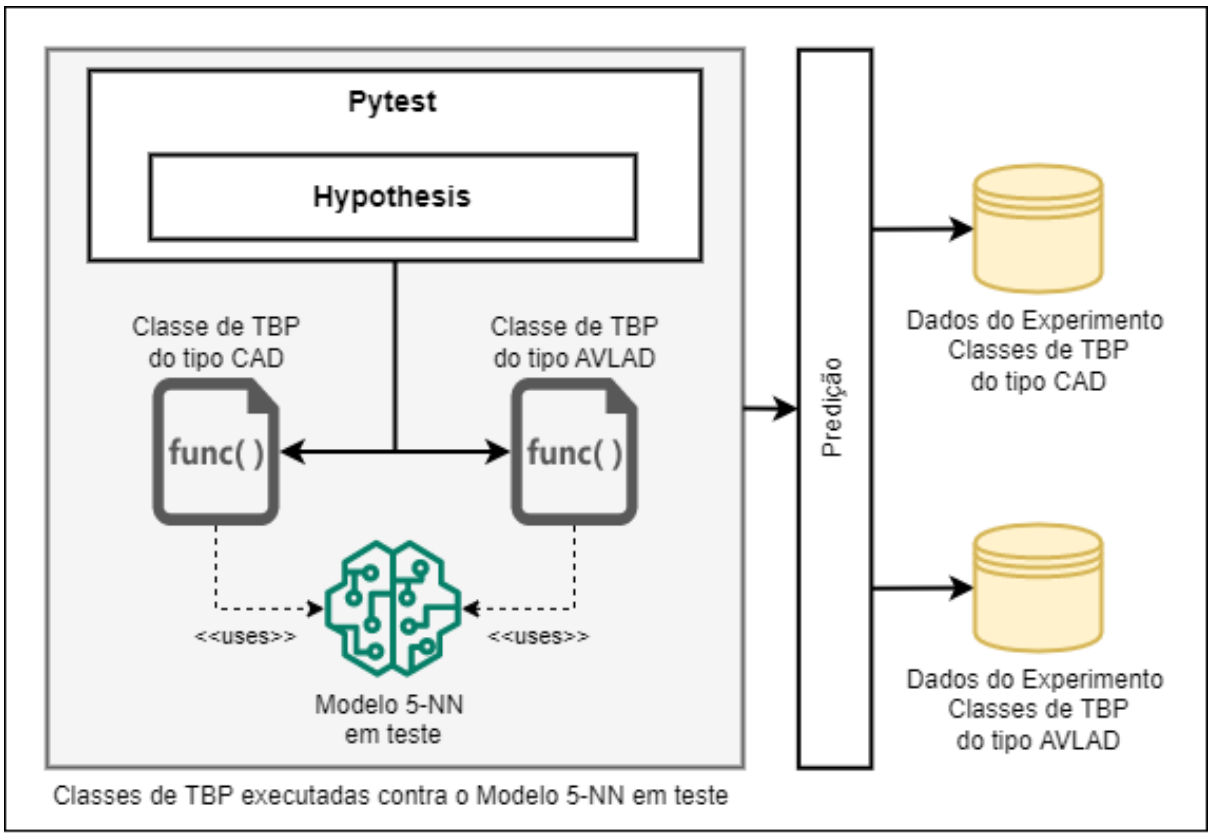

Figura 21 – Execução das classes de teste baseado em propriedades contra os modelos 5-NN.

A coluna `Amostras Geradas` da Tabela 9 lista a quantidade total de amostras de dados de teste que foram geradas com a execução das classes de teste baseado em propriedades contra os modelos 5-NN que foram construídos com cada conjunto de dados e a coluna `Amostras Únicas Geradas` descreve a quantidade de amostras únicas que foram geradas. Por exemplo, ao executar a classe de teste baseado em propriedades do tipo CAD que foi criada para testar o modelo 5-NN construído com o conjunto de dados `Phoneme`, foram geradas 52100 amostras de dados de teste, desse total, 51667 eram amostras únicas. De maneira semelhante, a classe de teste do tipo AVLAD também possibilitou a geração de 52100 amostras de teste, sendo, 51647 amostras únicas. A Tabela 9 também apresenta algumas estatísticas descritivas relacionadas às amostras de dados de teste que foram geradas com a execução das classes de teste baseado em propriedades criadas com a ferramenta proposta. Por exemplo, a quantidade mínima de amostras de dados de teste que foram geradas com a execução de uma classe de teste foi de 900 amostras e a quantidade máxima foi de 738900 amostras.

7.2.6 Execução das Métricas de Avaliação de Desempenho, Análise Estatística e Geração dos Resultados

Esta subseção descreve a etapa do experimento em que foi realizada a execução das métricas de avaliação de desempenho, a análise estatística e a geração dos resultados do experimento. Após a execução das classes de teste baseado em propriedades criadas com a ferramenta proposta e a execução das validações cruzadas estratificadas, foram coletados os dados e os resultados de teste gerados em cada uma dessas etapas. Conforme pode ser observado na Figura 22, esses dados foram submetidos às métricas de avaliação de desempenho utilizadas no experimento. Especificamente, as métricas acurácia, F_1, revocação e precisão foram utilizadas para avaliar, respectivamente, os dados obtidos com a execução das validações cruzadas estratificadas, os dados obtidos com a execução das classes de teste baseado em propriedades do tipo CAD e os dados obtidos com a execução das classes de teste baseado em propriedades do tipo AVLAD. Nesta etapa da execução do experimento também foi realizada uma análise com a finalidade de obter estatísticas descritivas relacionadas aos resultados das métricas de avaliação de desempenho utilizadas. Adicionalmente, foram realizados dois testes estatísticos com a finalidade de possibilitar uma melhor interpretação dos resultados do experimento.

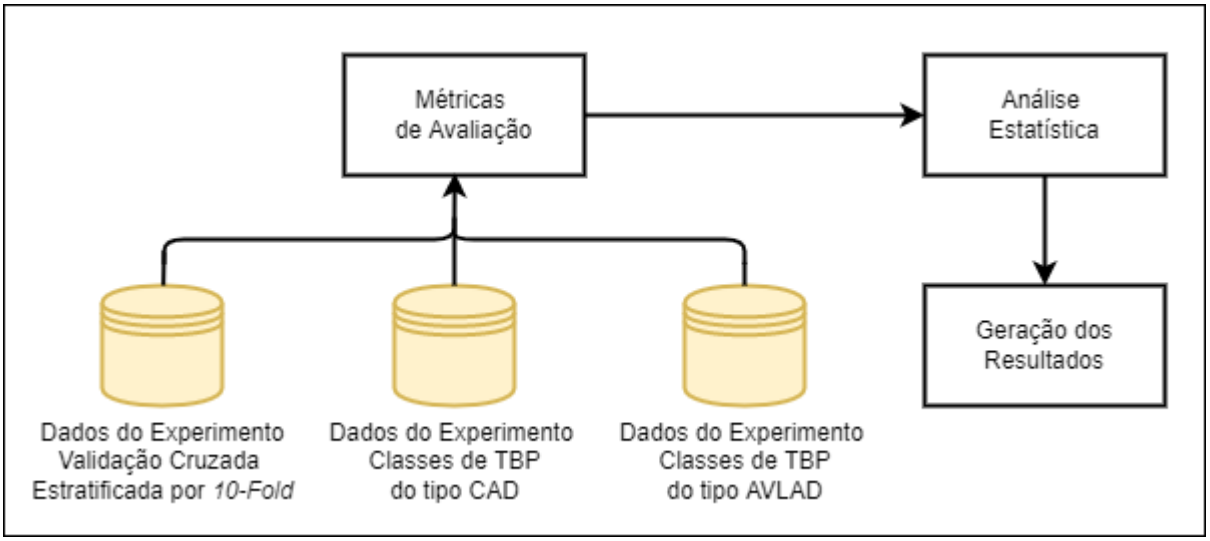

Figura 22 – Execução das métricas de avaliação de desempenho, testes estatísticos e geração dos resultados.

Para executar as métricas de avaliação de desempenho foi utilizado um *script* Python do Google Colab que importou as métricas de avaliação que são implementadas e disponibilizadas no *framework* `scikit-learn`. Todos os dados coletados durante a execução do experimento foram importados por meio desse *script* que calcula as métricas de avaliação de desempenho, realiza a análise e executa os testes estatísticos. Para tal análise, utilizou-se o *framework* `pandas` e a biblioteca `scipy`[4]. Os resultados dos testes

[4] https://scipy.org/

estatísticos e a discussão dos resultados do experimento serão apresentados na próxima seção.

7.3 Resultados

Na Tabela 10 são apresentados os resultados do experimento. Cada linha da tabela que possui o nome de um conjunto de dados representa o modelo 5-NN que foi treinado com aquele conjunto e depois foi avaliado e testado. Para cada um dos três tipos de teste (classe de teste baseado em propriedades do tipo CAD, classe de teste baseado em propriedades do tipo AVLAD e validação cruzada estratificada por *10-fold*) que foram executados durante a realização do experimento, são apresentados na tabela, os resultados em termos das quatro métricas de avaliação de desempenho utilizadas. Adicionalmente, são apresentados na Tabela 10 as estatísticas descritivas e os resultados da execução dos testes de normalidade. Os testes de normalidade e os resultados serão discutidos nas próximas subseções.

Tabela 10 – Resultados da execução das classes de teste baseado em propriedades do tipo CAD e do tipo AVLAD que foram geradas com a ferramenta proposta e da aplicação da validação cruzada estratificada por *10-fold* contra os modelos 5-NN que foram construídos com cada conjunto de dados.

Conjuntos de Dados	**Classes de TBP do Tipo CAD**				**Classes de TBP do Tipo AVLAD**				**Validação Cruzada Estratificada *10-fold***			
	Acurácia	**F_1**	**Revocação**	**Precisão**	**Acurácia**	**F_1**	**Revocação**	**Precisão**	**Acurácia**	**F_1**	**Revocação**	**Precisão**
Bank Marketing	0,51	0,43	0,51	0,48	0,51	0,45	0,51	0,48	0,88	0,87	0,88	0,86
Banknote Authentication	0,78	0,78	0,78	0,78	0,69	0,68	0,69	0,69	1,00	1,00	1,00	1,00
Blood Transfusion	0,52	0,53	0,52	0,53	0,54	0,54	0,54	0,53	0,76	0,73	0,76	0,74
Breast Cancer Wisconsin	0,75	0,75	0,75	0,75	0,73	0,73	0,73	0,74	0,94	0,93	0,94	0,94
Cleveland Heart Disease	0,50	0,50	0,50	0,50	0,50	0,50	0,50	0,50	0,64	0,63	0,64	0,64
Glass Identification	0,37	0,34	0,37	0,37	0,36	0,32	0,36	0,35	0,66	0,63	0,66	0,64
Haberman's Survival	0,62	0,58	0,62	0,60	0,61	0,57	0,61	0,59	0,72	0,69	0,72	0,69
Horse Racing Tipster Bets	0,49	0,38	0,49	0,43	0,49	0,39	0,49	0,43	0,77	0,72	0,77	0,70
Indian Liver Patient	0,51	0,41	0,51	0,52	0,50	0,42	0,50	0,49	0,67	0,66	0,67	0,66
Ionosphere	0,60	0,58	0,60	0,74	0,62	0,61	0,62	0,77	0,84	0,83	0,84	0,86
Iris	0,58	0,59	0,58	0,61	0,57	0,57	0,57	0,63	0,95	0,95	0,95	0,96
Oil Spill	0,50	0,37	0,50	0,64	0,50	0,35	0,50	0,68	0,96	0,94	0,96	0,92
Page Blocks Classification	0,40	0,39	0,40	0,40	0,41	0,40	0,41	0,40	0,96	0,95	0,96	0,95
Phoneme	0,49	0,49	0,49	0,49	0,51	0,51	0,51	0,51	0,89	0,89	0,89	0,89
Pima Indians Diabetes	0,49	0,49	0,49	0,49	0,50	0,48	0,50	0,50	0,71	0,70	0,71	0,70
Seeds	0,65	0,65	0,65	0,66	0,74	0,73	0,74	0,74	0,87	0,86	0,87	0,88
Sonar, Mines vs. Rocks	0,66	0,66	0,66	0,67	0,67	0,67	0,67	0,69	0,80	0,80	0,80	0,81
Students Knowledge Levels	0,54	0,53	0,54	0,55	0,50	0,46	0,50	0,54	0,88	0,88	0,88	0,89
Vertebral Column	0,60	0,60	0,60	0,61	0,57	0,57	0,57	0,58	0,82	0,82	0,82	0,84
Wine Recognition	0,41	0,41	0,41	0,44	0,35	0,36	0,35	0,40	0,69	0,67	0,69	0,71
Woods Mammography	0,60	0,52	0,60	0,64	0,61	0,54	0,61	0,63	0,99	0,98	0,99	0,98
Estatísticas Descritivas												
Máx	0,78	0,78	0,78	0,78	0,74	0,73	0,74	0,77	1,00	1,00	1,00	1,00
Mín	0,37	0,34	0,37	0,37	0,35	0,32	0,35	0,35	0,64	0,63	0,64	0,64
Média	0,55	0,52	0,55	0,57	0,55	0,52	0,55	0,57	0,83	0,82	0,83	0,82
Mediana	0,52	0,52	0,52	0,55	0,51	0,51	0,51	0,54	0,84	0,83	0,84	0,86
Desvio Padrão	0,11	0,12	0,11	0,12	0,11	0,12	0,11	0,12	0,12	0,12	0,12	0,12
Testes de Normalidade												
Shapiro-Wilk (W)	W=0,96, p=0,45	W=0,96, p=0,46	W=0,96, p=0,45	W=0,97, p=0,72	W=0,95, p=0,33	W=0,96, p=0,58	W=0,95, p=0,33	W=0,96, p=0,54	W=0,94, p=0,21	W=0,93, p=0,12	W=0,94, p=0,21	W=0,92, p=0,07

7.3.1 Teste de Hipóteses

Para investigar se a abordagem proposta para o teste baseado em propriedades de modelos de aprendizado de máquina possibilita gerar dados de teste eficazes, foram realizados testes T de duas amostras não pareadas para avaliar as diferenças existentes entre as médias de valores que foram obtidas como resultados das métricas de desempenho utilizadas no experimento, a saber: acurácia, F_1, revocação e precisão. Porém, antes de

realizar os testes T, foram realizados testes Shapiro-Wilk para verificar a normalidade dos valores que foram obtidos como resultados. Conforme pode ser constatado na Tabela 10, todos os resultados da execução das classes de teste baseado em propriedades construídas com a abordagem proposta bem como os resultados da realização de validação cruzada estratificada por *10-fold* parecem seguir uma distribuição normal.

Conforme mencionado, após a realização dos testes de normalidade foram realizados os testes T de duas amostras não pareadas para testar as hipóteses que foram definidas na Seção 7.1.2. O teste T é um tipo de teste paramétrico que normalmente é utilizado para comparar a média de duas amostras (ou seja, as amostras de dados de teste) em termos de variáveis dependentes (ou seja, as métricas de desempenho escolhidas), mas sob diferentes níveis de uma variável independente (ou seja, as classes de teste baseado em propriedades do tipo CAD, as classes de teste baseado em propriedades do tipo AVLAD e o método de validação cruzada estratificada por *10-fold*). De acordo com os resultados da realização dos testes T, apresentados na Tabela 11, as classes de teste construídas com a abordagem de teste baseado em propriedades proposta neste trabalho possibilitam gerar amostras de dados de teste para modelos clássicos de aprendizado de máquina que diferem significativamente das amostras de dados de teste que podem ser obtidas com a utilização do método de validação cruzada estratificada por *10-fold*.

Tabela 11 – Resumo dos resultados da realização dos testes T de duas amostras não pareadas.

	Teste T de Duas Amostras Não Pareadas	
Métrica	**Classes do Tipo CAD x Validação Cruzada**	**Classes do Tipo AVLAD x Validação Cruzada**
Acurácia	t = -8,09 p-value $\leq$ 0,001	t = -8,15 p-value $\leq$ 0,001
F_1	t = -7,73 p-value $\leq$ 0,001	t = -7,91 p-value $\leq$ 0,001
Revocação	t = -8,09 p-value $\leq$ 0,001	t = -8,15 p-value $\leq$ 0,001
Precisão	t = -6,96 p-value $\leq$ 0,001	t = -6,83 p-value $\leq$ 0,001

7.3.2 Discussão dos Resultados

Para responder a **QP**, focamos em investigar se as classes de teste construídas com a abordagem de teste baseado em propriedades podem ser utilizadas para realização do teste sistemático de modelos clássicos de aprendizado de máquina. Dado que o objetivo das classes de teste geradas com a ferramenta que implementa a abordagem proposta é possibilitar o teste automatizado de modelos clássicos de aprendizado de máquina. Especificamente, o objetivo é possibilitar a automatização da geração de amostras de dados

de teste que sejam mais eficazes (estressam o modelo sendo testado, resultando na diminuição do desempenho refletido pelas métricas sendo consideradas) do que as amostras de dados de teste que podem ser obtidos com a utilização de métodos de validação cruzada. Conjectura-se que a abordagem de teste baseado em propriedades proposta neste trabalho possibilita a geração de amostras de dados de teste que resultam na diminuição de desempenho dos modelos testados.

Postula-se que a abordagem de teste baseado em propriedades para modelos clássicos de aprendizado de máquina proposta neste trabalho possibilita gerar de maneira sistemática, amostras de dados de teste que são mais diversas do que as amostras de dados que podem ser amostradas de um conjunto de dados pré-existente por meio da utilização do método de validação cruzada estratificada por *10-fold*. Porém, não são amostras que possuem valores gerados de forma totalmente arbitrária. Pelo contrário, são amostras de dados de teste cujos valores são fundamentados nas informações que foram extraídas de um modelo de árvore de decisão. Nesse contexto, medimos a eficácia de um conjunto de amostras de dados de teste em termos de quanto as amostras de dados do conjunto de teste conseguem provocar a diminuição do desempenho preditivo do modelo em teste. Nesse sentido, quanto mais eficaz um conjunto de dados de teste, pior será o desempenho preditivo que o modelo em teste apresentará para dados desse conjunto.

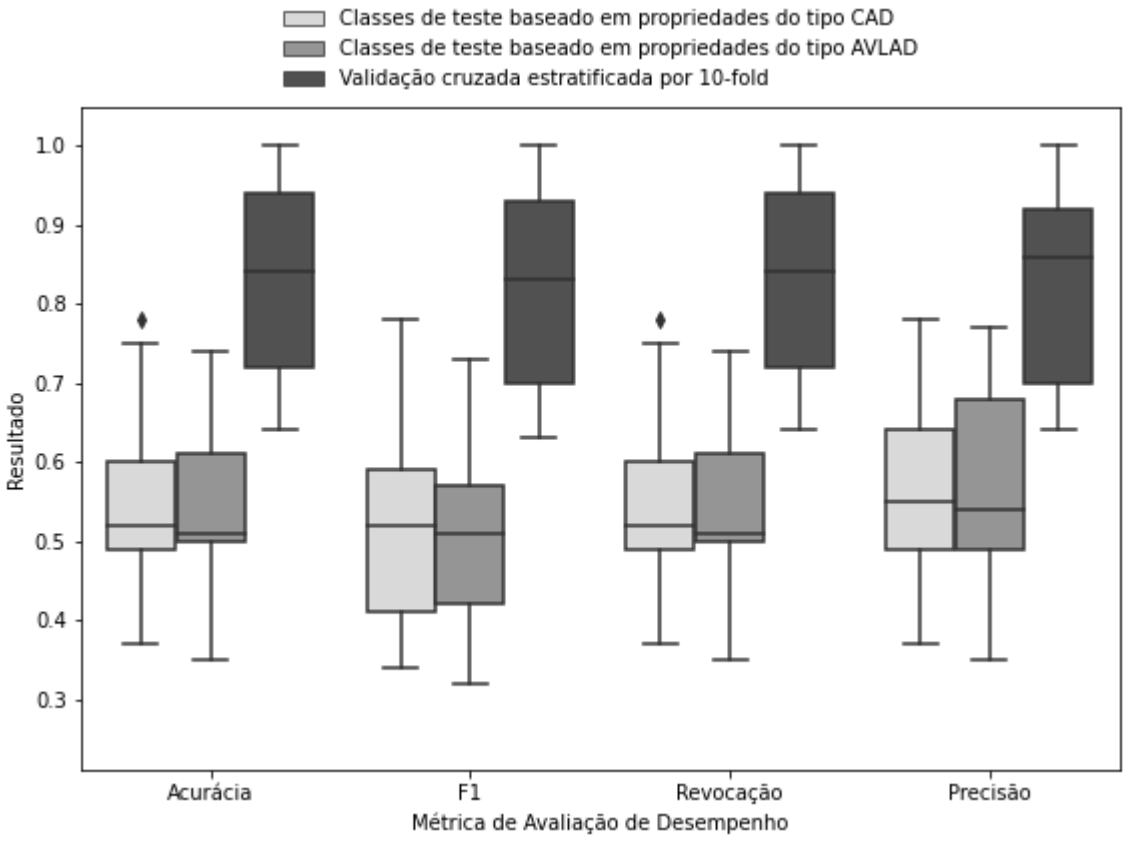

Figura 23 – Resultados da execução das classes de teste baseado em propriedades do tipo CAD e do tipo AVLAD que foram geradas com a ferramenta proposta e da aplicação da validação cruzada estratificada por *10-fold* contra os modelos 5-NN que foram construídos com os 21 conjuntos de dados.

Conforme mostrado na Figura 23, ao utilizar as classes de teste baseado em propriedades construídas com a abordagem proposta para testar os modelos 5-NN que foram avaliados com o método de validação cruzada estratificada por *10-fold*, os modelos testados apresentaram reduções no desempenho preditivo para todas as métricas utilizadas no experimento. Diante disso, é possível fundamentar a resposta para a **QP** nos resultados das métricas: acurácia, F_1, revocação e precisão. Conforme mencionado, tais métricas foram utilizadas como uma aproximação para medir a eficácia das amostras de dados de teste geradas com as classes de teste utilizados no experimento. Os resultados apresentados na Tabela 10 sugerem que as amostras de dados de teste geradas com a execução das classes construídas com a abordagem proposta são mais diversas do que as amostras de dados de teste que podem ser obtidas por meio da realização da validação cruzada estratificada por *10-fold*. Constatação essa que é embasada na diminuição significativa de desempenho que foi apresentada pelos modelos ao serem testados com as classes de teste geradas com a ferramenta proposta. Por exemplo, o modelo 5-NN construído com a utilização do conjunto de dados `Glass Identification`, durante a realização da validação cruzada estratificada por *10-fold* apresentou um valor F_1 de 0,63. Porém, apresentou um desempenho consideravelmente menor para as amostras de dados de teste que foram geradas com a execução da classe de teste baseado em propriedades do tipo CAD: $F_1 = 0,34$. De forma semelhante, o modelo também apresentou um desempenho menor para as amostras de dados de teste geradas com a execução da classe de teste baseado em propriedades do tipo AVLAD: $F_1 = 0,32$. Para o modelo 5-NN construído com a utilização do conjunto de dados `Oil Spill`, o valor de F_1 obtido com realização da validação cruzada foi de $0,94$. Todavia, houve uma diminuição de desempenho significativa quando o modelo foi testado com a classe de teste baseado em propriedades do tipo CAD ($F_1 = 0,37$) e uma diminuição ainda maior quando testado com a classe de teste baseado em propriedades do tipo AVLAD ($F_1 = 0,35$).

Ao considerar os resultados da métrica F_1, é possível perceber que existe uma diferença estatística significativa entre a média dos resultados da execução das classes de teste baseado em propriedades do tipo CAD *(média = 0,52)* e a média dos resultados da realização da validação cruzada estratificada por *10-fold (média = 0,82)*: $t = -7.73$, $p - value \leq 0,001$ (Tabela 11). Também existe uma diferença estatística significativa entre a média dos resultados da utilização das classes de teste baseado em propriedades do tipo AVLAD *(média = 0,52)* e a média dos resultados da validação cruzada: $t = -7,91$, $p - value \leq 0,001$ (Tabela 11).

Conforme apresentado na Tabela 11, existe uma diferença estatística significativa entre os resultados da realização do teste dos modelos utilizando as classes de teste baseado em propriedades do tipo CAD e os resultados da realização da validação cruzada estratificada por *10-fold* para todas as métricas consideradas. Além disso, os resultados dos testes sugerem que também existe uma diferença estatística significativa em relação

aos resultados das classes de teste baseado em propriedades do tipo AVLAD e os resultados das validações cruzadas para todas as métricas utilizadas. Argumenta-se que isso indica que as classes de teste construídas com a abordagem de teste baseado em propriedades podem ser utilizadas para realização do teste sistemático e automatizado de modelos de aprendizado de máquina; possibilitando assim a geração automatizada de amostras de dados de teste diversas que tendem a ser mais eficazes do que as amostras de dados de teste normalmente obtidas por meio da utilização de métodos de validação cruzada. Acredita-se que a utilização de dados de teste mais diversos pode levar a detecção de comportamentos errôneos (ou seja, falhas relacionadas à corretude) de modelos clássicos de aprendizado de máquina.

7.4 Limitações e Ameças à Validade

Apesar do experimento realizado ter sido projetado e executado de modo que fosse possível minimizar as ameaças à sua validade, assim como pode acontecer com a condução de qualquer estudo experimental, algumas ameaças não puderam ser eliminadas. Diante disso, esta seção apresenta uma breve discussão de algumas ameaças que podem comprometer a validade do experimento. A discussão será centrada em dois tipos de ameaças à validade (WOHLIN et al., 2012): *(i)* ameaça à validade externa e *(ii)* ameaça à validade de construto.

7.4.1 Ameaças à Validade Externa

Uma ameaça externa à validade do experimento realizado é devido a possibilidade dos conjuntos de dados selecionados não serem suficientemente representativos dos conjuntos de dados que normalmente são utilizados no mundo real. Os conjuntos de dados que foram utilizados no experimento são menores e mais simples do que os conjuntos de dados que normalmente são utilizados em aplicações reais. Portanto, não pode ser descartada a ameaça de que os resultados poderiam ter sido diferentes caso conjuntos de dados maiores, provenientes de aplicações reais, tivessem sido utilizados.

7.4.2 Ameaças à Validade de Construto

Uma ameaça à validade de construto refere-se a possibilidade das métricas de avaliação de desempenho utilizadas no experimento não serem adequadas para avaliar a eficácia da abordagem proposta e também de avaliar a eficácia do método de validação cruzada estratificada que foi utilizado. Especificamente, as métricas, acurácia, F_1, revocação e precisão podem não fornecer os melhores indicadores da eficácia das amostras de dados de teste que foram geradas ou selecionadas com a abordagem e método de teste utilizados. No entanto, é importante ressaltar que essas quatro métricas são amplamente

utilizadas para avaliação e teste de modelos classificadores de aprendizado de máquina, o que pode contribuir para a redução do risco dessa ameaça à validade do experimento realizado.

7.5 Considerações Finais

Este capítulo descreveu o experimento que foi realizado neste trabalho para avaliar a eficácia da abordagem de teste baseado em propriedades para modelos clássicos de aprendizado de máquina que foi proposta. Para executar o experimento foram selecionados 21 conjuntos de dados distintos do *UCI Machine Learning Repository* que foram submetidos a ferramenta proposta para geração de duas classes de teste baseado em propriedades (uma do tipo CAD e outra do tipo AVLAD) para cada conjunto de dados. Em seguida, executou-se cada par de classes de teste baseado em propriedades contra um modelo 5-NN que foi construído com o mesmo conjunto de dados que originou cada par de classes de teste. Também realizou-se a validação cruzada estratificada por *10-fold* de modelos 5-NN construídos com cada conjunto de dados selecionado. Por fim, foram realizados uma comparação e uma análise estatística dos valores obtidos a partir da aplicação de quatro métricas de avaliação de desempenho sobre os dados obtidos com as execuções das classes de teste baseado em propriedades e com a realização da validação cruzada estratificada por *10-fold*.

Os resultados sugerem que a abordagem proposta neste trabalho é eficaz para realização do teste baseado em propriedades de modelos clássicos de aprendizado de máquina, gerando classes de teste baseado em propriedades que quando executadas são capazes de produzir amostras de dados de teste que tendem a ser mais eficazes do que as amostras que podem ser obtidas por meio da utilização do método de validação cruzada estratificada por *10-fold*.

8 Considerações Finais

Neste capítulo são apresentadas as considerações finais do trabalho em relação aos seguintes elementos apresentados no decorrer do documento: a abordagem proposta, a ferramenta desenvolvida e ao experimento conduzido. Para apresentar essa discussão de forma estruturada foi optado por dividir o capítulo em três seções. Na Seção 8.1 apresenta-se a conclusão do trabalho. Na Seção 8.2 são discutidas algumas limitações identificadas na abordagem proposta. Por fim, na Seção 8.3 possíveis trabalhos futuros que podem ser desenvolvidos como uma extensão deste trabalho são brevemente apresentados.

8.1 Considerações Finais Relacionadas ao Trabalho Proposto

Neste trabalho foi proposta uma abordagem de teste baseado em propriedades para modelos clássicos de aprendizado de máquina. A abordagem proposta consiste na utilização dos critérios de teste CAD e AVLAD inicialmente propostos por Santos et al. (2021), conjuntamente com a técnica de teste baseado em propriedades utilizada no teste de sistemas tradicionais. A utilização desses critérios juntamente com a abordagem de teste baseada em propriedades possibilita o desenvolvimento de uma abordagem de teste mais adaptada para o contexto de sistemas baseados em aprendizado de máquina, visto que o conhecimento relacionado a criação dos modelos (por meio dos critérios CAD e AVLAD) é combinado com uma abordagem para geração de dados de teste que respeitam determinados valores limites (ou seja, a abordagem de teste baseada em propriedades).

Diferente da proposta original de Santos et al. (2021), que propuseram os critérios de teste CAD e AVLAD com o objetivo de possibilitar a seleção (amostragem) de amostras de dados de teste de conjuntos de dados pré-existentes. Neste trabalho, os critérios de teste CAD e AVLAD foram utilizados para desenvolver uma abordagem que permite gerar conjuntos de propriedades. Os conjuntos de propriedades podem ser codificados em uma classe de teste (`arquivo.py`) executável, compatível com alguma biblioteca de teste baseado em propriedades e com algum *framework* de teste automatizado para realização do teste automatizado de modelos clássicos de aprendizado de máquina. Além disso, foi estabelecida a hipótese de que as classes de teste executáveis contendo as propriedades geradas com a abordagem proposta poderiam ser utilizadas para geração automatizada de amostras de dados de teste que podem ser mais eficazes do que as amostras de dados de teste selecionadas por meio da utilização de métodos de validação cruzada.

Para possibilitar o teste automatizado de modelos clássicos de aprendizado de máquina foi desenvolvida uma ferramenta que automatiza o processo de geração de propriedades da abordagem proposta. A ferramenta desenvolvida consiste na implementação

do gerador de propriedades (o elemento central do processo de geração de propriedades) que foi especificado na Seção 6.4. Ao utilizar a ferramenta proposta é possível gerar classes de teste executáveis que codificam as propriedades geradas pelo gerador de propriedades implementado. Porém, para a geração de classes de teste executáveis ser possível, foram adotadas algumas tecnologias como referência para o desenvolvimento da abordagem e da ferramenta que foram propostas. Python foi escolhida como a linguagem de referência, o que possibilitou a escolha do *framework* de aprendizado de máquina `scikit-learn`, a escolha do *framework* de teste automatizado `pytest` e a escolha da biblioteca de teste baseado em propriedades `Hypothesis`. Com a utilização dessas tecnologias, foi desenvolvido a ferramenta (apresentada na Seção 6.6) que possibilita gerar classes de teste baseado em propriedades dos tipos CAD e AVLAD para realização do teste automatizado de modelos clássicos de aprendizado de máquina construídos com o `scikit-learn`.

Foi realizado um experimento com o objetivo de investigar a hipótese de que as classes de teste baseado em propriedades dos tipos CAD e AVLAD construídas com a ferramenta proposta possibilitam gerar amostras de dados de teste eficazes para realização do teste de modelos clássicos de aprendizado de máquina. O experimento consistiu em comparar a eficácia das amostras de dados de teste geradas com a execução das classes de teste construídas com a ferramenta proposta com a eficácia das amostras de dados de teste obtidas por meio da utilização do método de validação cruzada estratificada por *10-fold*. Os resultados sugerem que as classes de teste baseado em propriedades do tipo CAD e AVLAD possibilitaram a geração de amostras de dados de teste mais eficazes do que as amostras de dados de teste obtidas com a utilização da validação cruzada estratificada para todos os 21 modelos 5-NN testados e avaliados durante o experimento. Indicando que a abordagem e a ferramenta propostas neste trabalho compõem uma opção válida para realização do teste baseado em propriedades de modelos clássicos de aprendizado de máquina. Adicionalmente, os resultados indicam que a abordagem proposta tende a gerar dados de teste que expõe modelos à situações mais complexas e variadas. Visto que submeter os modelos aos dados de teste baseados em propriedades resultou na diminuição das métricas de desempenho, acredita-se que tais dados são capaz de expor as limitações dos modelos sendo testados, de forma que engenheiros de aplicações baseadas em modelos de aprendizado de máquina podem utilizar algumas dessas informações para criação de modelos mais robustos e capazes de lidar com situações mais variadas.

8.2 Limitações da Abordagem Proposta

Assim como muitas técnicas e abordagens de teste encontradas na literatura e, principalmente, na literatura de teste de sistemas baseados em aprendizado de máquina, a abordagem de teste baseado em propriedades para modelos clássicos de aprendizado de máquina proposta neste trabalho também apresenta algumas limitações. Uma limitação

existente na abordagem proposta decorre diretamente de uma limitação encontrada nos critérios de teste CAD e AVLAD que foram propostos por Santos et al. (2021), que refere-se a necessidade de utilizar apenas valores numéricos em nós internos da árvore de decisão utilizada para aplicação desses critérios – ou seja, os critérios de teste CAD e AVLAD exigem que as características (atributos) existentes nos conjuntos de dados utilizados sejam compostas apenas por valores numéricos. Da mesma forma, a abordagem proposta neste trabalho também apresenta essa limitação, uma vez que, os intervalos de valores que são derivados da árvore de decisão para geração das propriedades de teste também necessitam ser formados apenas por valores numéricos. Naturalmente, tal limitação também é aplicável a ferramenta que implementa a abordagem proposta.

Outra limitação que deve ser destacada está relacionada às asserções de teste que são construídas para as propriedades geradas com a abordagem proposta. Os resultados esperados definidos nas asserções de teste são considerados como aproximações para os resultados esperados reais. Na verdade, essa também é uma limitação que pode surgir ao aplicar o critério AVLAD que foi proposto por Santos et al. (2021). Tal limitação decorre diretamente do problema da inexistência de oráculos de teste para o teste de sistemas baseados em aprendizado, conforme discutido na Seção 4.6. Como destacado por Zhang et al. (2022), a identificação de oráculos de teste permanece sendo um problema para o teste de sistemas baseados em aprendizado de máquina e pode ser necessário contratar especialistas de domínio do sistema em teste para verificar manualmente os resultados gerados pelo sistema. Diante disso, a construção de asserções de teste contendo resultados esperados aproximados foi a solução adotada neste trabalho para mitigar tal problema.

8.3 Trabalhos Futuros

Com o desenvolvimento deste trabalho foram identificados alguns pontos de melhorias existentes na abordagem e na ferramenta de teste baseado em propriedades para modelos clássicos de aprendizado de máquina que foram propostas. Também foram identificadas questões relacionadas ao experimento conduzido que podem ser investigadas de forma mais abrangente. Assim, nas Subseções 8.3.1, 8.3.2 e 8.3.3 são destacados os trabalhos futuros que podem ser desenvolvidos a partir deste trabalho.

8.3.1 Trabalho Futuro Relacionado ao Experimento Conduzido

Para ser possível investigar a eficácia da abordagem proposta de forma mais abrangente, é considerada a necessidade de expandir o experimento realizado, com a finalidade de englobar modelos que sejam construídos com outros tipos de algoritmos de aprendizado de máquina, como por exemplo, os algoritmos Naive Bayes e SVM. Dessa forma, pesquisas e experimentos futuros podem ser conduzidos com o objetivo de avaliar a eficácia e

aplicabilidade das amostras de dados de teste geradas com as classes de teste baseado em propriedades construídas com a ferramenta proposta durante a realização do teste de modelos de aprendizado de máquina que sejam diferentes do K-NN.

8.3.2 Trabalho Futuro Relacionado a Abordagem Proposta

É considerada a possibilidade de modificar o procedimento de geração de propriedades da abordagem proposta para fazer com que as propriedades possibilitem a geração de uma quantidade máxima de amostras de dados de teste que seja proporcional à quantidade de amostras de dados de treinamento que forem classificadas pelo caminho raiz à folha que originou a propriedade. Por exemplo, na abordagem proposta, caso um caminho da raiz à folha $\mathcal{A}$ classifique 100 amostras de dados de treinamento durante o processo de treinamento do modelo de árvore de decisão, e outro caminho $\mathcal{B}$, classifique apenas 10 amostras. Durante o processo de geração de propriedades da abordagem proposta, serão geradas duas propriedades, uma originada do caminho $\mathcal{A}$ e outra originada do caminho $\mathcal{B}$, que serão configuradas para gerarem a mesma quantidade máxima de amostras de dados de teste. Uma possível extensão da abordagem proposta que pode ser realizada em um trabalho futuro consiste na modificação desse procedimento de geração. No caso de tal modificação, considerando o exemplo anterior: a propriedade originada do caminho $\mathcal{A}$ seria configurada para gerar uma quantidade máxima de amostras de dados de teste proporcional às 100 amostras de dados de treinamento que foram classificadas pelo caminho $\mathcal{A}$. De modo semelhante, a propriedade originada do caminho $\mathcal{B}$ também seria configurada para gerar uma quantidade máxima de amostras de dados de teste proporcional às 10 amostras de dados de treinamento que foram classificadas pelo caminho $\mathcal{B}$. Dessa forma, acredita-se ser possível melhorar a representatividade das amostras de dados de teste que serão geradas ao executar as classes de teste baseado em propriedades construídas com a ferramenta proposta: visto que serão geradas quantidades de amostras de dados de teste proporcionais às quantidades de amostras de dados de treinamento de cada tipo de rótulo existente no conjunto de treinamento.

8.3.3 Trabalho Futuro Relacionado a Ferramenta Proposta

Um ponto de melhoria relacionado à ferramenta proposta refere-se às tecnologias adotadas como referência, especificamente, o *framework* de teste automatizado `pytest` e a biblioteca de teste baseado em propriedades `Hypothesis`. A utilização conjunta do `pytest` com a `Hypothesis` caracteriza uma escolha assertiva para o desenvolvimento da ferramenta proposta neste trabalho. Porém, essas ferramentas foram originalmente propostas para o teste de sistemas tradicionais. Devido a isso, o suporte nativo que elas oferecem para a realização do teste baseado em propriedades de modelos clássicos de aprendizado de máquina é de certa forma limitado. Principalmente, no que diz respeito

aos relatórios de teste que são fornecidos ao testador no final da execução das propriedades de teste. Por exemplo, no teste baseado em propriedades de sistemas tradicionais, o ideal é que assim que uma propriedade em teste resulte em uma falha, o processo de geração de dados seja interrompido e que a falha encontrada seja reportada ao testador. Porém, no caso de modelos de aprendizado de máquina que possuem um comportamento estatístico por natureza, o comportamento desejado do `pytest` e da `Hypothesis` pode não ser esse.

No contexto mencionado acima, o cenário ideal seria aquele em que, durante o teste de um modelo de aprendizado de máquina, a `Hypothesis` gerasse exatamente a quantidade de amostras de dados de teste que as propriedades foram configuradas para gerar, independente da quantidade de falhas que forem identificadas durante a execução de determinada propriedade. Ao término da execução das classes de teste, o recomendado seria que o `pytest` fosse capaz de reportar ao testador a porcentagem de erros e acertos obtidos com a execução de cada propriedade, ou até mesmo os escores obtidos por meio do cálculo de determinadas métricas de avaliação de desempenho, ao invés de reportar um indicador de sucesso ou falha para cada propriedade executada.

A biblioteca `Hypothesis` permite customizar o comportamento do processo de geração de propriedades para executar apenas a fase de geração de dados de teste. Por meio de uma modificação realizada no código-fonte da `Hypothesis`, é possível fazer com que a ferramenta gere a quantidade de amostras de dados de teste que foram configuradas em cada propriedade, independente da identificação de alguma falha durante o teste da propriedade. Porém, para ser possível obter um relatório que contenha informações significativas ao término da realização do teste de um modelo é preciso implementar um novo plugin para o `pytest` que realize essa função específica.

Na abordagem proposta, tal limitação técnica foi mitigada fazendo com que as classes de teste baseado em propriedades construídas com a ferramenta proposta, gerassem ao término de sua execução, um relatório no formato JSON contendo: todas as amostras de dados de teste, os resultados esperados e os resultados gerados por cada propriedade executada. De posse desse relatório, o testador pode analisar os resultados da realização do teste do modelo com mais informações. Diante disso, outra proposta de trabalho futuro que pode ser desenvolvida como uma extensão deste trabalho, consiste na implementação de um plugin para o `pytest` que seja capaz de fornecer esse tipo de relatório para o testador ao término da realização do teste de modelos de aprendizado de máquina. Tal plugin poderia aumentar a utilidade prática da ferramenta proposta e possibilitar que a ferramenta seja mais amplamente utilizada, inclusive na indústria.

Referências

ABADI, M.; BARHAM, P.; CHEN, J.; CHEN, Z.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; IRVING, G.; ISARD, M.; KUDLUR, M.; LEVENBERG, J.; MONGA, R.; MOORE, S.; MURRAY, D. G.; STEINER, B.; TUCKER, P.; VASUDEVAN, V.; WARDEN, P.; WICKE, M.; YU, Y.; ZHENG, X. Tensorflow: A system for large-scale machine learning. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. [S.l.: s.n.], 2016. p. 265–283. Citado 2 vezes nas páginas 86 e 89.

AHUJA, M. K.; GOTLIEB, A.; SPIEKER, H. Testing deep learning models: A first comparative study of multiple testing techniques. *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, p. 130–137, 2022. Citado 2 vezes nas páginas 75 e 76.

ALBON, C. *Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning*. [S.l.]: O'Reilly Media, 2018. ISBN 9781491989388. Citado 3 vezes nas páginas 52, 55 e 56.

ALI, N.; NEAGU, D.; TRUNDLE, P. Evaluation of k-nearest neighbour classifier performance for heterogeneous data sets. *SN Applied Sciences*, v. 1, 11 2019. Citado 2 vezes nas páginas 47 e 48.

ALMANA, A. M.; AKSOY, M. S. An overview of inductive learning algorithms. *International Journal of Computer Applications*, v. 88, p. 20–28, 2014. Citado 2 vezes nas páginas 46 e 47.

AMERSHI, S.; BEGEL, A.; BIRD, C.; DELINE, R.; GALL, H.; KAMAR, E.; NAGAPPAN, N.; NUSHI, B.; ZIMMERMANN, T. Software engineering for machine learning: A case study. In: *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*. [S.l.]: IEEE Press, 2019. (ICSE-SEIP '19), p. 291–300. Citado 9 vezes nas páginas 9, 17, 48, 49, 50, 60, 66, 67 e 72.

AMMANN, P.; OFFUTT, J. *Introduction to Software Testing*. [S.l.]: Cambridge University Press, 2016. ISBN 9781316773123. Citado 15 vezes nas páginas 9, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 34, 35 e 41.

ANICHE, M. *Effective Software Testing: A developer's guide*. [S.l.]: Manning, 2022. ISBN 9781638350583. Citado 5 vezes nas páginas 27, 32, 36, 37 e 98.

ASTHANA, P.; HAZELA, B. Applications of machine learning in improving learning environment. *Intelligent Systems Reference Library*, 2019. Citado na página 55.

BERTOLINO, A.; MARCHETTI, E. A brief essay on software testing. *Software Engineering: The Development Process - A Brief Essay on Software Testing*, v. 1, 01 2005. Citado na página 29.

BRAIEK, H. B.; KHOMH, F. Deepevolution: A search-based testing approach for deep neural networks. *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, p. 454–458, 2019. Citado 2 vezes nas páginas 89 e 93.

BRAIEK, H. B.; KHOMH, F. Tfcheck : A tensorflow library for detecting training issues in neural network programs. In: *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*. [S.l.: s.n.], 2019. p. 426–433. Citado 2 vezes nas páginas 86 e 90.

BRAIEK, H. B.; KHOMH, F. On testing machine learning programs. *Journal of Systems and Software*, v. 164, p. 110542, 2020. ISSN 0164-1212. Citado 14 vezes nas páginas 9, 16, 17, 57, 58, 66, 68, 72, 74, 75, 76, 80, 82 e 86.

BRECK, E.; ZINKEVICH, M.; POLYZOTIS, N.; WHANG, S.; ROY, S. Data validation for machine learning. In: *Proceedings of SysML*. [S.l.: s.n.], 2019. Citado na página 67.

BREIMAN, L.; FRIEDMAN, J.; STONE, C.; OLSHEN, R. *Classification and Regression Trees*. [S.l.]: Taylor & Francis, 1984. ISBN 9780412048418. Citado na página 46.

BRIJAIN, M.; PATEL, R.; KUSHIK, M.; RANA, K. A survey on decision tree algorithm for classification. *International Journal of Engineering Development and Research*, v. 2, p. 1, 2014. Citado 2 vezes nas páginas 46 e 47.

BUITINCK, L.; LOUPPE, G.; BLONDEL, M.; PEDREGOSA, F.; MUELLER, A.; GRISEL, O.; NICULAE, V.; PRETTENHOFER, P.; GRAMFORT, A.; GROBLER, J.; LAYTON, R.; VANDERPLAS, J.; JOLY, A.; HOLT, B.; VAROQUAUX, G. Api design for machine learning software: experiences from the scikit-learn project. *ArXiv*, abs/1309.0238, 2013. Citado 3 vezes nas páginas 58, 59 e 60.

CLAESSEN, K.; HUGHES, J. Quickcheck: A lightweight tool for random testing of haskell programs. *SIGPLAN Not.*, Association for Computing Machinery, New York, NY, USA, v. 46, n. 4, p. 53–64, may 2011. ISSN 0362-1340. Citado 5 vezes nas páginas 16, 17, 35, 36 e 37.

COVER, T.; HART, P. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, v. 13, n. 1, p. 21–27, 1967. Citado na página 47.

CUNNINGHAM, P.; DELANY, S. J. K-nearest neighbour classifiers - a tutorial. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 54, n. 6, jul 2021. ISSN 0360-0300. Citado 3 vezes nas páginas 9, 47 e 48.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. *Introdução ao teste de software*. [S.l.]: Elsevier, 2016. Citado 14 vezes nas páginas 9, 21, 22, 23, 24, 25, 26, 27, 29, 32, 33, 34, 35 e 36.

DOL, M.; GEETHA, A. A learning transition from machine learning to deep learning: A survey. In: *2021 International Conference on Emerging Techniques in Computational Intelligence (ICETCI)*. [S.l.: s.n.], 2021. p. 89–94. Citado na página 43.

DOSHI-VELEZ, F.; KIM, B. Towards a rigorous science of interpretable machine learning. *arXiv: Machine Learning*, 2017. Citado na página 71.

DWARAKANATH, A.; AHUJA, M.; SIKAND, S.; RAO, R. M.; BOSE, R. P. J. C.; DUBASH, N.; PODDER, S. Identifying implementation bugs in machine learning based image classifiers using metamorphic testing. In: *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*. New York, NY, USA: Association for Computing Machinery, 2018. (ISSTA 2018), p. 118–128. ISBN 9781450356992. Citado na página 85.

ENGSTROM, L.; TRAN, B.; TSIPRAS, D.; SCHMIDT, L.; MADRY, A. *A Rotation and a Translation Suffice: Fooling CNNs with Simple Transformations*. 2019. Citado na página 88.

GÉRON, A. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. [S.l.]: O'Reilly Media, Incorporated, 2019. ISBN 9781492032649. Citado 6 vezes nas páginas 42, 43, 49, 50, 52 e 54.

GOODFELLOW, I.; POUGET-ABADIE, J.; MIRZA, M.; XU, B.; WARDE-FARLEY, D.; OZAIR, S.; COURVILLE, A.; BENGIO, Y. Generative adversarial networks. *Advances in Neural Information Processing Systems*, v. 3, 06 2014. Citado na página 87.

GOPINATH, D.; PASAREANU, C. S.; WANG, K.; ZHANG, M.; KHURSHID, S. Symbolic execution for attribution and attack synthesis in neural networks. In: *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings*. [S.l.]: IEEE Press, 2019. (ICSE '19), p. 282–283. Citado 2 vezes nas páginas 89 e 93.

GRAVES, A.; MOHAMED, A. rahman; HINTON, G. E. Speech recognition with deep recurrent neural networks. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, p. 6645–6649, 2013. Citado na página 93.

GU, R.; NIU, C.; WU, F.; CHEN, G.; HU, C.; LYU, C.; WU, Z. From server-based to client-based machine learning: A comprehensive survey. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 54, n. 1, jan 2021. ISSN 0360-0300. Citado na página 44.

GUO, J.; JIANG, Y.; ZHAO, Y.; CHEN, Q.; SUN, J. Dlfuzz: Differential fuzzing testing of deep learning systems. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2018. (ESEC/FSE 2018), p. 739–743. ISBN 9781450355735. Citado 2 vezes nas páginas 18 e 88.

HEBERT, F. *Property-Based Testing with PropEr, Erlang, and Elixir: Find Bugs Before Your Users Do*. [S.l.]: Pragmatic Bookshelf, 2019. (Pragmatic programmers). ISBN 9781680506211. Citado 2 vezes nas páginas 36 e 37.

HUGHES, J. How to specify it! a guide to writing properties of pure functions. In: *Trends in Functional Programming: 20th International Symposium, TFP 2019, Vancouver, BC, Canada, June 12–14, 2019, Revised Selected Papers*. Berlin, Heidelberg: Springer-Verlag, 2019. p. 58–83. ISBN 978-3-030-47146-0. Citado 4 vezes nas páginas 16, 35, 36 e 37.

HYNES, N.; SCULLEY, D.; TERRY, M. The data linter: Lightweight automated sanity checking for ml data sets. In: . [S.l.: s.n.], 2017. Citado na página 83.

JAMES, G.; WITTEN, D.; HASTIE, T.; TIBSHIRANI, R. *An Introduction to Statistical Learning: with Applications in R*. [S.l.]: Springer US, 2021. (Springer Texts in Statistics). ISBN 9781071614181. Citado na página 83.

Jia, Y.; Harman, M. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, v. 37, n. 5, p. 649–678, 2011. Citado na página 76.

JIANG, L.; CAI, Z.; WANG, D.; JIANG, S. Survey of improving k-nearest-neighbor for classification. In: *Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007)*. [S.l.: s.n.], 2007. v. 1, p. 679–683. Citado na página 48.

JORDAN, M. I.; MITCHELL, T. Machine learning: Trends, perspectives, and prospects. *Science*, v. 349, p. 255 – 260, 2015. Citado na página 44.

JORGENSEN, P.; DEVRIES, B. *Software Testing: A Craftsman's Approach, Fifth Edition*. [S.l.]: CRC Press, 2021. ISBN 9781000391497. Citado 2 vezes nas páginas 32 e 34.

KIM, J.; FELDT, R.; YOO, S. Guiding deep learning system testing using surprise adequacy. In: *Proceedings of the 41st International Conference on Software Engineering*. [S.l.]: IEEE Press, 2019. (ICSE '19), p. 1039–1049. Citado na página 89.

KIM, Y. Convolutional neural networks for sentence classification. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014. p. 1746–1751. Citado na página 93.

KRISHNAN, S.; FRANKLIN, M. J.; GOLDBERG, K.; WU, E. Boostclean: Automated error detection and repair for machine learning. *CoRR*, abs/1711.01299, 2017. Citado na página 83.

KRISHNAN, S.; WANG, J.; WU, E.; FRANKLIN, M. J.; GOLDBERG, K. Activeclean: Interactive data cleaning for statistical modeling. *Proc. VLDB Endow.*, VLDB Endowment, v. 9, n. 12, p. 948–959, aug 2016. ISSN 2150-8097. Citado na página 82.

KUWAJIMA, H.; YASUOKA, H.; NAKAE, T. Engineering problems in machine learning systems. *Machine Learning*, v. 109, 05 2020. Citado na página 17.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, v. 521, p. 436–44, 05 2015. Citado na página 93.

LIPTON, Z. C. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, Association for Computing Machinery, New York, NY, USA, v. 16, n. 3, p. 31–57, jun 2018. ISSN 1542-7730. Citado na página 71.

LORENA, A.; FACELI, K.; ALMEIDA, T.; CARVALHO, A. de; GAMA, J. *Inteligência Artificial: uma abordagem de Aprendizado de Máquina (2a edição)*. [S.l.: s.n.], 2021. ISBN 9788521637493. Citado na página 45.

LöSCHER, A.; SAGONAS, K. Targeted property-based testing. In: *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. New York, NY, USA: Association for Computing Machinery, 2017. (ISSTA 2017), p. 46–56. ISBN 9781450350761. Citado 3 vezes nas páginas 35, 36 e 41.

LöSCHER, A.; SAGONAS, K. Automating targeted property-based testing. In: *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. [S.l.: s.n.], 2018. p. 70–80. Citado 3 vezes nas páginas 16, 35 e 36.

MA, L.; JUEFEI-XU, F.; XUE, M.; LI, B.; LI, L.; LIU, Y.; ZHAO, J. Deepct: Tomographic combinatorial testing for deep learning systems. In: *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. [S.l.: s.n.], 2019. p. 614–618. Citado na página 89.

MA, L.; JUEFEI-XU, F.; ZHANG, F.; SUN, J.; XUE, M.; LI, B.; CHEN, C.; SU, T.; LI, L.; LIU, Y.; ZHAO, J.; WANG, Y. Deepgauge: Multi-granularity testing criteria for deep learning systems. In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2018. (ASE 2018), p. 120–131. ISBN 9781450359375. Citado 2 vezes nas páginas 88 e 93.

MA, Y.-S.; OFFUTT, J.; KWON, Y.-R. Mujava: A mutation system for java. In: . [S.l.: s.n.], 2006. v. 2006, p. 827. Citado na página 85.

MACIVER, D.; HATFIELD-DODDS, Z.; CONTRIBUTORS, M. Hypothesis: A new approach to property-based testing. *Journal of Open Source Software*, v. 4, p. 1891, 11 2019. Citado 2 vezes nas páginas 37 e 99.

MARIJAN, D.; GOTLIEB, A. Software testing for machine learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, v. 34, p. 13576–13582, 04 2020. Citado 2 vezes nas páginas 74 e 76.

MARIJAN, D.; GOTLIEB, A.; AHUJA, M. K. Challenges of testing machine learning based systems. In: *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*. [S.l.: s.n.], 2019. p. 101–102. Citado 9 vezes nas páginas 16, 61, 62, 70, 72, 73, 80, 81 e 82.

MCKEEMAN, W. M. Differential testing for software. *Digit. Tech. J.*, v. 10, p. 100–107, 1998. Citado na página 75.

MILI, A.; TCHIER, F. *Software Testing: Concepts and Operations*. [S.l.]: Wiley, 2015. (Quantitative Software Engineering Series). ISBN 9781119065579. Citado na página 26.

MOHAMMED, M.; KHAN, M.; BASHIER, E. *Machine Learning: Algorithms and Applications*. [S.l.: s.n.], 2016. ISBN 9781498705387. Citado 3 vezes nas páginas 42, 43 e 44.

MRABET, M. A. E.; MAKKAOUI, K. E.; FAIZE, A. Supervised machine learning: A survey. In: *2021 4th International Conference on Advanced Communication Technologies and Networking (CommNet)*. [S.l.: s.n.], 2021. p. 1–10. Citado 2 vezes nas páginas 56 e 57.

MÜLLER, A.; GUIDO, S. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. [S.l.]: O'Reilly Media, Incorporated, 2016. ISBN 9781449369408. Citado 9 vezes nas páginas 42, 49, 50, 52, 53, 54, 55, 56 e 57.

MURPHY, C.; KAISER, G.; ARIAS, M. An approach to software testing of machine learning applications. In: . [S.l.: s.n.], 2007. p. 167–. Citado 2 vezes nas páginas 72 e 84.

MURPHY, C.; KAISER, G.; HU, L.; WU, L. Properties of machine learning applications for use in metamorphic testing. In: . [S.l.: s.n.], 2008. p. 867–872. Citado na página 84.

MYERS, G. J.; SANDLER, C.; BADGETT, T. *The Art of Software Testing*. 3rd. ed. [S.l.]: Wiley Publishing, 2011. ISBN 1118031962. Citado 5 vezes nas páginas 21, 26, 27, 33 e 78.

NGUYEN, G.; DLUGOLINSKY, S.; BOBáK, M.; TRAN, V.; GARCÍA, A. L.; HEREDIA, I.; MALÍK, P.; HLUCH?, L. Machine learning and deep learning frameworks and libraries for large-scale data mining: A survey. *Artif. Intell. Rev.*, Kluwer Academic Publishers, USA, v. 52, n. 1, p. 77–124, jun 2019. ISSN 0269-2821. Citado 3 vezes nas páginas 57, 58 e 99.

NICULAESCU, O. Classifying data with decision trees. *XRDS*, Association for Computing Machinery, New York, NY, USA, v. 24, n. 4, p. 55–57, jul 2018. ISSN 1528-4972. Citado na página 46.

NIE, C.; LEUNG, H. A survey of combinatorial testing. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 43, n. 2, feb 2011. ISSN 0360-0300. Citado na página 75.

ODENA, A.; OLSSON, C.; ANDERSEN, D.; GOODFELLOW, I. TensorFuzz: Debugging neural networks with coverage-guided fuzzing. In: CHAUDHURI, K.; SALAKHUTDINOV, R. (Ed.). *Proceedings of the 36th International Conference on Machine Learning*. [S.l.]: PMLR, 2019. (Proceedings of Machine Learning Research, v. 97), p. 4901–4911. Citado na página 86.

OKKEN, B. *Python Testing with Pytest: Simple, Rapid, Effective, and Scalable (Second Edition)*. [S.l.]: Pragmatic Bookshelf, 2022. (Pragmatic programmers). ISBN 9781680508604. Citado 2 vezes nas páginas 31 e 32.

PAPADAKIS, M.; KINTIS, M.; ZHANG, J.; JIA, Y.; TRAON, Y. L.; HARMAN, M. Chapter six - mutation testing advances: An analysis and survey. In: MEMON, A. M. (Ed.). [S.l.]: Elsevier, 2019, (Advances in Computers, v. 112). p. 275 – 378. Citado na página 76.

PAPADAKIS, M.; SAGONAS, K. A proper integration of types and function specifications with property-based testing. In: *Proceedings of the 10th ACM SIGPLAN Workshop on Erlang*. New York, NY, USA: Association for Computing Machinery, 2011. (Erlang '11), p. 39–50. ISBN 9781450308595. Citado na página 35.

PARASKEVOPOULOU, Z.; HRIŢCU, C.; DÉNÈS, M.; LAMPROPOULOS, L.; PIERCE, B. C. Foundational property-based testing. In: URBAN, C.; ZHANG, X. (Ed.). *Interactive Theorem Proving*. Cham: Springer International Publishing, 2015. p. 325–343. ISBN 978-3-319-22102-1. Citado na página 37.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, JMLR.org, v. 12, n. null, p. 2825–2830, nov 2011. ISSN 1532-4435. Citado 2 vezes nas páginas 58 e 59.

PEI, K.; CAO, Y.; YANG, J.; JANA, S. Deepxplore: Automated whitebox testing of deep learning systems. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 62, n. 11, p. 137–145, oct 2017. ISSN 0001-0782. Citado 5 vezes nas páginas 18, 74, 87, 91 e 93.

PHALAK, M.; BHANDARI, M.; SHARMA, R. Analysis of decision tree-a survey. *International journal of engineering research and technology*, v. 3, 2014. Citado na página 46.

POUYANFAR, S.; SADIQ, S.; YAN, Y.; TIAN, H.; TAO, Y.; REYES, M. P.; SHYU, M.-L.; CHEN, S.-C.; IYENGAR, S. S. A survey on deep learning: Algorithms, techniques, and applications. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 51, n. 5, sep 2018. ISSN 0360-0300. Citado na página 94.

QI, Z.; WANG, H.; LI, J.; GAO, H. Impacts of dirty data: an experimental evaluation. *ArXiv*, abs/1803.06071, 2018. Citado na página 84.

QI, Z.-X.; WANG, H.-Z.; WANG, A.-J. Impacts of dirty data on classification and clustering models: An experimental evaluation. *Journal of Computer Science and Technology*, v. 36, p. 806–821, 2021. Citado na página 84.

RASCHKA, S.; MIRJALILI, V. *Python Machine Learning - Second Edition*. [S.l.]: Packt Publishing, 2017. ISBN 9781787126022. Citado 7 vezes nas páginas 9, 50, 51, 52, 53, 54 e 55.

RATHEE, A.; MATHUR, R. P. Survey on decision tree classification algorithms for the evaluation of student performance. In: *BIOINFORMATICS 2013*. [S.l.: s.n.], 2013. Citado na página 46.

RICCIO, V.; JAHANGIROVA, G.; STOCCO, A.; HUMBATOVA, N.; WEISS, M.; TONELLA, P. Testing machine learning based systems: A systematic mapping. *Empirical Software Engineering*, 11 2020. Citado 11 vezes nas páginas 16, 17, 61, 68, 69, 71, 72, 73, 80, 82 e 92.

ROKACH, L.; MAIMON, O. Top-down induction of decision trees classifiers - a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, v. 35, n. 4, p. 476–487, 2005. Citado 2 vezes nas páginas 46 e 47.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, v. 3, n. 3, p. 210–229, 1959. Citado na página 43.

SANTOS, S. a.; SILVEIRA, B.; DURELLI, V.; DURELLI, R.; SOUZA, S.; DELAMARO, M. On using decision tree coverage criteria for testing machine learning models. In: _____. *Brazilian Symposium on Systematic and Automated Software Testing*. New York, NY, USA: Association for Computing Machinery, 2021. p. 1–9. ISBN 9781450385039. Citado 17 vezes nas páginas 9, 11, 17, 18, 20, 77, 78, 79, 80, 81, 89, 91, 95, 97, 109, 131 e 133.

SARKER, I. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, v. 2, 05 2021. Citado 6 vezes nas páginas 9, 16, 43, 44, 45 e 92.

SEGURA, S.; TOWEY, D.; ZHOU, Z. Q.; CHEN, T. Y. Metamorphic testing: Testing the untestable. *IEEE Software*, v. 37, n. 3, p. 46–53, 2020. Citado na página 75.

SHAHROKNI, A.; FELDT, R. A systematic review of software robustness. *Information and Software Technology*, v. 55, n. 1, p. 1–17, 2013. ISSN 0950-5849. Special section: Best papers from the 2nd International Symposium on Search Based Software Engineering 2010. Citado na página 70.

SHIVAHARE, B. D.; SUMAN, S.; CHALLAPALLI, S. S. N.; KAUSHIK, P.; GUPTA, A. D.; BIBHU, V. Survey paper: Comparative study of machine learning techniques and its recent applications. In: *2022 2nd International Conference on Innovative Practices in Technology and Management (ICIPTM)*. [S.l.: s.n.], 2022. v. 2, p. 449–454. Citado na página 16.

TELIKANI, A.; TAHMASSEBI, A.; BANZHAF, W.; GANDOMI, A. H. Evolutionary machine learning: A survey. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 54, n. 8, oct 2021. ISSN 0360-0300. Citado na página 44.

THOMAS, R.; GUPTA, R. A survey on machine learning approaches and its techniques:. In: *2020 IEEE International Students' Conference on Electrical,Electronics and Computer Science (SCEECS)*. [S.l.: s.n.], 2020. p. 1–6. Citado na página 44.

TIAN, Y.; PEI, K.; JANA, S.; RAY, B. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In: *Proceedings of the 40th International Conference on Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2018. (ICSE '18), p. 303–314. ISBN 9781450356381. Citado 4 vezes nas páginas 18, 87, 91 e 93.

TRIPATHY, P.; NAIK, K. *Software Testing and Quality Assurance: Theory and Practice*. [S.l.]: Wiley, 2011. ISBN 9781118211632. Citado 8 vezes nas páginas 21, 26, 28, 29, 30, 32, 33 e 34.

TóMASDóTTIR, K. F.; ANICHE, M.; DEURSEN, A. van. Why and how javascript developers use linters. In: *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. [S.l.: s.n.], 2017. p. 578–589. Citado na página 83.

VAROQUAUX, G.; BUITINCK, L.; LOUPPE, G.; GRISEL, O.; PEDREGOSA, F.; MUELLER, A. Scikit-learn: Machine learning without learning the machinery. *GetMobile: Mobile Comp. and Comm.*, Association for Computing Machinery, New York, NY, USA, v. 19, n. 1, p. 29–33, jun 2015. ISSN 2375-0529. Citado na página 58.

VERBRAEKEN, J.; WOLTING, M.; KATZY, J.; KLOPPENBURG, J.; VERBELEN, T.; RELLERMEYER, J. S. A survey on distributed machine learning. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 53, n. 2, mar 2020. ISSN 0360-0300. Citado na página 44.

WITTEN, I.; HALL, M.; FRANK, E.; HOLMES, G.; PFAHRINGER, B.; REUTEMANN, P. The weka data mining software: An update. *SIGKDD Explorations*, v. 11, p. 10–18, 11 2009. Citado na página 85.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M.; REGNELL, B.; WESSLÉN, A. *Experimentation in Software Engineering*. Germany: Springer, 2012. ISBN 978-3-642-29044-2. Citado 2 vezes nas páginas 110 e 129.

XIE, X.; HO, J. W. K.; MURPHY, C.; KAISER, G.; XU, B.; CHEN, T. Y. Testing and validating machine learning classifiers by metamorphic testing. *J. Syst. Softw.*, Elsevier Science Inc., USA, v. 84, n. 4, p. 544–558, apr 2011. ISSN 0164-1212. Citado na página 85.

YETURU, K. Chapter 3 - machine learning algorithms, applications, and practices in data science. In: Srinivasa Rao, A. S.; RAO, C. (Ed.). *Principles and Methods for Data Science*. [S.l.]: Elsevier, 2020, (Handbook of Statistics, v. 43). p. 81–206. Citado na página 43.

ZHANG, J. M.; HARMAN, M.; MA, L.; LIU, Y. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, v. 48, n. 1, p. 1–36, 2022. Citado 30 vezes nas páginas 9, 16, 17, 18, 42, 44, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 76, 80, 84, 87, 90, 91, 92, 93, 99 e 133.

ZHANG, M.; ZHANG, Y.; ZHANG, L.; LIU, C.; KHURSHID, S. Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems. In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2018. (ASE 2018), p. 132–142. ISBN 9781450359375. Citado 4 vezes nas páginas 18, 88, 91 e 93.