UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

# Seleção de Dados de Teste com Base na Aplicação de Testes de Mutação a Modelos de Árvore de Decisão

**Beatriz Nogueira Carvalho da Silveira**

Dissertação de Mestrado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP
SÃO CARLOS

**Beatriz Nogueira Carvalho da Silveira**

# Seleção de Dados de Teste com Base na Aplicação de Testes de Mutação a Modelos de Árvore de Decisão

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestra em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientadora: Profa. Dra. Simone do Rocio Senger de Souza

**USP – São Carlos**
**Junho de 2024**

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

S587s

Silveira, Beatriz Nogueira Carvalho da
Seleção de Dados de Teste com Base na Aplicação de Testes de Mutação a Modelos de Árvore de Decisão / Beatriz Nogueira Carvalho da Silveira; orientadora Simone do Rocio Senger de Souza. -- São Carlos, 2024.
95 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2024.

1. Teste de Software. 2. Aprendizagem de Máquina. 3. Teste de Mutação. 4. Árvore de decisão. 5. Inteligência Artificial. I. Souza, Simone do Rocio Senger de, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

**Beatriz Nogueira Carvalho da Silveira**

# Test Data Selection Based on Applying Mutation Testing to Decision Tree Models

Dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Master in Science. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Profa. Dra. Simone do Rocio Senger de Souza

**USP – São Carlos**
**June 2024**

# AGRADECIMENTOS

Queria agradecer em especial à minha família que sempre esteve presente em minha vida. À minha mãe, Antônia Valéria, uma pessoa animada, que sempre me motivou, acreditando em meu potencial mesmo quando eu não acreditava. Ao meu pai, Maurício, um exemplo de resiliência, esforço e responsabilidade. Às minhas irmãs, Bárbara, Mariana e Marília, que são minhas melhores amigas e companheiras de correções ortográficas, minhas três amoras queridas. Às minhas avós Maria das Neves e Floriza pelo cuidado e carinho ao longo dos nossos anos juntas. Ao meu marido, Keslley, por ser sempre esse parceiro de todas as horas, me motivando e me apoiando.

Agradeço à minha orientadora, Simone, uma excelente professora e referência na área. A TI precisa de mais mulheres como a senhora. Ao meu coorientador, Delamaro, pelas reuniões cheias de novas ideias. Ao meu colega Sebastião, que iniciou o mestrado comigo e que trabalhou forma muito colaborativa. Aos demais membros do grupo de pesquisa Vinícius e Rafael, aprendi muito com vocês, principalmente com a forma brilhante com que o Vinicius escreve. E ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) por ter financiado esta pesquisa.

*“Alice: This is impossible.*
*Mad Hatter: Only if you believe it is.”*
*(Lewis Carroll)*

# RESUMO

SILVEIRA, B. N. C. **Seleção de Dados de Teste com Base na Aplicação de Testes de Mutação a Modelos de Árvore de Decisão**. 2024. 95 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2024.

O teste de software é crucial para garantir a qualidade do software, verificando se ele se comporta conforme o esperado. Esta atividade desempenha um papel crucial na identificação de defeitos desde os estágios iniciais do processo de desenvolvimento. O teste de software é especialmente essencial em sistemas complexos ou críticos, como aqueles que utilizam técnicas de Aprendizagem de Máquina (AM), uma vez que os modelos podem apresentar incertezas e erros que afetam sua confiabilidade. Para enfrentar os desafios inerentes ao teste de sistemas baseados em AM, este trabalho propõe critérios de teste fundamentados em modelos de árvores de decisão. Desta forma, este projeto investiga a aplicação do teste de mutação à estrutura interna de modelos de árvores de decisão, visando apoiar a seleção de entradas de teste e, assim, aprimorar a validação de aplicações de AM. Nossa abordagem introduz pequenas modificações no modelo de árvore de decisão, resultando no que denominamos "árvores mutantes". As árvores mutantes geradas servem como referência para a seleção de conjuntos de dados de teste capazes de identificar classificações incorretas em modelos de AM. Para avaliar nossa abordagem, conduzimos um experimento que abrangeu 16 conjuntos de dados. Neste experimento, avaliamos a eficácia dos conjuntos de teste gerados por nossa abordagem em comparação com a seleção aleatória de dados de teste. Avaliamos o desempenho usando modelos de AM adicionais e aplicamos métricas amplamente aceitas para avaliar modelos de classificação em AM. Os resultados do experimento indicam que a abordagem proposta tem o potencial de aprimorar com sucesso a seleção de dados de teste para a validação de aplicações de AM.

**Palavras-chave:** Teste de Software, Aprendizagem de Máquina, Árvore de decisão, Teste de Mutação.

# ABSTRACT

SILVEIRA, B. N. C. **Test Data Selection Based on Applying Mutation Testing to Decision Tree Models**. 2024. 95 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2024.

Software testing is crucial to ensuring the quality of the software by checking that it behaves as expected. This activity plays a crucial role in identifying defects from the early stages of the development process. Software testing is especially essential in complex or critical systems, such as those using Machine Learning (ML) techniques, as models can introduce uncertainties and errors that affect their reliability. This work proposes testing criteria for ML based on decision tree models. These criteria aim to address the challenges inherent in testing ML-based systems. In this direction, this work investigates the definition of mutation testing to be applied in decision tree models. Mutant operators are adapted to the decision tree model where minor modifications to the decision tree result in "mutant trees," which are references for selecting test data to validate ML applications. We conducted an experimental study to evaluate our proposed testing criteria compared to random testing. A set of 16 ML applications was used in this study, where a set of metrics to evaluate ML classification models was used to compare the testing criteria. The experiment results indicate that the proposed testing criteria have the potential to successfully improve the selection of test data for the validation of ML applications, offering initial studies to evaluate software testing criteria adapted for ML problems.

**Keywords:** Software Testing, Machine Learning, Decision Tree, Mutation Testing.

# LISTA DE ILUSTRAÇÕES


Figura 1 – Exemplificação do processo de criação de mutantes e seleção de casos de teste. 25
Figura 2 – Operadores de Mutação definidos para a Linguagem C . . . . . 31
Figura 3 – Exemplo de construção de Árvore de Decisão . . . . . 37
Figura 4 – Exemplo funcionamento KNN . . . . . 39
Figura 5 – Matriz de Confusão . . . . . 40
Figura 6 – Modelo de árvore de decisão gerado a partir do *dataset* 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. . 51
Figura 7 – Exemplo de mutações para o operador ORRN no nó nº 2. . . . . 54
Figura 8 – Exemplo de mutações para o operador de Cccr no nó nº 0. . . . . 55
Figura 9 – Comparação dos resultados obtidos para as métricas avaliadas com DTC, BVA e *k-fold cross validation* . . . . . 65
Figura 10 – Ilustração do fluxo do experimento . . . . . 67
Figura 11 – Comparação dos resultados obtidos para as métricas avaliadas com DTMT e *k-fold cross validation* . . . . . 73
Figura 12 – Visão geral da eficácia das quatro abordagens com a mediana para cada métrica. 80

# LISTA DE ALGORITMOS

Algoritmo 1 – Algoritmo de Euclides . . . . . 32
Algoritmo 2 – Algoritmo de Euclides - Exemplo de Mutante . . . . . 32

# LISTA DE TABELAS


Tabela 1 – *Dataset* de Exemplo Árvore - 1. . . . . 35
Tabela 2 – *Dataset* de Exemplo Árvore - 1.1. . . . . 35
Tabela 3 – *Dataset* de Exemplo Árvore - 1.2. . . . . 36
Tabela 4 – *Dataset* de Exemplo Árvore - 1.1.1. . . . . 36
Tabela 5 – *Dataset* de Exemplo Árvore - 1.1.2. . . . . 36
Tabela 6 – *Dataset* de Exemplo Árvore - 1.2.1. . . . . 36
Tabela 7 – *Dataset* de Exemplo Árvore - 1.2.2. . . . . 36
Tabela 8 – Exemplo de caso de teste DTC que satisfaz o requisito de teste de cobertura do nó folha 16. . . . . 52
Tabela 9 – Exemplo de caso de teste BVA que satisfaz o requisito de teste de cobertura do nó folha 16. . . . . 52
Tabela 10 – Visão geral dos *datasets* usados no experimento. . . . . 60
Tabela 11 – Número de nós nos modelos de árvore de decisão resultantes e número de casos de teste gerados a partir da aplicação dos critérios DTC e BVA a esses modelos. . . . . 62
Tabela 12 – Resultados da aplicação de dados de teste DTC e BVA a modelos k-NN gerados a partir de cada *dataset*. . . . . 63
Tabela 13 – Resumo dos resultados dos testes estatísticos entre DTC, BVA e *k-fold cross validation*. . . . . 64
Tabela 14 – Visão geral dos mutantes e do conjunto de teste gerado no experimento DTMT. 81
Tabela 15 – Resultados obtidos pelas métricas avaliadas para a abordagem DTMT e *k-fold cross validation*. . . . . 82
Tabela 16 – Resumo dos resultados dos testes estatísticos para DTMT e 10-fold cross validation. . . . . 82
Tabela 17 – Visão geral dos dados gerados no experimento. . . . . 83
Tabela 18 – Número de nós nos modelos de árvore de decisão resultantes e número de casos de teste gerados pela aplicação dos critérios DTC e BVA. . . . . 83
Tabela 19 – Visão geral dos dados gerados no experimento. . . . . 84
Tabela 20 – Resultados das métricas avaliadas para as abordagens DTC, BVA, DTMT approach e 10-fold cross-validation. . . . . 85
Tabela 21 – Resumo dos resultados dos testes estatísticos para DTC, BVL e 10-fold cross-validation. . . . . 86
Tabela 22 – Resumo dos resultados dos testes estatísticos para BVA e DTMT. . . . . 86

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| ADMEs | adequação de desempenho e métricas de erro |
| AM | Aprendizagem de Máquina |
| BVA | Boundary Value Analysis |
| CART | Árvores de classificação e regressão |
| Cccr | Operador de substituição de Constante por Constante |
| DTC | Decision Tree Coverage |
| DTMT | Decision Tree Mutation Testing |
| GFC | Grafos de Fluxo de Controle |
| GQM | Goal/Question/Metric |
| ICMC | Instituto de Ciências Matemáticas e de Computação |
| ID3 | Iterative Dichotomiser 3 |
| IEEE | Institute of Electrical and Electronics Engineers |
| JSERD | Journal of Software Engineering Research and Development |
| KNN | K-Nearest Neighbors |
| ORRN | Operador de Mutação Relacional |
| PMML | Predictive Model Markup Language |
| QP | Questão de Pesquisa |
| USP | Universidade de São Paulo |
| VV&T | Verificação, Validação e Teste de Software |

# SUMÁRIO


1 INTRODUÇÃO . . . . . 23
1.1 Objetivo . . . . . 25
1.2 Resultados Obtidos . . . . . 26
1.3 Organização do Trabalho . . . . . 26

2 FUNDAMENTAÇÃO TEÓRICA . . . . . 27
2.1 Teste de Software . . . . . 28
*2.1.1 Teste Baseado em Defeito* . . . . . *30*
2.2 Algoritmos de Aprendizagem de Máquina . . . . . 33
*2.2.1 Algoritmo de Árvore de Decisão de Aprendizado* . . . . . *34*
*2.2.1.1 Algoritmos de Árvore de Decisão de Aprendizado* . . . . . *37*
*2.2.2 Algoritmo K-Nearest Neighbors* . . . . . *38*
*2.2.3 Métricas para Avaliação de Modelos de AM* . . . . . *40*
*2.2.4 Considerações Finais* . . . . . *42*

3 TESTE DE SOFTWARE PARA ALGORITMOS DE APRENDIZAGEM DE MÁQUINA . . . . . 43
3.1 Trabalhos Relacionados . . . . . 45
3.2 Considerações Finais . . . . . 47

4 TESTE DE SOFTWARE BASEADO EM ÁRVORE DE DECISÃO . 49
4.1 Critérios Baseados em Árvores de Decisão . . . . . 50
4.2 Teste de Mutação Aplicado em Árvore de Decisão . . . . . 52
*4.2.1 Teste de Mutação em Árvores de Decisão* . . . . . *53*
4.3 Considerações Finais . . . . . 56

5 AVALIAÇÃO EXPERIMENTAL . . . . . 57
5.1 Configuração do Experimento sobre Critérios de Cobertura de Árvores de Decisão . . . . . 57
*5.1.1 Escopo* . . . . . *58*
*5.1.2 Formulação de hipóteses* . . . . . *58*
*5.1.3 Instrumentação* . . . . . *59*
*5.1.4 Execução* . . . . . *60*

5.2 Resultados do Experimento sobre Critérios de Cobertura de Árvores de Decisão . . . 61
*5.2.1 Tabela de Características* . . . *62*
*5.2.2 Teste de hipóteses* . . . *63*
*5.2.3 Discussão dos Resultados do Experimento sobre Critérios de Cobertura de Árvores de Decisão* . . . *64*
5.3 Configuração do Experimento sobre Teste de Mutação aplicado a árvores de Decisão . . . 65
*5.3.1 Escopo* . . . *66*
*5.3.2 Formulação de Hipóteses* . . . *66*
*5.3.3 Instrumentação* . . . *68*
*5.3.4 Execução* . . . *69*
5.4 Resultados do Experimento sobre Teste de Mutação aplicado a árvores de Decisão . . . 70
*5.4.1 Características dos Objetos* . . . *70*
*5.4.2 Hipóteses de Teste* . . . *71*
*5.4.3 Discussão dos Resultados do Experimento sobre Teste de Mutação aplicado a árvores de Decisão* . . . *72*
5.5 Configuração do Experimento que compara DTC, BVA e DTMT . 73
*5.5.1 Escopo* . . . *74*
*5.5.2 Formulação de Hipóteses* . . . *74*
*5.5.3 Instrumentação e Execução* . . . *75*
5.6 Resultados do Experimento que Compara DTC, BVA e DTMT . . 75
*5.6.1 Critérios de cobertura da árvore de decisão: características dos objetos* *75*
*5.6.2 DTMT: Características dos Objetos* . . . *76*
*5.6.3 Teste de Hipóteses* . . . *76*
5.7 Ameaças à Validade . . . 78
5.8 Considerações Finais . . . 79

6 CONCLUSÃO . . . 87
6.1 Principais Contribuições . . . 87
*6.1.1 Publicações Científicas* . . . *88*
6.2 Trabalhos Futuros . . . 88

REFERÊNCIAS . . . 91

CAPÍTULO

1

# INTRODUÇÃO

Nos últimos anos, juntamente com uma explosão na disponibilidade de dados, os algoritmos de Aprendizagem de Máquina (AM) vêm sendo usados como uma alternativa promissora às abordagens tradicionais, solucionando problemas de domínios diversos (Durelli *et al.*, 2019; Aniche *et al.*, 2020). A crescente popularidade dos algoritmos de AM provoca uma questão importante, alvo da atenção de pesquisadores: a avaliação e melhoria da qualidade e da confiabilidade das soluções baseadas em AM (BRAIEK; KHOMH, 2020; Zhang *et al.*, 2020).

Antes de implantar modelos de AM, os testadores precisam abordar esses modelos como software convencional e realizar esforços de teste para descobrir possíveis problemas. Esta fase de teste é uma parte essencial do processo de treinamento e implantação. No entanto, apesar da sua importância, descobrir problemas em aplicações baseadas em AM apresenta desafios significativos (Marijan; Gotlieb; Kumar Ahuja, 2019). Muitos desafios surgem devido às diferenças inerentes entre o software tradicional e o software baseado em AM, que é mais orientado estatisticamente e inerentemente menos determinístico. O comportamento do software baseado em AM é derivado de um processo baseado em dados: Os algoritmos de AM são usados para modelar e compreender conjuntos de dados (*datasets*) complexos, e o processo de derivação da lógica de decisão dos dados por meio do treinamento pode variar muito, dependendo do algoritmo de AM específico empregado (Zhang *et al.*, 2020).

Para enfrentar os desafios associados ao teste de sistemas baseados em AM, os pesquisadores começaram a adaptar métodos e abordagens comprovados do teste de software tradicional, como teste estrutural, teste metamórfico e teste de mutação. É importante destacar que falamos de adaptações porque a maior parte das técnicas de teste atuais não é diretamente aplicável às representações de software de modelos de AM. Inspirando-se nas abordagens de teste estrutural, os pesquisadores têm investigado como a estrutura interna dos modelos de AM pode ser usada para gerar casos de teste (PEI *et al.*, 2019). Seguindo nessa linha, em um estudo preliminar conduzido em colaboração com outros pesquisadores, foi realizada uma adaptação de conceitos

do teste de software estrutural, explorando a estrutura interna de modelos de árvores de decisão: foi proposto critérios de teste de cobertura de árvore de decisão (SANTOS *et al.*, 2021).

Árvore de decisão é uma família de algoritmos de AM que produz modelos gráficos que se assemelham a árvores, tornando-os facilmente compreensíveis para humanos. Esses algoritmos particionam o espaço do preditor em partes únicas e não sobrepostas. Os critérios utilizados para este particionamento estão incorporados em um modelo que reflete uma estrutura em árvore. A representação gráfica do modelo de árvore resultante se assemelha a uma estrutura semelhante a um fluxograma, onde os nós folha indicam o resultado de uma série de seções, enquanto os nós internos representam decisões de ramificação (JAMES *et al.*, 2013). Assim, os nós folhas representam resultados possíveis, enquanto os nós internos descrevem as relações entre as características.

Assim como outros modelos que representam sistemas de software (por exemplo, gráficos de fluxo de controle), a representação gráfica de árvores de decisão pode ser empregada para teste de software. Logo, a principal ideia dos critérios de cobertura de árvore de decisão é usar a estrutura interna de tais modelos (em forma de árvore) e as informações nos nós de decisão para guiar a seleção de entradas de teste. Assim, a amostragem de entradas de teste que aumentam a cobertura dos nós de decisão e das folhas resulta em casos de teste mais eficazes. De acordo com os resultados do experimento do trabalho preliminar, esses critérios de adequação podem ser usados para amostrar dados de teste mais eficazes do que dados de treinamento selecionados aleatoriamente. Motivados por tais resultados, prosseguimos com a exploração da estrutura dos modelos de árvores de decisão, agora incorporando os benefícios da técnica de teste de mutação.

Nas configurações convencionais de teste de software, o teste de mutação envolve fazer repetidamente alterações sutis na forma de pequenos desvios sintáticos (ou seja, *mutações*) no software que está sendo testado. O objetivo disso é examinar se algum caso de teste falha quando executado no programa mutado. Um programa mutado é chamado de *mutante* e é considerado *morto* quando pelo menos um caso de teste detecta a mutação introduzida (ou seja, falha devido à mutação). Quando um caso de teste é sensível o suficiente para matar um mutante, a principal implicação é que o conjunto de teste é eficaz. Assim, o objetivo principal do teste de mutação é avaliar a qualidade do conjunto de teste, promovendo a criação de casos de teste que revelam problemas decorrentes dos mutantes (ou seja, os pequenos desvios sintáticos feitos no programa em teste) (PAPADAKIS *et al.*, 2019). O teste de mutação tem atraído atenção significativa da pesquisa desde o seu início. Ele tem sido empregado para diversos propósitos e, nos últimos anos, avanços significativos foram feitos para aumentar sua eficiência computacional, tornando-o uma abordagem valiosa para testar sistemas de software.

Visando combinar os benefícios do teste de mutação com a exploração da estrutura da árvore de decisão, que se mostrou muito promissora, propomos uma abordagem que explora o teste de mutação para criar conjuntos de teste eficazes para descobrir falhas em modelos de AM. Especificamente, a abordagem proposta está centrada na mutação de alguns elementos

da estrutura interna dos modelos de árvore de decisão, permitindo a seleção de entradas (ou seja, casos de teste) que sejam sensíveis o suficiente para revelar discrepâncias nas previsões subsequentes (ou seja, resultados). A lógica por trás desta abordagem é explorar a forma como as decisões são codificadas em árvores de decisão e aplicar um subconjunto de operadores de mutação aos nós de decisão (ou seja, nós de ramificação). Em seguida, avaliar a adequação dos dados de teste de maneira semelhante à forma como o conjunto de teste é avaliado para programas convencionais ao aplicar teste de mutação. Mais especificamente, os modelos de árvore mutada servem como referência para a seleção de dados de teste mais eficazes, que também podem ser projetados para descobrir efetivamente classificações incorretas em modelos de AM.

De forma simplificada, nossa abordagem está centrada na aplicação de teste de mutação a modelos de árvores de decisão: os nós de decisão são mutados por meio de operadores relacionais e de mutação constante. Em seguida, examinamos o *dataset* em busca de casos em que a classificação fornecida pelo modelo de árvore mutada divergiu daquela produzida pela árvore de decisão original. Este processo é executado na esperança de criar um conjunto de teste especificamente adaptado para eliminar os modelos de árvores mutantes. De maneira semelhante à análise de mutação em teste de softwares tradicionais, novos casos de teste são então gerados, aprimorando assim o *dataset* original. A Figura 1 ilustra esse processo. O conjunto de teste resultante pode então ser usado para avaliar outros modelos de AM.

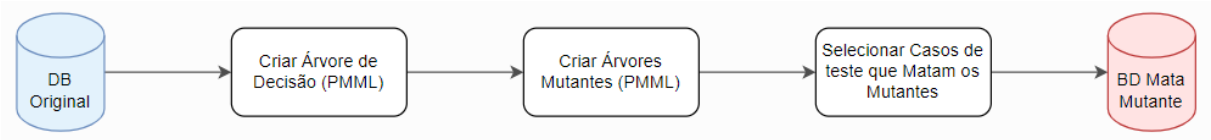



Figura 1 – Exemplificação do processo de criação de mutantes e seleção de casos de teste.

Conduzimos um estudo experimental para avaliar a eficácia dos dados de teste produzidos por nossa abordagem. Comparamos nossa abordagem com *k-fold cross validation*, uma técnica comum para avaliar modelos de AM, que seleciona dados de teste de forma aleatória. Exploramos se nossa abordagem poderia melhorar a escolha de um conjunto de dados de teste de alta qualidade.

## 1.1 Objetivo

O objetivo deste estudo é investigar a aplicação de técnicas de teste de mutação na estrutura interna de modelos de árvores de decisão, com o propósito de aprimorar a seleção de conjuntos de dados de teste destinados a aplicações de AM. Ademais, este trabalho busca servir como inspiração para futuras pesquisas voltadas à resolução dos desafios intrínsecos ao teste de sistemas que fazem uso do AM.

## 1.2 Resultados Obtidos

As contribuições apresentadas neste estudo abrangem os seguintes aspectos:

- Propomos uma abordagem que aplica teste de mutação a modelos de árvores de decisão. Esta abordagem utiliza operadores relacionais e de mutação constante para imitar falhas na representação interna de modelos de árvores de decisão;
- Realizamos a implementação da abordagem proposta e, por meio de um experimento, aplicamos essa abordagem em 12 *dataset* amplamente reconhecidos e utilizados em pesquisas anteriores.
- Demonstramos como essa abordagem pode ser usada para escolher amostras de dados de teste que sejam mais úteis do que dados de teste selecionados aleatoriamente. Aqui, a eficácia é definida pelo quanto os dados do teste podem diminuir as pontuações de previsão;
- Este trabalho representa uma contribuição significativa para o estado da arte no domínio de teste de algoritmos de AM, por meio da apresentação de uma abordagem eficaz na seleção de dados de teste.

## 1.3 Organização do Trabalho

Esta dissertação está organizada da seguinte forma. O capítulo 2 apresenta os conceitos e terminologias de teste de software e de algoritmos de aprendizagem de máquina. O Capítulo 3 exibe o que vem sendo discutido nesta área de pesquisa, bem como trabalhos relacionados. No Capítulo 4 são apresentadas as definições dos critérios de cobertura de árvore de decisão (desenvolvido por nosso grupo de pesquisa que se relaciona com o trabalho proposto), e as definições relacionada à abordagem de teste de mutação para árvores de decisão. Capítulo 5 apresenta as configurações e resultados dos experimentos realizados para avaliar as duas abordagens apresentadas. Por fim, no Capítulo 6 discute-se os resultados obtidos e a conclusão do trabalho.

CAPÍTULO

2

# FUNDAMENTAÇÃO TEÓRICA

A qualidade de software pode ser definida como o "grau de conformidade de um sistema, componente ou processo com as necessidades e expectativas de clientes ou usuários" (FILHO, 2009), ou o grau de conformidade do sistema de software com os seus requisitos. Atividades de Verificação, Validação e Teste de Software (VV&T) visam garantir a qualidade de software. Verificação é a confirmação, por exame e fornecimento de evidência objetiva, do atendimento aos requisitos especificados, ou seja, visa garantir que o produto foi construído de forma correta. Validação é a confirmação, por exame e fornecimento de evidência objetiva, de que os requisitos específicos para um determinado uso pretendido são atendidos, esta atividade tem como objetivo assegurar a consistência, completitude e correctitude do produto, buscando garantir que o mesmo corresponda aos requisitos de software (MALDONADO; JINO; DELAMARO, 2016). O teste é uma atividade que tem como objetivo examinar o comportamento do produto por meio de sua execução com a intenção de revelar erros.

As atividades de VV&T compõem atividades caracterizadas como estáticas e dinâmicas, cujo objetivo é avaliar os diversos artefatos de software. As atividades estáticas estão associadas a uma avaliação das propriedades de qualidade do produto e não implicam na execução do mesmo. Um exemplo de atividade estática é a revisão formal como por exemplo a inspeção, pois esse tipo de atividade não implica na execução do produto que está sendo validado como é o caso das atividades dinâmicas, que têm como principal exemplo, o teste de software.

O teste de software avalia o funcionamento do software executando-o em ambiente controlado, dessa forma busca evidências para conferir a satisfatibilidade dos requisitos do software e assim encontrar falhas. O teste de software normalmente é realizado em 4 etapas:

- a) Planejamento, na qual são definidos critérios, técnicas e ferramentas;
- b) Projeto de Casos de Teste, onde são criados/especificados os casos de teste;
- c) Execução, onde o programa é exercitado de acordo com os casos de teste;

- d) Encerramento, onde os resultados da execução são relatados.

Devido a um aumento expressivo no uso de algoritmos de AM nos últimos anos, avaliar a qualidade desses algoritmos atraiu a atenção de pesquisadores o que refletiu diretamente no crescente número de pesquisas em teste de software para algoritmos de AM. Porém, descobrir e corrigir falhas em modelos de AM ainda apresenta grandes desafios. Muitos desses desafios decorrem do fato de que o software tradicional, diferente de modelos de AM, não é orientado por estatísticas e dados.

Nas seções subsequentes são abordadas as terminologias pertinentes ao tópico deste trabalho, que englobam tanto o campo de teste de software quanto os algoritmos de AM.

## 2.1 Teste de Software

Teste de software compreende atividades que visam analisar a conformidade de um software com seus requisitos, a fim de identificar a presença de falhas e divergências com a especificação de requisitos. As atividades de teste de software são capazes de mostrar a existência de falhas, não podendo ser aplicadas para aferir sua ausência (AMMANN; OFFUTT, 2016). Alguns dos principais conceitos de teste de software são os relacionados à taxonomia de defeitos, que incluem: engano, defeito, erro e falha.

Para definir a taxonomia de defeitos foi adotado como referência o Glossário de Padrões do *Institute of Electrical and Electronics Engineers (IEEE)* (IEEE. . . , 1990). Nele tem-se que um engano introduz um defeito no software, o defeito, quando ativado, pode produzir um erro, e o erro, se propagado até a saída do software, constitui uma falha. Dessa forma tem-se as seguintes definições:

- Engano *(Mistake)* = Ação humana que introduz um defeito no software: ação incorreta tomada pelo programador.
- Defeito *(Fault)* = Passo, processo ou definição de dados incorreta, incompleta, ausente ou extra que, ao ser executada, pode produzir um erro no programa: instrução ou comando incorreto.
- Erro *(Error)* = Diferença entre o valor obtido e o valor esperado, ou seja, qualquer estado na execução do programa, intermediário ou final, que seja inconsistente: erro computacional ou erro de domínio.
- Falha *(Failure)* = Evento notável em que o sistema viola suas especificações: produção de uma saída incorreta com relação à especificação.

O teste de software pode ser elaborado a partir de diferentes artefatos do software e podem ser aplicados em etapas distintas do desenvolvimento (AMMANN; OFFUTT, 2016).

Dependendo da etapa do processo de desenvolvimento de software em que o teste é realizado ele poderá ser classificado em diferentes níveis de teste. Abaixo estão as diferenças entre os níveis de teste:

- Teste de aceitação - avalia o software em relação aos requisitos.
- Teste de sistema - avalia o software em relação ao projeto de arquitetura.
- Teste de integração - avalia o software em relação ao projeto do subsistema.
- Teste de módulo - avalia o software em relação ao projeto detalhado.
- Teste de unidade - avalia o software em relação à implementação.

Avaliar a qualidade de uma solução não é uma tarefa trivial. Portanto, as atividades de teste precisam ser estruturadas com base em estratégias, técnicas e critérios adequados ao contexto e ao objetivo. A estratégia de teste desempenha um papel fundamental na etapa de planejamento de teste, pois define a abordagem geral e os objetivos das tarefas de teste. Nesse estágio, também são definidas as técnicas de teste, que determinam o nível de atuação do teste. É importante observar que existem diversas técnicas que se diferenciam pela fonte de informação usada para estabelecer os requisitos de teste. Cada técnica de teste está associada a critérios específicos, que sistematizam a atividade de teste de software. Os critérios desempenham um papel crucial, auxiliando na seleção dos casos de teste mais apropriados e na tomada de decisões sobre quando encerrar os testes.

As principais técnicas aplicadas atualmente são: Teste funcional, teste estrutural e teste baseado em defeitos. O uso da técnica de teste funcional tem como principal característica o fato da estrutura interna do software não ser considerada ou conhecida para geração dos casos de teste. Por outro lado, a técnica de teste estrutural possuí a geração dos casos de teste baseado na estrutura do código fonte da solução, sendo necessário acesso e conhecimento da estrutura interna. Teste baseado em defeitos busca elaborar casos de tests levando em consideração os tipos de erros mais comuns cometidos pelos desenvolvedores (AMMANN; OFFUTT, 2016).

Teste funcional ou teste de caixa preta são testes derivados de descrições externas do software, incluindo especificações, requisitos e design (AMMANN; OFFUTT, 2016). Nele o testador não tem acesso ao código fonte do software testado e elabora os casos de teste a partir de outras informações. Alguns critérios de teste funcional bem conhecidos são partição de equivalência (ou particionamento em classes de equivalência) e análise de valor limite. O critério de partição de equivalência consiste em dividir o domínio de entrada em classes de equivalência que, de acordo com a especificação do sistema, seus elementos devem ser tratados da mesma maneira, pois assume-se que os elementos desta classe se comportam de maneira similar e assim qualquer elemento é capaz de ser representante da classe. O critério de análise de valor limite parte do princípio que é bastante comum que os desenvolvedores de software cometam erros em

tomadas de decisões do programa, como por exemplo usar um > no lugar de >=. Assim, este critério consiste na utilização da técnica de partição de equivalência, testando os limites de cada classe de equivalência. Logo, os casos de teste são selecionados das fronteiras de cada classe, pois é nesses casos que os erros costumam ocorrer com mais frequência (PRESSMAN, 1997).

Teste estrutural ou teste de caixa branca são testes derivados do código-fonte interno do software, especificamente incluindo condições individuais e instruções (AMMANN; OFFUTT, 2016). Neste tipo de teste é comum a representação gráfica do código em modelos, como em Grafos de Fluxo de Controle (GFC). A partir dos grafos são definidos os requisitos de teste, que podem ser orientado ao fluxo de controle ou ao fluxo de dados, isso depende do critério de teste selecionado, os mais comuns são: o critério de cobertura de nós que exige que o conjunto de casos de teste seja capaz de exercitar todos os nós do grafo; critério de cobertura de arestas que exige que o conjunto de casos de teste seja capaz de exercitar todas as arestas do grafo e o critério de cobertura de pares de arestas que exige que o o conjunto de casos de teste seja capaz de exercitar todos os pares de arestas (um caminho que contenha aresta - nodo - aresta subsequente) do grafo.

Os conceitos relacionados à técnica de teste baseada em defeito, em particular o critério de teste de mutação, serão abordados em detalhes na subseção a seguir por serem de fundamental importância no contexto desta dissertação.

### 2.1.1 Teste Baseado em Defeito

O teste baseado em defeito leva em consideração os tipos de defeitos mais frequentes cometidos pelos desenvolvedores e a partir deles elaboram os casos de teste. A técnica de teste baseado em defeito mais comum é análise de mutantes (DEMILLO; LIPTON; SAYWARD, 1978). A técnica de análise de mutantes ou teste de mutação identifica os desvios sintáticos mais recorrentes e, aplicando pequenas alterações no código do software em teste, incentiva a elaboração de casos de teste que apresentem erros oriundos destas alterações, visando avaliar a qualidade de um conjunto de casos de teste (MALDONADO *et al.*, 2016).

A aplicação do critério da análise de mutantes envolve as seguintes atividades: execução do programa com um conjunto de casos de teste, geração dos mutantes, execução dos mutantes com os casos de teste e análise dos mutantes. Um programa é testado com um conjunto de casos de teste, se o programa funciona corretamente, então o programa sofre pequenas perturbações, gerando mutantes que são executados com os mesmos casos de teste. Caso o comportamento do mutante do programa seja diferente do programa original, então esse mutante é dito *morto* (DEMILLO; LIPTON; SAYWARD, 1978).

Os mutantes são gerados a partir de operadores de mutação que são específicos para uma linguagem de programação ou técnica de especificação e procuram sintetizar os enganos mais comuns a linguagem ou técnica. Assim, operadores de mutação são regras que definem as

alterações que devem ser aplicadas no programa em teste (DEMILLO; LIPTON; SAYWARD, 1978). Pois os operadores de mutação são construídos para induzir mudanças sintáticas simples com base nos erros típicos cometidos pelos programadores ou para forçar determinados objetivos de teste como, por exemplo, executar cada aresta do programa (AMMANN; OFFUTT, 2016).

Um excelente exemplo de definição de operadores de mutação pode ser observado na definição de operadores mutação para a Linguagem de Programação C (AGRAWAL *et al.*, 1989). Nele, os autores apresentam uma ampla variedade de operadores que são divididos em: mutações de instrução; mutações de operador; mutações de variável e mutações constantes. A Figura 2 apresenta alguns exemplos dos operadores definidos.

Figura 2 – Operadores de Mutação definidos para a Linguagem C

| Name | Domain | Range | Example |
|---|---|---|---|
| OAAA | Arithmetic assignment | Arithmetic assignment | a+= b → a -= b |
| OAAN | Arithmetic | Arithmetic | a +b → a * b |
| OBBA | Bitwise assignment | Bitwise assignment | a &= b → a\|= b |
| OBBN | Bitwise | Bitwise | a & b → a\| b |
| OLLN | Logical | Logical | a && b → a \|\|b |
| ORRN | Relational | Relational | a < b → a <= b |
| OSSA | Shift assignment | Shift assignment | a <<= b → a >>= b |
| OSSN | Shift | Shift | a << b → a>> b |

(AGRAWAL *et al.*, 1989)

Nosso trabalho foi inspirado nessa definição, portanto, faremos uso de mutação de operadores e de constantes. Logo, utilizamos o Operador de Mutação Relacional (ORRN) e o Operador de substituição de Constante por Constante (Cccr). A definição de ambos são apresentadas abaixo:

1. Operador ORRN: este operador substitui toda ocorrência de um operador relacional ($<$, $>$, $<=$, $>=$, $==$ $!=$) por outro possível operador relacional. Exemplo: original: $a < b$; Mutante: $a <= b$.

2. Operador Cccr: este operador substitui ocorrências de uma constante por outra constante utilizada no programa em teste. Exemplo: Dado um conjunto de constantes: $[7.5, 2.3, 3.14]$; original: $a = 7.5$; Mutante: $a = 3.14$.

Para exemplificar foi criado um código mutante para o algoritmo de Euclides (Algoritmo 2). Nele foi feita uma pequena alteração na linha de código 3, no qual o número 0 é substitui pelo número 1. Tal mudança seria capaz de provocar um resultado diferente do esperado no caso de teste TC.1 pois o código iria entrar em *loop* infinito.

Dentre as limitações do teste de mutação podemos destacar o problema de equivalência de mutantes, por se tratar de pequenas alterações no código fonte original, é comum que

**Algoritmo 1** – Algoritmo de Euclides

```
1: procedimento EUCLID(a, b)                    ▷ O maior divisor comum de a e b
2:     r ← a mod b
3:     enquanto r ≠ 0 faça                      ▷ Tem-se a resposta se r é 0
4:         a ← b
5:         b ← r
6:         r ← a mod b
7:     fim enquanto
8:     retorna b                                ▷ O maior divisor comum é b
9: fim procedimento
```

**Algoritmo 2** – Algoritmo de Euclides - Exemplo de Mutante

```
1: procedimento EUCLID(a, b)                    ▷ O maior divisor comum de a e b
2:     r ← a mod b
3:     enquanto r ≠ 1 faça▷ Linha em que ocorreu a mutação, agora tem-se a resposta se r é 1
4:         a ← b
5:         b ← r
6:         r ← a mod b
7:     fim enquanto
8:     retorna b                                ▷ O maior divisor comum é b
9: fim procedimento
```

pequenas alterações feitas por mutantes diferentes levem a um mesmo comportamento errôneo. Por exemplo, considerando a linha 3 do algoritmo de Euclides (Algoritmo 1), um mutante equivalente ao apresentado no algoritmo mutante (Algoritmo 2) seria utilizar r > 0. Para o caso de teste em que r fosse igual a 2 os mutantes iriam apresentar o mesmo comportamento.

É comum encontrar trabalhos que utilizam o teste de mutação em outros domínios, como em teste de integração (DELAMARO; MALDONADO, 1997). Também foram encontradas algumas abordagem que utilizam adaptações do teste de mutação para testar diferentes algoritmos de AM em diferentes contextos de aplicações (SHEN; WAN; CHEN, 2018; MA *et al.*, 2018; Hu *et al.*, 2019; JAHANGIROVA; TONELLA, 2020; RICCIO *et al.*, 2022; HUMBATOVA; JAHANGIROVA; TONELLA, 2021; LU *et al.*, 2022; TAMBON; KHOMH; ANTONIOL, 2023). Este assunto será abordado de forma mais detalhada no Capítulo 3, que apresentará com está sendo a utilização de teste de mutação em algoritmos de AM.

É interessante ressaltar que apesar de ser uma atividade consolidada há muitos anos dentro da engenharia de software, o teste de software continua passando por constantes evoluções e aprimoramentos, como automatização de casos de teste e geração automática de dados de teste (RODRIGUES *et al.*, 2018). Inclusive é comum encontrar trabalhos que utilizam AM na busca de melhorar as atividades de teste de software (Durelli *et al.*, 2019), por exemplo, para estimar a cobertura de caminho principal de um código de um conjunto de teste (SILVA; COTA, 2020).

## 2.2 Algoritmos de Aprendizagem de Máquina

Os expressivos avanços em ferramentas de coleta e armazenamento de dados permitiram o acúmulo diário de um grande volume de dados. Diante deste contexto, algoritmos de AM podem ser usados para auxiliar na exploração e análise desses dados, por meio da descoberta e aplicação de padrões na base de dados. De um modo geral, os algoritmos de AM oferecem técnicas estatísticas para aprender padrões de um *dataset* de treinamento fornecido, permitindo aplicar as funções aprendidas em novos dados. Essa capacidade de generalizar é frequentemente usada para fazer previsões sobre propriedades desconhecidas de dados observados.

Os algoritmos de AM são divididos em duas macrocategorias: aprendizado supervisionado e aprendizado não supervisionado. E dentro delas podemos encontrar classes como algoritmos de classificação, regressão, associação e agrupamento. Um modelo de AM é uma instância treinada de um algoritmo de AM específico.

A principal diferença entre algoritmos de aprendizado supervisionado e não supervisionado está na natureza dos dados de treinamento e nos objetivos do aprendizado. O aprendizado supervisionado requer dados de treinamento rotulados, com o objetivo de prever ou classificar com base em exemplos conhecidos. O aprendizado não supervisionado lida com dados não rotulados e se concentra na descoberta de padrões ou estruturas nos dados. Ambos os tipos de aprendizado têm suas próprias aplicações e são usados de acordo com os requisitos da tarefa em questão.

Os algoritmos de aprendizagem supervisionada podem ser divididos como algoritmos de classificação e de regressão que se diferenciam pela natureza das saídas desejadas e, consequentemente, aos objetivos das tarefas de aprendizado. Exemplos de algoritmos de classificação são: Árvores de Decisão, k-Nearest Neighbours, Florestas Aleatórias, Naive Bayes e Support Vector Machines (SVM). Exemplo de algoritmos de regressão: Linear Regression, Árvores de Regressão e Multilayer Perceptron. Alguns algoritmos, como Multilayer Perceptron ou k-Nearest Neighbors, podem ser usados para regressão e classificação com pequenas adaptações. Em problemas de classificação, o objetivo é atribuir categorias ou rótulos discretos às entradas, enquanto em problemas de regressão, o objetivo é prever valores numéricos contínuos.

Dentre os algoritmos de aprendizagem não supervisionados, os algoritmos de agrupamento são os mais amplamente adotados. A análise de *cluster* visa agrupar instâncias em *clusters* de forma que a similaridade entre duas instâncias seja máxima se elas pertencerem ao mesmo *cluster* e mínima caso contrário (Riccio *et al.*, 2020). Os algoritmos de agrupamento populares incluem agrupamento aglomerativo hierárquico, K-means++, K-medoids e Modelos de mistura gaussiana.

As Redes Neurais Artificiais (RNA) e as técnicas de aprendizagem associadas são chamadas de Aprendizado Profundo (DL). RNA são inspirados pelos neurônios no cérebro de animais e consistem em neurônios artificiais (simulados) para os quais uma ativação de saída é

calculada com base em uma agregação ponderada, muitas vezes não linear, das entradas. Elas são amplamente utilizadas em várias tarefas de aprendizado supervisionado, como classificação e regressão. No entanto, as RNA também podem ser aplicadas a tarefas não supervisionadas, como redução de dimensionalidade e geração de dados.

Um dos grande problemas presentes no teste para algoritmos de AM está nas restrições na capacidade de verificar se a saída do algoritmo está correta, ausência do oráculo de teste (MARIJAN; GOTLIEB; AHUJA, 2019). O oráculo de teste determina a saída ou o resultado esperado de um sistema ou componente durante a execução do teste de software (WEYUKER, 1982). Geralmente os algoritmos de AM são testados somente quanto a sua precisão na tarefa de classificação. Dessa forma, é crescente o número de pesquisas na área do teste de software para algoritmos de AM com o objetivo de desenvolver técnicas e aplicações para verificar a presença de falhas para garantir a qualidade dos algoritmos de AM (Zhang *et al.*, 2020; Riccio *et al.*, 2020; Sherin; khan; Iqbal, 2019).

Os algoritmos de AM têm suas raízes que remontam aos anos 1940 (SCHMIDHUBER, 2015) e, atualmente, estão sendo amplamente empregados em resposta à crescente disponibilidade de dados. Com o aumento da utilização desses algoritmos em contextos cada vez mais críticos, torna-se evidente a necessidade de aprimorar sua confiabilidade. Este estudo se concentra no desenvolvimento de uma abordagem que utiliza teste de mutação aplicado a estrutura de modelos de árvores de decisão a fim de selecionar casos de teste para a avaliação de outros modelos de AM, incluindo o modelo k-Nearest Neighbors. Nas subseções a seguir, descreveremos detalhadamente o funcionamento desses dois algoritmos de AM (árvores de decisão e k-Nearest Neighbors).

### 2.2.1 Algoritmo de Árvore de Decisão de Aprendizado

Árvore de decisão de aprendizado ou árvore de decisão indutiva ou *decision tree* envolve o uso de um *dataset* de treinamento para gerar uma árvore de decisão que classifique corretamente os dados de treinamento. Se o aprendizado estiver funcionando, a árvore de decisão classificará corretamente os novos dados de entrada (COPPIN, 2010).

O emprego de árvore de decisão é um modelo muito popular em mineração de dados. Muitos pesquisadores argumentam que as árvores de decisão são populares devido à sua simplicidade e transparência. As árvores de decisão são autoexplicativas; não há necessidade de ser um especialista em mineração de dados para seguir uma determinada árvore de decisão. Normalmente, as árvores de classificação são representadas graficamente como estruturas hierárquicas, o que as torna mais fáceis de interpretar do que outras técnicas. Se a árvore de classificação se tornar complicada (ou seja, tiver muitos nós), sua representação gráfica direta se tornará inútil. Para árvores complexas, outros procedimentos gráficos devem ser desenvolvidos para simplificar a interpretação (ROKACH; MAIMON, 2014).

Uma árvore de decisão pode ser usada para representar classificadores e modelos de regressão. Na pesquisa operacional as árvores de decisão referem-se a um modelo hierárquico de decisões e suas consequências. Quando uma árvore de decisão é usada para tarefas de classificação, é mais comumente chamada de árvore de classificação. Quando é usado para tarefas de regressão, é chamada de árvore de regressão (ROKACH; MAIMON, 2014).

Este trabalho faz uso de algoritmos de árvore de decisão de classificação. Árvores de decisão de classificação são usadas para classificar um objeto ou uma instância em um conjunto predefinido de classes com base nos valores de seus atributos. As árvores de classificação são frequentemente usadas em diferentes campos, como finanças, marketing, engenharia e medicina.

Para ilustrar o funcionamento da arvore de decisão será apresentado um exemplo, cujo objetivo é construir uma árvore de decisão para classificar se um estudante passa ou não em uma disciplina com base em três características: horas de estudo por semana, frequência nas aulas e nota anterior. O conjunto de dados fictício é apresentado na Tabela 1.

Tabela 1 – *Dataset* de Exemplo Árvore - 1.

| **Horas de Estudo** | **Frequência** | **Nota Anterior** | **Passa?** |
|---|---|---|---|
| 10 | Alta | Boa | Sim |
| 5 | Média | Ruim | Não |
| 8 | Alta | Média | Sim |
| 7 | Baixa | Média | Não |
| 4 | Média | Ruim | Não |
| 6 | Alta | Boa | Sim |

Passos para a Criação da Árvore de Decisão:

1. Escolha do Atributo de Divisão Inicial: O primeiro passo é escolher o atributo que melhor divide os dados em termos de previsibilidade da variável alvo (se o estudante passa ou não). Esta escolha é feita com base em métricas como Entropia e Ganho de Informação.

2. Divisão dos Dados: O atributo "Horas de Estudo"é escolhido para a primeira divisão, pois é o que melhor separa os exemplos.

   a. Horas de Estudo >= 7: Tabela 2.

Tabela 2 – *Dataset* de Exemplo Árvore - 1.1.

| **Horas de Estudo** | **Frequência** | **Nota Anterior** | **Passa?** |
|---|---|---|---|
| 10 | Alta | Boa | Sim |
| 8 | Alta | Média | Sim |
| 7 | Baixa | Média | Não |

   b. Horas de Estudo < 7: Tabela 3.

3. Continuação da Divisão: Para os subconjuntos gerados, repete-se o processo de escolha do melhor atributo para dividir os dados.

Tabela 3 – *Dataset* de Exemplo Árvore - 1.2.

| Horas de Estudo | Frequência | Nota Anterior | Passa? |
|---|---|---|---|
| 5 | Média | Ruim | Não |
| 4 | Média | Ruim | Não |
| 6 | Alta | Boa | Sim |

a. Para "Horas de Estudo >= 7": O próximo atributo escolhido é "Frequência".

   i. Frequência = Alta: Tabela 4.

Tabela 4 – *Dataset* de Exemplo Árvore - 1.1.1.

| Horas de Estudo | Frequência | Nota Anterior | Passa? |
|---|---|---|---|
| 10 | Alta | Boa | Sim |
| 8 | Alta | Média | Sim |

   ii. Frequência = Baixa: Tabela 5.

Tabela 5 – *Dataset* de Exemplo Árvore - 1.1.2.

| Horas de Estudo | Frequência | Nota Anterior | Passa? |
|---|---|---|---|
| 7 | Baixa | Média | Não |

b. Para "Horas de Estudo < 7": O próximo atributo escolhido é "Nota Anterior".

   i. Nota Anterior = Boa: Tabela 6.

Tabela 6 – *Dataset* de Exemplo Árvore - 1.2.1.

| Horas de Estudo | Frequência | Nota Anterior | Passa? |
|---|---|---|---|
| 6 | Alta | Boa | Sim |

   ii. Nota Anterior = Ruim: Tabela 7.

Tabela 7 – *Dataset* de Exemplo Árvore - 1.2.2.

| Horas de Estudo | Frequência | Nota Anterior | Passa? |
|---|---|---|---|
| 5 | Média | Ruim | Não |
| 4 | Média | Ruim | Não |

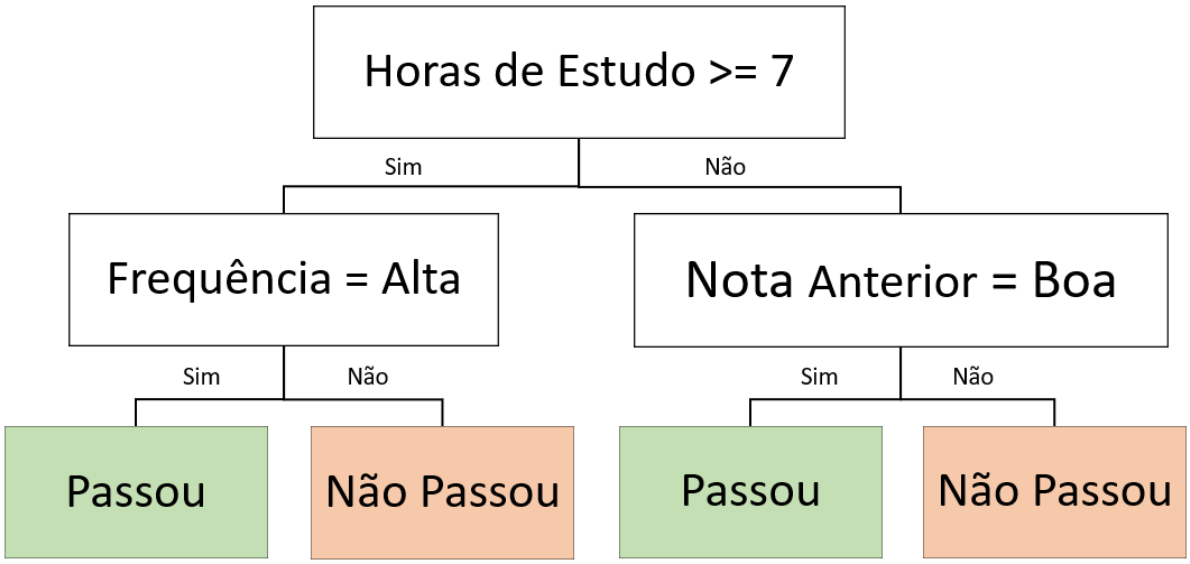
4. Construção da Árvore: A árvore de decisão é construída até que todas as folhas (decisões finais) sejam atingidas ou até que não haja mais atributos para dividir (ver imagem 3).

   Explicação da Lógica da Árvore:

   a. Primeiro Nível (Horas de Estudo):

      i. Se um estudante estuda 7 horas ou mais por semana, a próxima decisão será baseada na frequência nas aulas.

Figura 3 – Exemplo de construção de Árvore de Decisão




ii. Se um estudante estuda menos de 7 horas por semana, a próxima decisão será baseada na nota anterior.

b. Segundo Nível:

i. Para estudantes que estudam 7 horas ou mais por semana:

1. Se a frequência é alta, o estudante passa.
2. Se a frequência é baixa, o estudante não passa.

ii. Para estudantes que estudam menos de 7 horas por semana:

1. Se a nota anterior é boa, o estudante passa.
2. Se a nota anterior é ruim, o estudante não passa.

A árvore de decisão é uma ferramenta útil para a classificação, permitindo que dados complexos sejam segmentados em decisões claras e interpretáveis. Este exemplo prático ilustra como uma árvore de decisão pode ser criada usando um conjunto de dados fictício, com uma lógica clara e detalhada para cada nível de decisão.

#### *2.2.1.1 Algoritmos de Árvore de Decisão de Aprendizado*

O algoritmo de árvore de decisão mais conhecido é o Iterative Dichotomiser 3 (ID3), que foi desenvolvido por Ross Quinlan nos anos de 1980. O algoritmo cria uma árvore de decisão a partir do topo. Os nós são selecionados a partir das características do *dataset* de treinamento, sendo escolhido o que traz o máximo de informações, então transforma isso em perguntas. Essas perguntas formam os nós de decisão (ou nós intermediários) da árvore (COPPIN, 2010).

O método usado pelo ID3 para determinar qual característica usar a cada estágio da árvore de decisão é selecionar, em cada estágio, a característica que oferece o maior ganho

de informação. Ganho de informação é definido como redução de entropia. A entropia de um *dataset* de treinamento, S, é definida por:

$$H(S) = -p_1 \log_2 p_1 - p_0 \log_2 p_0$$

Onde p1 é definido como a proporção dos dados de treinamento que inclui exemplos positivos e p0 os exemplos negativos. A Entropia será 0 quando todos os exemplos forem positivos ou todos negativos. E alcançará valor máximo de 1 quando metade dos exemplos forem positivos e a outra metade negativos (COPPIN, 2010).

Logo, é encontrada para cada nó a característica que produzirá o maior ganho de informação. As árvores crescem até seu tamanho máximo e, em seguida, uma etapa de poda geralmente é aplicada para melhorar a capacidade da árvore de generalizar dados não vistos. Além do algoritmo de árvore de decisão ID3, também temos o C4.5, C5.0 e CART:

C4.5 é o sucessor do ID3 e removeu a restrição de que os recursos devem ser categóricos, definindo dinamicamente um atributo discreto (baseado em variáveis numéricas) que particiona o valor do atributo contínuo em um conjunto discreto de intervalos. C4.5 converte as árvores treinadas (isto é, a saída do algoritmo ID3) em conjuntos de regras se - então. A precisão de cada regra é então avaliada para determinar a ordem em que devem ser aplicadas. A poda é feita removendo a pré-condição de uma regra se a precisão da regra melhorar sem ela.

C5.0 é o lançamento da versão mais recente de Quinlan sob uma licença proprietária. Ele usa menos memória e cria conjuntos de regras menores do que C4.5 enquanto é mais preciso.

O Árvores de classificação e regressão (CART) é muito semelhante a C4.5, mas difere por oferecer suporte a variáveis de destino numéricas (regressão) e não calcular conjuntos de regras. O CART constrói árvores binárias usando o recurso e o limite que geram o maior ganho de informação em cada nó.

A biblioteca da linguagem Python que utilizamos em nosso experimento é o scikit-learn que usa uma versão otimizada do algoritmo CART; no entanto, até o presente momento, a implementação do scikit-learn não suporta variáveis categóricas[1].

### 2.2.2 Algoritmo K-Nearest Neighbors

O algoritmo dos vizinhos mais próximos ou K-Nearest Neighbors (KNN) é um exemplo de aprendizado baseado em instância. Métodos de aprendizado desse tipo armazenam os dados de treinamento e usam esses dados para determinar uma classificação para cada novo fragmento que for encontrado (COPPIN, 2010).

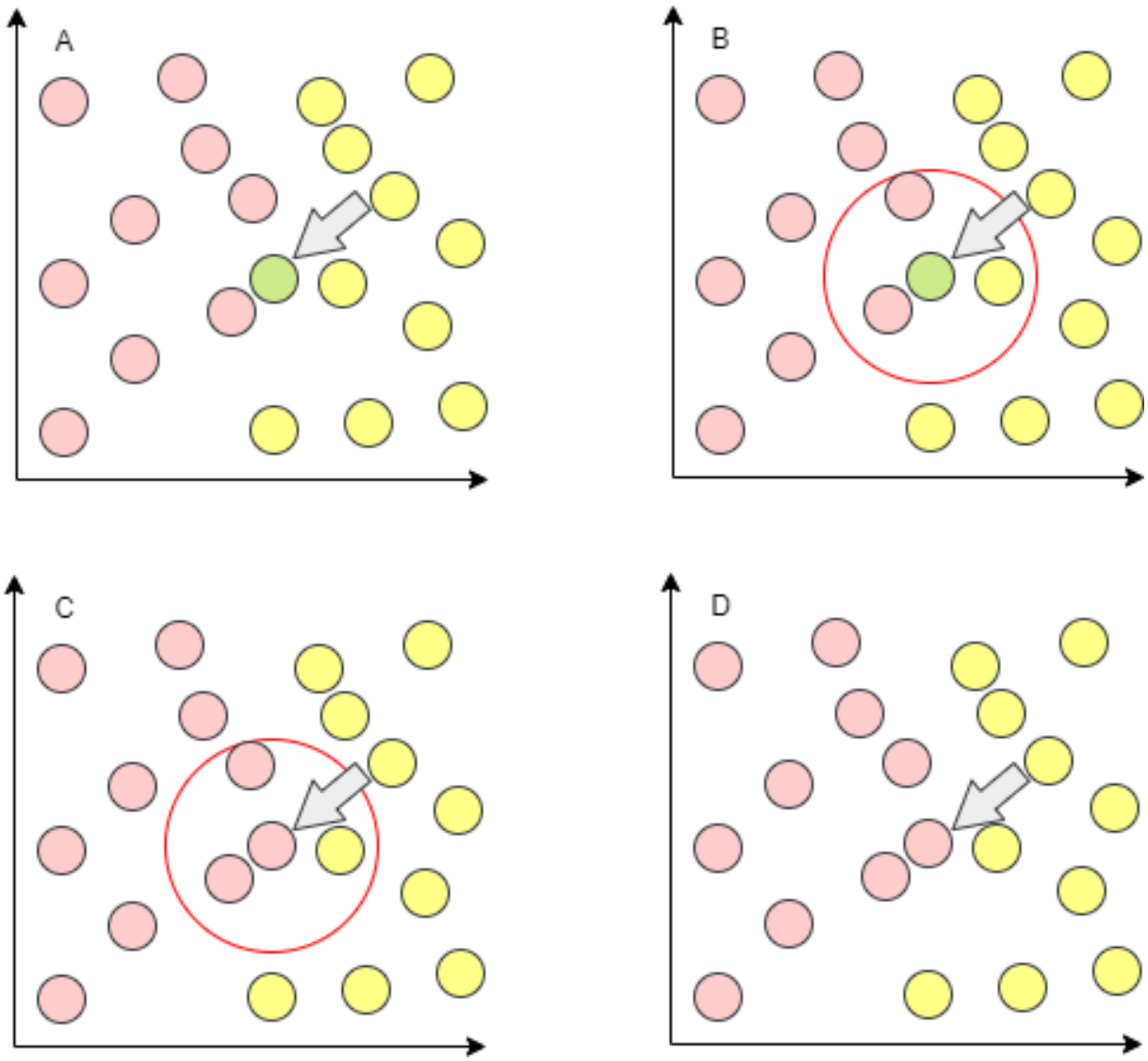
O KNN opera em situações onde cada instância pode ser definida por um vetor de N dimensões, onde N é o número de atributos usados para descrever cada instância. Geralmente

[1] Biblioteca python scikit-learn: https://scikit-learn.org/stable/modules/tree.html

esse tipo de algoritmo obtém a classificação dos K vizinhos mais próximos da instância a ser classificada e atribui a ela a classificação mais encontrada naquele grupo de vizinhos.

A Figura 4 ilustra o funcionamento do KNN em que k = 3. Nela, queremos classificar a bolinha verde em uma das duas classes de bolinha que temos, classe de bolinhas rosa e classe de bolinhas amarela. Primeiramente, na etapa A é feita a identificação da instância a ser classificada, em seguida, na etapa B, identificamos os k vizinhos mais próximos, para este exemplo, como adotamos k = 3 vamos selecionar os três vizinhos mais próximos. Na etapa C identificamos qual a classificação mais comumente encontrada nos vizinhos selecionados, aqui identificamos que a cor que mais aparece é a rosa. Por fim, na etapa D, a nova instância recebe a classificação de cor rosa que corresponde a cor da maioria de seus vizinhos mais próximos.

Figura 4 – Exemplo funcionamento KNN




Fonte: Elaborada pelo autor.

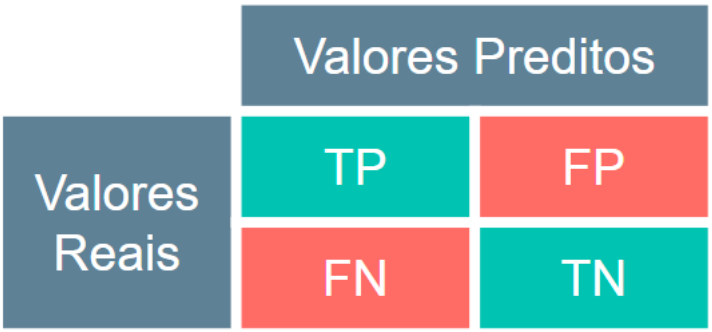
### 2.2.3 Métricas para Avaliação de Modelos de AM

Tradicionalmente, a adequação dos modelos de AM é estabelecida por meio de adequação de desempenho e métricas de erro (ADMEs). As medidas de desempenho e erro são elementos vitais no processo de avaliação de modelos e estruturas de AM, sendo definidas como construções lógicas e/ou matemáticas destinadas a medir a proximidade entre as observações reais e as esperadas (ou previstas). Em outras palavras, os ADMEs são utilizados para estabelecer uma compreensão de como as previsões de um modelo se comparam às observações reais (ou medidas), sendo que tais métricas geralmente se relacionam com a variação entre as observações previstas e medidas em termos de erros. (LASZCZYK; MYSZKOWSKI, 2019; CREMONESI; KOREN; TURRIN, 2010; SCHMIDT; LIPSON, 2010).

Na prática, é comum a utilização de uma ou a combinação de várias métricas para avaliar a adequação de um determinado modelo de AM. Recentemente, foi feito um trabalho que busca definir uma visão sistemática sobre em quais cenários as métricas específicas são mais adequadas para uso (NASER; ALAVI, 2021).

O desempenho de modelos de AM de classificação são geralmente listados em uma matriz de confusão (ilustrada na Figura 5). Esta matriz contém estatísticas sobre classificações reais e previstas e estabelece os fundamentos necessários para entender as medições de precisão para um classificador específico. Cada coluna nesta matriz significa instâncias previstas, enquanto cada linha representa instâncias reais (NASER; ALAVI, 2021).

Figura 5 – Matriz de Confusão




Fonte: Elaborada pelo autor.

Cada um desses campos corresponde a classificação correta ou a presença de falsos positivos e falsos negativos geradas por um modelo ao ser executado com dados diferentes dos dados de treinamento.

1. Verdadeiro positivo (true positive — TP): ocorre quando, no conjunto real, a classe que estamos buscando foi prevista corretamente.

2. Falso positivo (false positive — FP): ocorre quando, no conjunto real, a classe que estamos buscando prever foi prevista incorretamente.

3. Verdadeiro negativo (true negative — TN): ocorre quando, no conjunto real, a classe que não estamos buscando prever foi prevista corretamente.

4. Falso negativo (false negative — FN): ocorre quando, no conjunto real, a classe que não estamos buscando prever foi prevista incorretamente.

Dentre as métricas derivadas da Matriz de Confusão vamos passar pela definição de quatro: Precision (Precisão), Recall (Sensibilidade), Accuracy (Acurácia) e F1-score (Pontuação F1). Tais métricas serão aplicadas no nosso experimento. Legenda das siglas utilizadas nas fórmulas: P (denota o número reais positivos), N (denota o número de reais negativos).

1. Precision ou Valor Preditivo Positivo (PPV): corresponde à proporção de observações positivas que são verdadeiramente corretas. Seu valor ideal é 1, indicando que todas as observações positivas são verdadeiras, enquanto o pior valor é 0, indicando que nenhuma das observações positivas é verdadeira. Essa métrica é utilizada para avaliar o desempenho em dados categóricos.

$$PPV = TP/(TP+FP)$$

2. Recall ou Taxa de Verdadeiros Positivos ou Sensibilidade (TPR): Mede a proporção de positivos reais que foram identificados corretamente. Logo, esta medida descreve a capacidade do modelo de identificar corretamente os resultados positivos. É importante destacar que essas medidas não contabilizam resultados indeterminados. Além disso, essas medidas são frequentemente usadas em problemas de classificação binária e multiclasse, onde os dados são categorizados em duas ou mais classes.

$$TPR = (TP/P) = TP/(TP+FN)$$

3. Accuracy (ACC): indica a proporção de previsões corretas em relação ao número total de amostras. Essa métrica expressa quanto o modelo acertou das previsões possíveis, e funciona bem com dados categóricos. No entanto, a acurácia apresenta limitações, pois avalia o desempenho apenas em um único limite de classe e assume um custo igual para todos os tipos de erros.

$$ACC = (TP+TN)/(P+N) = (TP+TN)/(TP+TN+FP+FN)$$

4. F1-score (F1): é uma métrica que mostra o equilíbrio entre a precisão e recall. Ele é calculado como a média harmônica da precisão e da revocação e concentra-se em uma única classe. Essa métrica é útil quando há desigualdade entre as classes, tendendo a favorecer a classe majoritária. O F1-score funciona bem com dados categóricos, sendo uma medida comumente utilizada para avaliar a eficácia de algoritmos de classificação.

$$F1 = (2PPVTPR)/(PPV+TPR) = 2TP/(2TP+FP+FN)$$

As métricas de Precision, Recall, Accuracy e F1 desempenham um papel fundamental na avaliação de modelos de classificação no contexto de AM. Estas métricas são amplamente reconhecidas por fornecer medidas precisas da performance desses modelos. Utilizamos essas métricas para analisar e avaliar os resultados dos experimentos realizados nesta dissertação.

### 2.2.4 Considerações Finais

Este capítulo apresentou os principais conceitos relacionados ao teste de software e ao AM no contexto deste estudo de dissertação. Inicialmente, exploramos os conceitos essenciais de teste de software, discutindo tanto o teste estrutural quanto o funcional, uma vez que estão diretamente relacionados ao nosso estudo preliminar. Em seguida, aprofundamos o entendimento do teste de mutação, que é um dos focos do nosso estudo principal.

Em uma segunda seção, abordamos os conceitos fundamentais do AM, com ênfase em algoritmos de classificação, especificamente em Árvores de Decisão e KNN. Escolhemos destacar esses algoritmos devido ao papel central da Árvore de Decisão em nossa abordagem para criar mutantes que orientam a seleção de dados de teste. Além disso, o KNN desempenha um papel crucial na segunda fase dos experimentos, na qual avaliamos o desempenho de nossos conjuntos de teste em um modelo de Aprendizado de Máquina. Optamos por testar o KNN por ser um dos algoritmos de AM mais amplamente utilizados e de fácil compreensão.

Por fim, apresentamos as métricas que serão utilizadas nas análises estatísticas dos resultados dos experimentos. Após abordar os principais conceitos dentro do contexto deste estudo, procederemos para explorar os avanços no campo de teste para AM no Capítulo 3.

CAPÍTULO

3

# TESTE DE SOFTWARE PARA ALGORITMOS DE APRENDIZAGEM DE MÁQUINA

Para entender o estado da arte sobre o tema da pesquisa, foi realizada uma busca por estudos secundários sobre teste de software no contexto de AM. A busca foi realizada nos principais veículos de publicação da área de teste de software e/ou de inteligência artificial, inspirados no estudo feito por (Durelli *et al.*, 2019), que inclui periódicos, anais de conferências, *workshops* e simpósios. Assim, foram identificados três estudos secundários (Zhang *et al.*, 2020; Riccio *et al.*, 2020; Sherin; khan; Iqbal, 2019) que serão discutidos a seguir, bem como outros trabalhos relacionados com esta pesquisa que foram identificados através desses estudos utilizando técnica de *snowball*.

O trabalho (Zhang *et al.*, 2020) fornece uma pesquisa sobre teste de AM. O estudo abrange 128 artigos, apresenta definições e status de pesquisas sobre propriedades de teste, componentes de teste, fluxo de trabalho de teste e cenários de aplicação. O artigo também analisa as tendências relativas a *datasets* e foco de pesquisa. Além disso, resumiu os *datasets* mais usados em experimentos e as ferramentas/frameworks de teste de código aberto disponíveis e analisou a tendência de pesquisa, direções, oportunidades e desafios em teste de AM que, inclusive, mostraram-se promissoras em teste de AM para trabalhos futuros.

Em (Riccio *et al.*, 2020), os autores buscam obter uma melhor compreensão das tendências e do estado da arte em relação aos testes de AM, junto com as limitações que carregam. 70 artigos foram selecionados e analisados, mostrando que os problemas de teste mais relevantes abordados são o problema do oráculo (19%), seguido pela proposição de critérios de adequação de teste (17%). Sua análise mostra que 60% dos artigos tratam o problema de elaborar entradas de teste e 22% são relacionados a oráculo de teste. Para a geração de entradas de teste, as técnicas mais populares são teste de mutação (31%) e técnicas de search-based (25%). O principal tipo de oráculo usado é o oráculo metamórfico (44%), seguido por falhas específicas do domínio (20%). Os artigos abordam principalmente o teste baseado em modelo (63%), com apenas um artigo

focando no teste de integração. Quando se trata de técnica teste, o teste caixa branca corresponde a 43% e o teste caixa preta a 41%.

No geral, a pesquisa de (Riccio *et al.*, 2020) revelou que novas técnicas de teste precisam ser desenvolvidas para considerar as características peculiares dos sistemas de AM e destaca os problemas mais desafiadores. Dentre os quais destacam-se: a noção de falha para um sistema de AM, a caracterização precisa dos limites de decisão de tais sistemas, a geração de dados realistas dentro da validade dos domínios do sistema AM em teste e a criação de oráculos automatizados e confiáveis para esses sistemas afetados pelo não determinismo e que lidam com um enorme espaço de entrada.

O estudo de (Sherin; khan; Iqbal, 2019) fornece uma visão geral do estado da arte na área e pode ser usado como um catálogo sobre técnicas, critérios e ferramentas de teste para sistemas de AM. Os resultados mostram um aumento significativo no número de publicações nos últimos anos, tendo o maior número de artigos publicados em conferências (76%) em comparação com revistas (6%) e workshops (18%). Várias ferramentas e técnicas são propostas com base em diferentes abordagens. Os autores apresentam uma taxonomia detalhada das técnicas disponíveis, identificam várias lacunas e apresentam sugestões interessantes para investirmos em pesquisas futuras nesta área. A taxonomia destaca temas e abordagens na área, como teste metamórfico, teste baseados em cobertura, teste de mutação, teste simbólico e concólico e computação evolutiva.

O estudo verificou que a maioria das técnicas propostas são caixa preta, o que significa que essas técnicas não precisam de detalhes internos de um objeto de teste. Em contraste, um quarto das técnicas propostas são caixa branca que requerem os detalhes internos do objeto de teste. Há um espaço significativo para se desenvolver pesquisas em teste de programas AM que usam aprendizado semi-supervisionado (classe de algoritmos de AM que são capazes de aprender usando amostras de dados do quais apenas algumas possuem classificação, são usados quando se quer que o algoritmo aprenda informações nos dados mas também aprenda usando alguns dados supervisionados) e por reforço (classe de algoritmo de AM que aprende qual é a melhor ação a ser tomada, dependendo das circunstâncias na qual essa ação será executada). Existem poucas ferramentas disponíveis para os profissionais e é importante que mais ferramentas sejam desenvolvidas e estejam disponíveis para a comunidade de pesquisadores e profissionais em geral.

Uma observação interessante é que a grande maioria dos artigos analisados por esses três trabalhos têm em comum o esforço constante dos pesquisadores em adaptar técnicas e critérios de teste de software tradicional para algoritmos de AM. Nesse sentido, nosso trabalho segue essa mesma abordagem, adaptando conceitos do teste de mutação para AM. Dessa forma, na seção a seguir, serão apresentados trabalhos que também utilizam o teste de mutação para avaliar algoritmos de AM.

## 3.1 Trabalhos Relacionados

Ao investigar trabalhos de teste de software para aprendizado de máquina, observamos que abordagens que utilizam teste de mutação constituíram uma parte significativa desses estudos (BRAIEK; KHOMH, 2020; Zhang *et al.*, 2020). Além disso, encontramos um estudo que reflete criticamente o uso de teste de mutação para AM (PANICHELLA; LIEM, 2021). Discutimos alguns artigos citados neste artigo de reflexão (SHEN; WAN; CHEN, 2018; MA *et al.*, 2018; Hu *et al.*, 2019; JAHANGIROVA; TONELLA, 2020), trabalhos que apresentam uma continuidade e melhoria da abordagem de DeepMutation (RICCIO *et al.*, 2022; HUMBATOVA; JAHANGIROVA; TONELLA, 2021) e alguns outros mais recentes (LU *et al.*, 2022; TAMBON; KHOMH; ANTONIOL, 2023).

MuNN é um método para teste de mutação de redes neurais (SHEN; WAN; CHEN, 2018). Ele fornece cinco operadores de mutação projetados para construir modelos de aprendizado profundo com possíveis bugs. Dado um conjunto de teste, a pontuação de mutação pode ser calculada como a aptidão do teste. Os resultados da mutação podem orientar a geração de testes e a otimização dos testes de redes neurais. Os resultados experimentais mostram algumas características do teste de mutação diferentes do software convencional. Isto é mostrado como um incentivo para projetar alguns operadores de mutação dependentes de domínio em vez de operadores de mutação comuns. Além disso, são apresentadas algumas evidências de que a profundidade do neurônio é um fator sensível no teste de mutação.

A análise de mutação de MuNN não se concentra nas regras de sintaxe, mas sim nas características da rede neural. A intuição é que a mutação pode afetar a função da rede neural, levando a mudanças qualitativas em suas saídas. Por exemplo, consideremos uma rede neural usada para classificação. Essa rede obtém seus resultados finais combinando parâmetros e entradas. Os operadores de mutação incluem: Exclusão de Neurônio de Entrada, Exclusão de Neurônio Oculto, Mudança na Função de Ativação, Alteração no Valor de Polarização e Alteração no Valor do Peso.

Assim como MuNN, o DeepMutation também é uma estrutura para aplicação de técnicas de teste de mutação em sistemas DL (MA *et al.*, 2018). Este artigo propõe uma técnica de teste de mutação em nível de código fonte. Para fazer isso, um conjunto de operadores de mutação no nível de código fonte é projetado para manipular os dados de treinamento e o programa de treinamento. Os operadores de mutação para os dados de treinamento incluem: Repetição de Dados, Erro de Rótulo, Dados Ausentes, Embaralhamento de Dados e Perturbação de Ruído. Os operadores de mutação para código fonte de treinamento incluem: Remoção de Camada, Adição de Camada e Remoção da Função de Ativação. Uma técnica de teste de mutação em nível de modelo também é proposta, definindo um conjunto de operadores de mutação que injetam falhas diretamente em modelos DL, esses operadores incluem: Desfoque Gaussiano, Reorganização de Pesos, Bloqueio de Efeito de Neurônio, Inversão de Ativação de Neurônio, Troca de Neurônio, Desativação de Camada, Adição de Camada e Remoção da Função de

Ativação. Além disso, métricas de teste de mutação são propostas para medir a qualidade dos dados de teste. Uma continuação deste estudo é encontrada na estrutura Deepmutation++ (Hu *et al.*, 2019), uma estrutura de teste de mutação para Redes Neurais Feed-Forward (FNNs) e Redes Neurais Recorrentes (RNNs). DeepMutation++ incorpora oito operadores em nível de modelo para modelos FNN introduzidos em DeepMutation (MA *et al.*, 2018) e propõe nove novos operadores especializados para modelos RNN.

Encontramos outro trabalho que investiga operadores de mutação para DL (JAHANGIROVA; TONELLA, 2020). Identificou-se quais configurações os tornam não equivalentes e não triviais. Neste artigo, os autores propõem uma nova definição de mutantes mortos que considera a natureza estocástica dos sistemas DL. Eles comparam esta definição com a queda na precisão baseada no limite e mostram que isso pode ser inconsistente em diferentes execuções.

DeepCrime (HUMBATOVA; JAHANGIROVA; TONELLA, 2021) também propõe operadores de mutação específicos para sistemas DL, no entanto, esta é a primeira ferramenta que implementa um conjunto de operadores de mutação DL baseados em falhas reais de DL. Neste artigo, os autores definem 35 operadores de mutação DL e implementam 24 operadores de mutação DL no DeepCrime, a primeira ferramenta de mutação de pré-treinamento em nível de fonte baseada em falhas reais de DL. Para avaliar a ferramenta, eles compararam a sensibilidade da ferramenta a mudanças na qualidade dos dados de teste com a do DeepMutation++ (Hu *et al.*, 2019). Os resultados mostram que o DeepCrime produz mutantes matáveis e não triviais, pode discriminar efetivamente um conjunto de teste fraco de um forte e supera significativamente a ferramenta DeepMutation++ nesses aspectos.

Em DeepMetis (RICCIO *et al.*, 2022), os autores descrevem uma maneira automatizada de aumentar os conjuntos de teste existentes com entradas que matam mutantes gerados por DeepCrime (HUMBATOVA; JAHANGIROVA; TONELLA, 2021). O objetivo é aumentar a pontuação de mutação de um conjunto de teste, gerando novas entradas que matam mutantes não eliminados pelo conjunto de teste original. DeepMetis é um gerador de teste baseado em pesquisa para sistemas DL que usa ajuste de mutação como diretriz. Um experimento é realizado para avaliar a abordagem, os resultados mostraram que DeepMetis é eficaz em aumentar o conjunto de teste fornecido, aumentando sua capacidade de detectar mutantes em 63% em média. A avaliação experimental mostra que o aumento do conjunto de teste pode expor mutantes invisíveis, que simulam a ocorrência de falhas não detectadas. DeepMetis difere das abordagens existentes porque seu objetivo é aumentar a capacidade de eliminar mutações em um conjunto de teste. Com o advento de estruturas de mutação DL como DeepMutation (MA *et al.*, 2018), MuNN (SHEN; WAN; CHEN, 2018) e DeepCrime (HUMBATOVA; JAHANGIROVA; TONELLA, 2021), o problema de alcançar uma alta pontuação de mutação é cada vez mais importante, especialmente quando mutantes simulam reais falhas, como é o caso do DeepCrime (HUMBATOVA; JAHANGIROVA; TONELLA, 2021).

Um estudo recente apresenta MTUL, uma nova técnica de teste de mutação para sistemas

de Aprendizagem Não Supervisionada (ANS) (LU *et al.*, 2022). Os autores apresentam como projetaram uma série de operadores de mutação para simular as situações instáveis e possíveis erros que os sistemas ANS podem encontrar e definir os escores de mutação correspondentes. Eles combinam esta técnica com um autoencoder para gerar amostras contraditórias e demonstrar a viabilidade da técnica com base em três *datasets*.

Outro artigo recente focado em DL propõe uma abordagem de Teste Probabilístico de Mutação (PMT) que permite uma decisão mais consistente sobre se um mutante é morto, por meio de avaliação usando três modelos e oito operadores de mutação (TAMBON; KHOMH; ANTONIOL, 2023). Os autores também analisam o trade-off entre o erro de aproximação e o custo do método, mostrando que um erro relativamente pequeno pode ser alcançado com um custo administrável. Os autores argumentam que o PMT é o primeiro passo nessa direção que efetivamente elimina a falta de consistência entre as execuções de testes de métodos anteriores causada pela estocasticidade do treinamento da DNN.

Ao pesquisar o que está sendo discutido sobre teste de mutação e AM também encontramos artigos que discutem o uso de AM aplicado para melhorar alguns aspectos do teste de mutação, mas este não é o foco do nosso trabalho (STRUG; STRUG, 2012; STRUG; STRUG, 2019; ZHANG *et al.*, 2019).

Em contraste com as investigações mencionadas anteriormente, que exploram o uso de teste de mutação na área de aprendizagem profunda, a nossa abordagem centra-se na aplicação de teste de mutação em modelos de classificação. Inspirado nos resultados de nosso trabalho preliminar que investigou a estrutura de árvores de decisão para desenvolver critérios de cobertura, nosso trabalho principal também se baseia na exploração desta estrutura para a geração de mutantes. Diferente dos trabalhos citados nesse capítulo que criam operadores de mutação específicos pra o contexto de redes neurais, nós utilizamos operados já existente e amplamente utilizados na linguagem de programação C, a diferença é que aplicamos esses operadores na estrutura de modelos de árvores de decisão.

## 3.2 Considerações Finais

Este capítulo apresentou as principais contribuições no âmbito do teste de software aplicado ao AM. A pesquisa foi conduzida com base em mapeamentos pré-existentes, o que possibilitou uma investigação mais direcionada em relação aos trabalhos correlatos relacionados ao tema desta dissertação. Esse enfoque permitiu concentrar nossos esforços de pesquisa nos estudos mais relacionados ao escopo desta dissertação.

Em contraste com a investigação mencionada anteriormente, que explorou o uso de teste de mutação na área de aprendizagem profunda, a nossa abordagem centra-se na aplicação de teste de mutação em modelos de classificação. Inspirado em nosso estudo preliminar que investigou a estrutura de árvores de decisão para desenvolver critérios de cobertura (SANTOS *et*

*al.*, 2021), que serão detalhados no Capítulo 4, nosso trabalho também se baseia na exploração desta estrutura para a geração de mutantes. A partir desses mutantes, identificamos as instâncias do *dataset* que podem inativá-los.

Nossa abordagem busca melhorar a robustez e a confiabilidade dos modelos de classificação. Ao aplicar teste de mutação em modelos de classificação, podemos revelar vulnerabilidades ocultas para fortalecer e melhorar o desempenho geral dos algoritmos de AM.

CAPÍTULO

# 4

# TESTE DE SOFTWARE BASEADO EM ÁRVORE DE DECISÃO

Várias abordagens tradicionais de teste de software foram reformuladas para abordar os desafios apresentados pelo Aprendizado Profundo e pelos sistemas baseados em AM. Em um trabalho preliminar, realizado pelo grupo de pesquisa em Engenharia de Software do Instituto de Ciências Matemáticas e de Computação (ICMC) da Universidade de São Paulo (USP), com a participação da autora desta dissertação, foi proposto critérios de teste de cobertura de árvore de decisão (SANTOS *et al.*, 2021). A principal ideia dos critérios de cobertura de árvore de decisão é usar a estrutura interna de tais modelos (em forma de árvore) e as informações nos nós de decisão para guiar a seleção de entradas de teste. Assim, a amostragem de entradas de teste que aumentam a cobertura dos nós de decisão e das folhas resulta em casos de teste mais eficazes.

Esta abordagem se inspira na primeira metodologia de teste estrutural desenvolvida para testar modelos de Aprendizado Profundo, o DeepXplore (PEI *et al.*, 2019). Inspirado pelo conceito de cobertura de teste utilizado no teste de software convencional, PEI *et al.* propuseram o uso da cobertura de neurônios como um meio de impulsionar a geração de testes. A cobertura de neurônios se relaciona com a estimativa da quantidade de neurônios (ou seja, a lógica da rede neural profunda) ativados por um conjunto de entradas. A premissa central é que entradas de teste que alcançam alta cobertura de neurônios podem não apenas revelar problemas, mas também atuar como dados de treinamento. Em conjunto com o dados de treinamento original, esses dados podem aprimorar o modelo em teste por meio de um processo de retreinamento. De acordo com os autores, é possível obter melhorias de até 3% na precisão da classificação ao retreinar o modelo com os dados gerados pelo DeepXplore. Essa abordagem de caixa branca para testar modelos de Aprendizado Profundo lida com o problema da falta de um oráculo de teste, empregando testes diferenciais. Essa abordagem é baseada na observação de discrepâncias na execução de programas que implementam a mesma funcionalidade de maneiras diferentes. Problemas potenciais são identificados quando os programas produzem resultados diferentes ao

serem executados com a mesma entrada.

Nas subseções subsequentes, serão apresentados os critérios fundamentados em árvores de decisão, os quais foram elaborados nos estudos preliminares. Em sequência, serão delineadas as definições da nossa abordagem baseada em mutação. Por fim, serão fornecidas considerações finais, contendo informações sobre a continuidade dos estudos.

## 4.1 Critérios Baseados em Árvores de Decisão

A motivação para utilizar modelos em árvore para seleção de dados de teste é o poder de representação desses modelos. Além disso, os modelos de árvore podem orientar o testador com informações sobre o quanto o modelo de decisão de árvore foi coberto. Este é um avanço em relação ao teste manual ad hoc de software baseado em AM comumente usado para testar esse tipo de software. Portanto, foram propostos dois critérios de teste para cobertura do modelo de árvore de decisão:

- **Critério de Cobertura de Árvore de Decisão - *Decision Tree Coverage (DTC)*:** dado um modelo de árvore de decisão $M$, um conjunto de teste $T$ é considerado adequado para DTC se incluir testes que percorram a árvore de sua raiz a todos os nós folha pelo menos uma vez.

- **Critério de Análise de Valor Limite - *Boundary Value Analysis (BVA)*:** casos de teste são projetados para cobrir valores limites válidos de nós de decisão. Especificamente, ao projetar casos de teste, esse critério requer a seleção de valores que exploram o limite inferior ou superior de cada decisão.

O segundo critério é baseado em uma técnica de teste funcional, análise de valor limite. Adaptou-se tal técnica para ser usada no modelo de árvore de decisão, assim foi possível explorar os valores limites dos nós de decisão da árvore.

Para aplicar os critérios, o primeiro passo foi selecionar as bases de dados e, sempre que necessário, realizar o tratamento dos dados, por exemplo, excluir registros que contenham valores vazios. Em seguida é implementada a árvore de decisão e, para cada folha da árvore, o caminho que leva da raiz à folha é guardado. Cada um desses caminhos representa um requisito de teste. Como cada folha tem um único caminho que a leva à raiz da árvore, o número de requisitos de teste é exatamente o número de nós folhas que a árvore possui.

Com isso tem-se os insumos necessários para a criação dos conjuntos de casos de teste, sendo um conjunto para cada critério. Para atender o critério DTC, selecionou-se na própria base de dados registros que cobrissem cada um dos requisitos de teste. Logo, a quantidade de casos de tese para este conjunto é igual a quantidade de nós folhas da árvore.

O critério BVA prioriza os limites de decisão, isto é, os nós intermediários da árvore. Para atender a esse critério, foi necessário gerar cada caso de teste, pois nem sempre existiam registros na base de dados que estivessem nos limites das decisões. Dessa forma, foram incorporados dados de teste com valores no limite inferior ou superior em relação aos limites de decisão de cada nó interno ao longo de cada caminho da árvore. Considerando que independente de quantos caminhos para prosseguir os nós intermediários possuem, eles sempre finalizam em um nó folha. Assim, tem-se que para cada caminho da árvore (requisito de teste), seleciona-se o valor limite de seus nós intermediários, desse modo a quantidade de casos de tese gerados para cobrir este critério também é igual a quantidade de nós folhas.

Para exemplificar, aplica-se os critérios ao banco de dados de flores canônicas Iris[1], que consiste em 150 amostras/registros, sendo 50 de cada uma das 3 espécies de flores Iris (Iris Setosa, Iris Virginica e Iris Versicolor) e 4 atributos, variáveis que foram medidas, em cada amostra: o comprimento e a largura das sépalas e das pétalas em centímetros. Todos os campos do banco de dados possuem valores numéricos.

A aplicação do algoritmo da árvore de decisão ao *dataset* Iris resultou em um modelo de árvore com nove nós folha, conforme mostra a Figura 6. Como resultado, cada critério gerou um requisito de teste para cada nó folha. Ao usar os critérios para orientar a geração de casos de teste de forma a satisfazer esses nove requisitos de teste, foram criados dois conjuntos de casos de teste, cada um contendo nove casos de teste.

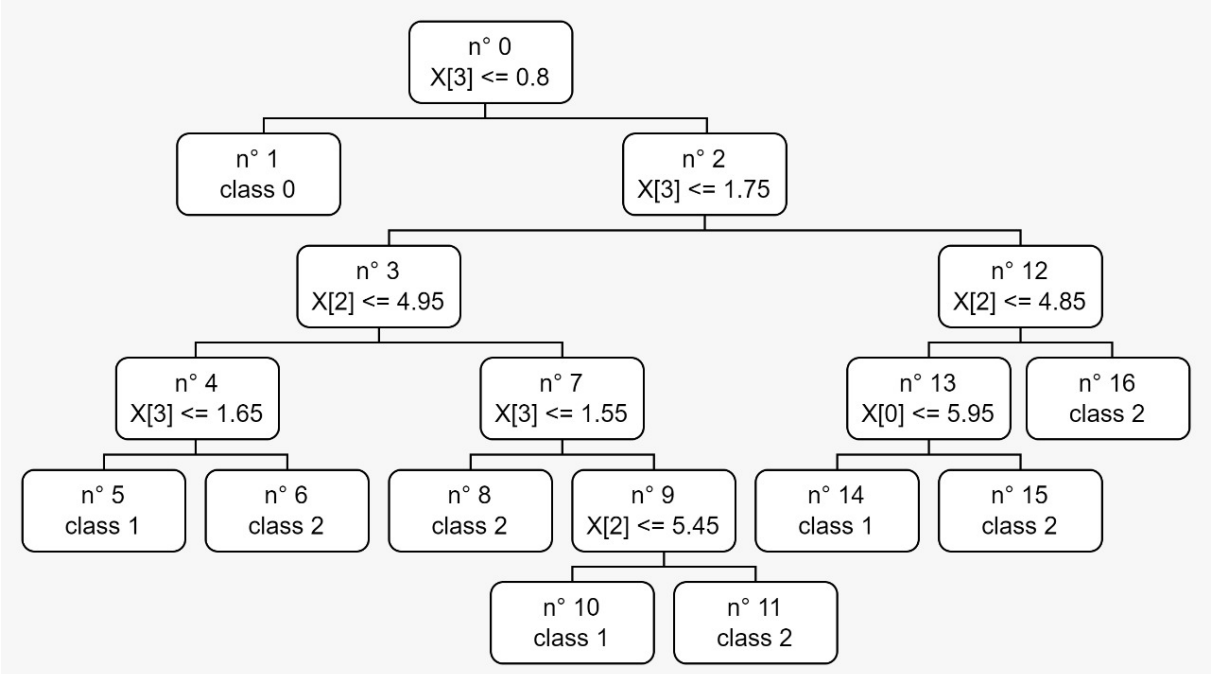



Figura 6 – Modelo de árvore de decisão gerado a partir do *dataset* Iris. As características representadas na figura correspondem a: x[0] = comprimento da sépala, x[1] = largura da sépala, x[2] = comprimento da pétala, x[3] = largura da pétala.

[1] <https://www.kaggle.com/datasets/uciml/iris>

De maneira mais específica, como cada nó folha corresponde a um requisito de teste, para atender ao requisito de teste de cobrir o nó folha 16 ao aplicar o DTC, é necessário selecionar do *dataset* original as entradas que percorram o modelo de árvore do nó 0 até o nó folha 16 (ou seja, seguindo o caminho $[0, 2, 12, 16]$). Um exemplo de caso de teste do DTC que faz o modelo percorrer do nó 0 até o nó folha 16 é mostrado na Tabela 8. Quanto ao BVA, é possível que, ao projetar casos de teste, seja necessário criar valores que não estejam presentes no *dataset* original. Por exemplo, o caso de teste do BVA que cobre o nó 16 (mostrado na Tabela 9) foi projetado ao percorrer a árvore e gerar valores de características que são 10% maiores ou 10% menores que os respectivos valores de características presentes nos nós de decisão.

Tabela 8 – Exemplo de caso de teste DTC que satisfaz o requisito de teste de cobertura do nó folha 16.

| **comprimento da sépala** | **largura da sépala** | **comprimento da pétala** | **largura da pétala** | **espécie (classe)** |
|---|---|---|---|---|
| 6.3 | 3.3 | 6.0 | 2.5 | 2.0 |

Tabela 9 – Exemplo de caso de teste BVA que satisfaz o requisito de teste de cobertura do nó folha 16.

| **comprimento da sépala** | **largura da sépala** | **comprimento da pétala** | **largura da pétala** | **espécie (classe)** |
|---|---|---|---|---|
| 5.000 | 3.00 | 5.335 | 1.925 | 2.0 |

Um estudo foi conduzido para avaliar os critérios propostos e os dois conjuntos de casos de teste gerados. Ao aplicar os conjuntos de teste, elaborados com base nos critérios DTC e BVA, para testar o algoritmo KNN na mesma base de dados, foi alcançada uma acurácia de 67% no modelos para ambos os conjuntos de teste criados. Por outro lado, ao aplicar um critério aleatório para a definição de outro conjunto de teste, como *k-fold cross validation*, no mesmo algoritmo KNN, obteve-se uma acurácia de 96% no modelo. Isso indica que os conjuntos de casos de teste propostos foram mais eficazes na identificação de erros de classificação.

Para avaliar a eficácia desses critério, foi conduzido um estudo experimental. Detalhes da definição e condução do experimento que aplica esses critérios podem ser consultado no Capítulo 5. De acordo com os resultados experimentais, esses critérios de teste de adequação podem ser usados para selecionar dados de teste mais eficazes do que dados de treinamento selecionados aleatoriamente.

Motivado por estes resultados preliminares e reconhecendo a eficácia do teste de mutação na atividade de teste, o próximo passo foi explorar a definição de uma estratégia para aplicar o teste de mutação para melhorar o dados de teste selecionados a partir da árvore de decisão de aplicações de AM. A estratégia proposta é apresentada na seção a seguir.

## 4.2 Teste de Mutação Aplicado em Árvore de Decisão

Com a finalidade de ampliar os critérios de cobertura de folhas de árvores de decisão desenvolvidos em nosso trabalho anterior e continuar contribuindo para a evolução dos testes

para algoritmos de AM, propomos uma nova abordagem que aplica a técnica de análise de mutação na estrutura da árvore de decisão.

Especificamente, nossa abordagem tem como premissa que ao utilizar estrutura interna dos modelos de árvore de decisão para criar árvores mutantes e utilizar essa mutações para selecionar casos de teste que matem esses mutantes, conseguimos criar um conjunto de casos de teste mais eficaz que os casos de teste selecionados aleatoriamente. Esse conjunto de casos de teste serão aplicados em outros algoritmos de AM como, por exemplo, o KNN.

A subseção a seguir apresenta a definição da aplicação de Teste de Mutação em Árvores de Decisão (Decision Tree Mutation Testing (DTMT)).

### 4.2.1 Teste de Mutação em Árvores de Decisão

Buscando ampliar a eficácia dos critérios de cobertura de árvores de decisão desenvolvidos no trabalho preliminar, trazemos os conceitos e os benefícios do teste de mutação para criar uma abordagem que aplica o teste de mutação em modelos de árvores de decisão para selecionar dados de teste.

Para definir o Teste de Mutação de Árvore de Decisão (DTMT), nos inspiramos nos operadores de mutação definidos para a linguagem C (AGRAWAL *et al.*, 1989). Esses operadores são divididos em 4 grupos: 1) operadores de mutação de instrução, 2) operadores de mutação de operador, 3) operadores de mutação variável e 4) operadores de mutação constante. Em particular, os seguintes operadores de mutação são utilizados na nossa definição:

1. Operador ORRN: este operador substitui toda ocorrência de um operador relacional ($<$, $>$, $<=$, $>=$, $==$ e $!=$) por outro possível operador relacional. Exemplo: original: $a < b$; Mutante: $a <= b$.

2. Operador Cccr: este operador substitui ocorrências de uma constante por outra constante possível. Exemplo: Dado um conjunto de contantes: $[7.5, 2.3, 3.14]$; original: $a = 7.5$; Mutante: $a = 3.14$.

Com base nesses operadores de mutação, o DTMT é definido da seguinte forma: dado um modelo de árvore de decisão, um conjunto de teste é considerado adequado para DTMT se incluir casos de teste que gerem um resultado de classificação diferente entre o modelo original e os modelos de árvore mutante. Os modelos de árvore mutante são mutações de um modelo de árvore de decisão na qual os operadores ORRN e Cccr são aplicados nos nós de decisão/intermediários do modelo de árvore original.

No DTMT, cada árvore mutante é um requisito de teste, que, para ser satisfeito, é necessário encontrar linhas do *dataset* que gerem uma classificação diferente entre o mutante e o modelo original.

O número de árvores mutantes depende do número de nós intermediários na árvore original. Assim, ao aplicar o operador ORRN, temos 4 árvores mutantes para cada nó, uma para cada operador relacional possível. Por exemplo, considere a árvore de decisão na Figura 6 que foi gerada a partir do *dataset* Iris [2]. Nesta árvore, 8 nós possuem operadores relacionais; portanto, é possível gerar 32 árvores de decisão mutantes para o operador ORRN. Na Figura 7, temos um exemplo de uma árvore de decisão mutante, onde o nó que teve mutação está destacado.

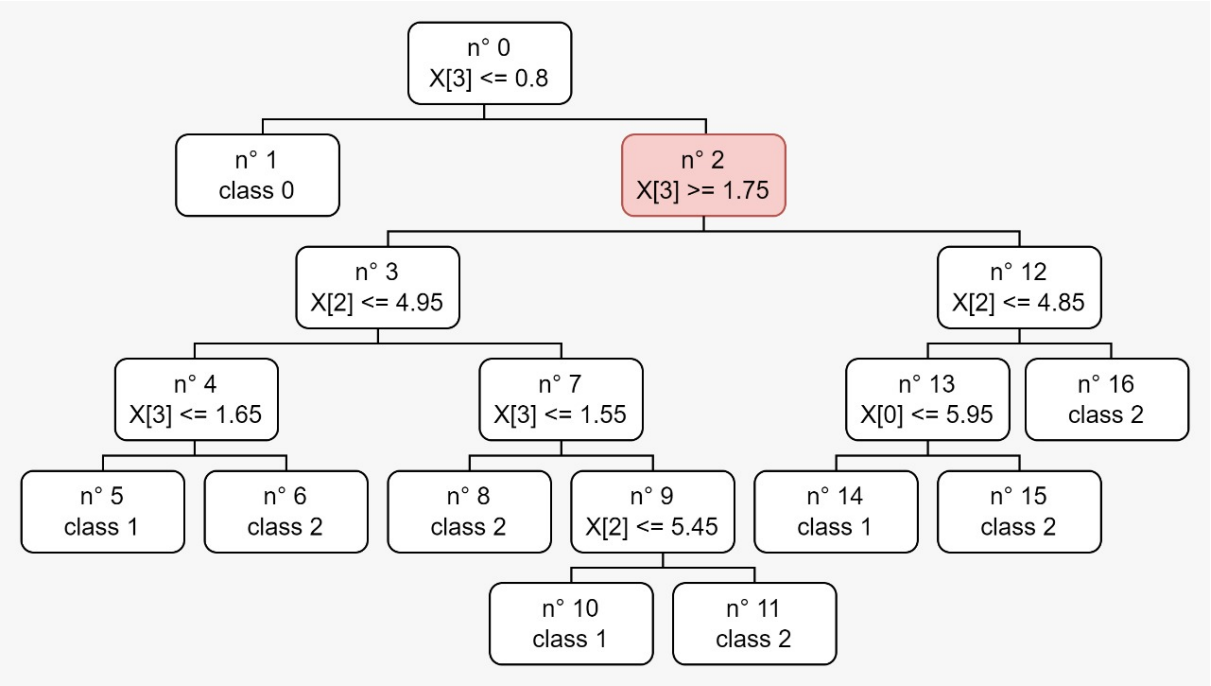



Figura 7 – Exemplo de mutações para o operador ORRN no nó nº 2.

Para aplicar o operador Cccr, cada nó da árvore contendo uma comparação entre uma característica (label) e um valor constante é selecionado, e o valor constante é alterado por outro valor constante usado em outra comparação com a mesma característica. A mutação é aplicada a todos os nós de decisão da nossa abordagem. Observou-se que o número de árvores mutantes geradas pode se tornar impraticável dependendo do número de características presentes na árvore de decisão. Para evitar um número exorbitante de mutantes, foi imposto uma limitação de duas alterações por nó de decisão.

Na Figura 8 temos um exemplo de uma árvore de decisão mutante gerada pelo operador Cccr. Considerando que a árvore de decisão possui 8 nós com comparação, aplicando no máximo duas mutações para cada nó, um total de 10 árvores mutantes são geradas pelo operador Cccr. Isso ocorre porque uma característica pode aparecer apenas uma vez na árvore, como é o caso na característica $X[0]$ em nosso exemplo, ou quando a característica aparecer mais de uma vez na árvore porém com o mesmo valor na constante.

Assim como os critérios de teste DTC e BVA, o DTMT pode ser aplicado para a geração de dados de teste. Do ponto de vista dos requisitos de teste, diferentemente das abordagens de

[2] <https://www.kaggle.com/datasets/uciml/iris>

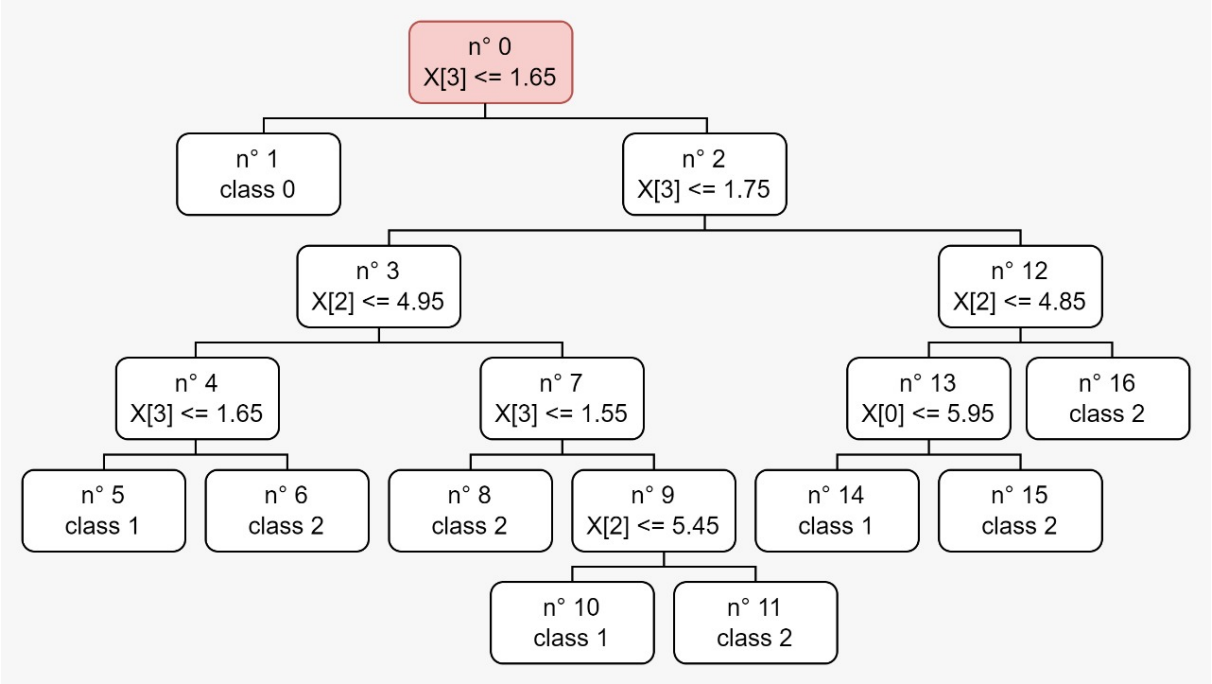



Figura 8 – Exemplo de mutações para o operador de Cccr no nó nº 0.

geração de casos de teste DTC e BVA que visam satisfazer o mesmo número de requisitos de teste: todos os caminhos da raiz à folha na árvore em teste. Nosso DTMT visa matar a maior quantidade possível de mutantes gerados.

Utilizando o DTMT para orientar a geração dos casos de teste, geramos um conjunto de 10 casos de teste. Como cada caso de teste pode matar mais de um mutante, não precisamos necessariamente ter o número de casos de teste igual ao número de requisitos de teste. Com este conjunto de teste, foi alcançada uma taxa de mutação de 80,95%. Logo, dentre as 42 mutações criadas, 8 permaneceram vivas.

Para exemplificar como um caso de teste pode ser considerado capaz de distinguir uma árvore mutante, utilizamos a árvore original apresentada na Figura 6, e a árvore mutante apresentada na Figura 7. Nesta mutação, o nó n°2 teve seu valor alterado de X[3] $<=$ 1,75 para X[3] $>=$ 1,75. Considerando um caso de teste em que X[3] = 1 na árvore original, o caminho seguiria até o nó filho esquerdo (n°3) e terminaria no nó folha n°5, enquanto na árvore mutante seria direcionado para o nó filho direito (n°12) e terminaria no nó folha nº 16. Obtivemos resultados de classificação diferentes e consideramos que o mutante foi morto.

Para avaliar a eficácia da nossa abordagem utilizando mutação, conduzimos um estudo experimental. Detalhes da definição e condução dos experimento que aplica teste de mutação em modelos de árvores de decisão, são apresentados no Capítulo 5.

## 4.3 Considerações Finais

Os critérios e a abordagem proposta baseiam-se na noção de que a estrutura interna dos modelos de árvore de decisão pode ajudar os testadores a selecionar os dados de teste. Como um benefício adicional de explorar a estrutura interna dos modelos, os critérios de teste também permitem que os testadores quantifiquem a eficácia dos dados de teste analisando o número de resultados/decisões abrangidos por esses dados de teste.

Um outro estudo de nível de mestrado está sendo realizado com o propósito de aprimorar os critérios adotados no estudo preliminar (DTC e BVA) e realizar novas avaliações. Entre as mudanças implementadas, destacam-se a inclusão de *datasets* mais robustos para a avaliação experimental, melhorias na metodologia de seleção de dados de teste para o critério BVA (no primeiro, a criação de casos de teste, e no segundo, a seleção daqueles com valores mais próximos), além da reestruturação do experimento com base no feedback recebido de outros pesquisadores da área.

Os critérios de cobertura DTC e BVA, assim como a abordagem que emprega a técnica de mutação, DTMT, demonstraram eficiência na seleção de dados de teste para AM. Embora o foco deste estudo esteja na DTMT no Capítulo 5, serão apresentados detalhes dos dois experimentos realizados, visando facilitar comparações entre DTC, BVA e DTMT.

CAPÍTULO

# 5

# AVALIAÇÃO EXPERIMENTAL

No âmbito desta pesquisa de mestrado, foram realizados três estudos experimentais. O primeiro estudo foi conduzido em colaboração com pesquisadores do grupo e outro trabalho de mestrado e se concentrou na aplicação de critérios de cobertura a modelos de árvores de decisão. O segundo estudo, que constitui o principal foco desta pesquisa de mestrado, investigou a aplicação do teste de mutação em árvores de decisão. O terceiro estudo expande a quantidade de *dataset* utilizada e compara o critério de cobertura de árvore de decisão mais eficaz com DTMT.

Este capítulo apresenta em detalhes a metodologia e os procedimentos adotados na condução dos experimentos, os quais serviram para comprovar a eficácia de ambas as abordagens. Além disso, serão discutidos os resultados obtidos em cada um dos estudos.

Vale ressaltar que, com o intuito de facilitar comparações significativas, foram adotados procedimentos similares nos experimentos, embora tenham sido levadas em consideração as particularidades inerentes a cada abordagem.

## 5.1 Configuração do Experimento sobre Critérios de Cobertura de Árvores de Decisão

Neste primeiro experimento foi feita a comparação entre os dois critérios BVA e DTC com *k-fold cross validation*. Aqui, supõe-se que os critérios propostos são mais adequados para gerar suítes de teste para programas baseados em AM. A hipótese é baseada no fato de que os critérios suportam a estrutura interna e as informações de decisão dos modelos de árvore de decisão durante a identificação dos requisitos de teste. A suposição foi que os nós de decisão desempenham um papel fundamental no comportamento do modelo e que é possível projetar casos de teste mais eficazes, beneficiando-se da estrutura e das informações de decisão subjacentes do modelo. Logo, foi proposto um experimento para investigar a seguinte Questão

de Pesquisa (QP):

***QP1**: Quão eficazes são os casos de teste derivados dos critérios DTC e BVA em comparação com casos de teste aleatórios k-fold cross validation?*

### 5.1.1 Escopo

Objetivos do experimento: O escopo do experimento é estabelecido pela definição de seus objetivos, que foram resumidos de acordo com o modelo de Objetivo / Pergunta / Métrica (GQM) (WOHLIN *et al.*, 2012):

**Analisar** os dois critérios de cobertura da árvore de decisão
**Para fins** de comparação
**Com relação à** eficácia
**A partir do ponto vista** do pesquisador
**No contexto** de testes de programas baseados em AM.

A eficácia em modelos de AM é uma medida geral da habilidade do modelo em realizar suas tarefas de forma precisa e confiável. Geralmente, refere-se à capacidade do modelo de atingir seus objetivos, que podem variar de acordo com o contexto e a aplicação específica. Essa eficácia é avaliada através de métricas que indicam quão bem o modelo está performando em relação aos dados de teste ou validação (MULLER; GUIDO, 2018). Essas métricas podem ser a Precision, Recall, Accuracy, $F_1$, entre outras, dependendo do tipo de problema de aprendizagem (classificação, regressão, etc.). É importante ressaltar que a avaliação da eficácia de um modelo muitas vezes não se limita a uma única métrica, pois diferentes métricas podem ser relevantes para diferentes aspectos do problema.

Na subseção a seguir, serão apresentadas as hipóteses usadas para investigar a QP e fornecer a definição operacional de eficácia (VANDERSTOEP; JOHNSON, 2008).

### 5.1.2 Formulação de hipóteses

QP1 foi formalizada em hipóteses para que testes estatísticos possam ser realizados. A previsão para QP1 foi estruturada como: o critério proposto é mais eficaz do que o teste aleatório. Assim, QP1 foi transformado nas seguintes hipóteses:

- **Hipótese nula, H0**: não há diferença na eficácia entre os critérios propostos e os testes aleatórios.
- **Hipótese alternativa, H1**: há uma diferença significativa na eficácia entre os critérios propostos e os testes aleatórios.

O objetivo principal foi investigar se os critérios propostos resultam em casos de teste mais eficazes. Para avaliar essa suposição, foram derivadas suítes de teste que, quando aplicadas a modelos de árvore de decisão, satisfazem DTC e BVA. Em seguida, utiliza-se os conjuntos de teste adequados para DTC e BVA para avaliar o desempenho dos modelos gerados a partir de outros algoritmos de AM (ou seja, k-NN). Especificamente, no início, houve o treinamento dos modelos aplicando k-NN ao *dataset* original (ou seja, dados de treinamento): esses modelos são treinados e, em seguida, testados usando *k-fold cross validation*. Posteriormente, foi feita a comparação entre os resultados de *k-fold cross validation* com os resultados da aplicação de teste DTC e BVA para o modelo k-NN mencionado anteriormente.

Considerando o teste de software tradicional, um critério $C_1$ é mais eficaz do que outro critério $C_2$ se $C_1$ leva à geração de entradas de teste que são capaz de revelar mais falhas do que as entradas derivadas da aplicação de $C_2$. No contexto deste estudo, se um determinado modelo $M$ apresenta desempenho pior para as entradas de teste derivadas de um critério $C_1$ do que para as entradas de teste geradas pela aplicação de $C_2$, então é dito que $C_1$ é mais eficaz do que $C_2$. Dada uma métrica $f$, que mede a qualidade de um determinado modelo $M$, e um conjunto de teste $T$, usa-se $f(M(T))$ para denotar a pontuação da execução. Consequentemente:

$$f(M(T_1)) < f(M(T_2))$$

indica que $T_1$ é mais eficaz do que $T_2$ de acordo com $f$ porque $T_1$ revelou mais casos em que $M$ falhou do que $T_2$.

Existem diversas maneiras de avaliar e comparar modelos de AM. Acredita-se ser apropriado levar em consideração várias métricas de desempenho de AM amplamente utilizadas nesta comparação. Assim, foram escolhidas as métricas *precision, recall, accuracy e* $F_1$ para avaliar os resultados. A variável dependente na qual há mais interesse em responder QP1 é a eficácia, que é medida em termos das métricas de desempenho de AM acima mencionadas no contexto deste experimento.

Este experimento tem um fator e dois tratamentos: o fator é a abordagem de teste e os tratamentos são os critérios de adequação e testes aleatórios (ou seja, *k-fold cross validation*).

### 5.1.3 Instrumentação

Antes de conduzir o experimento, foram definidos os objetos experimentais. Esses objetos são agrupados principalmente em duas categorias: scripts python para carregamento e processamento (incluindo geração de casos de teste) *datasets* e scripts para analisar os resultados. Foram usados o Google Colaboratory (ou Colab para abreviar)[1], que é um ambiente baseado em nuvem para executar scripts python no navegador. Especificamente, o Google Colab permite que

[1] <https://colab.research.google.com/>

os usuários criem notebooks[2] que combinam código Python executável e texto em linguagem natural em um único documento. Usa-se o pandas[3] para a preparação e análise de dados. Todos os algoritmos de AM destes objetos de experimento foram implementados usando diretamente o scikit-learn[4], que é a biblioteca Python mais proeminente para AM (MÜLLER; GUIDO, 2016).

Como mencionado anteriormente, para avaliar a eficácia dos casos de teste gerados conforme os critérios de árvore de decisão, prosseguiu-se com a utilização da suíte de teste resultante para avaliar os modelos gerados a partir da aplicação do algoritmo k-NN. Optou-se pela utilização do k-NN nessa avaliação por sua simplicidade, dado que o modelos classifica novos dados com base na classe predominante entre seus k vizinhos mais próximos, usando a distância entre os pontos como critério de classificação (MÜLLER; GUIDO, 2016). Os modelos resultantes foram comparados em termos de quatro métricas após 10 execuções cruzadas: Precision, Recall, Accuracy e $F_1$.

### 5.1.4 Execução

Para investigar os critérios propostos, foi selecionada uma amostra de 12 *datasets* de dados populares. Optou-se por selecionar *datasets* que têm sido amplamente usados para treinar modelos de AM porque são menos complexos e geralmente mais limpos do que os *datasets* do mundo real, o que resulta em árvores de decisão que são menores e mais fáceis de interpretar. Uma visão geral dos *datasets* usados nesse experimento é mostrada na Tabela 10.

Tabela 10 – Visão geral dos *datasets* usados no experimento.

| *Datasets* | Amostras | Atributos | Classes | Amostras por Classe |
|---|---|---|---|---|
| Phoneme | 5,404 | 6 | 2 | [3,818, 1,586] |
| Mammography | 11,183 | 7 | 2 | [10,923, 260] |
| Pima Indians Diabetes | 768 | 9 | 2 | [500, 268] |
| Haberman's Survival | 306 | 4 | 2 | [225, 81] |
| Cleveland Heart Disease | 297 | 15 | 2 | [160, 137] |
| Oil Spill | 937 | 50 | 2 | [896, 41] |
| Banknote Authentication | 1,372 | 5 | 2 | [762, 610] |
| Ionosphere | 351 | 35 | 2 | [225, 126] |
| Sonar, Mines vs. Rocks | 208 | 61 | 2 | [97, 111] |
| Breast Cancer Wisconsin | 569 | 31 | 2 | [212, 357] |
| Wine Recognition | 178 | 14 | 3 | [59, 71, 48] |
| Iris | 150 | 5 | 3 | [50, 50, 50] |

Antes da geração do caso de teste, foram carregados os dados de todos os *datasets* no ambiente Google Colab (ou seja, Python). Para isso, usou-se os métodos da biblioteca pandas para carregar dados externos. Considerando que alguns *datasets* da amostra de experimentos tinham alguns valores de N/A, realizou-se algumas etapas de limpeza (também empregando a

[2] Notebooks do Google Colab são espaços interativos para criar e executar códigos Python, permitindo inserir outras mídias, organizados em células.

[3] <https://pandas.pydata.org>

[4] <https://scikit-learn.org/stable/>

biblioteca pandas) com o objetivo de criar *datasets* mais consistentes. Depois de organizar os dados, os *datasets* foram separados em duas partes: dados de treinamento (ou seja, X) e suas saídas ou rótulos correspondentes (ou seja, Y).

A próxima etapa foi construir modelos de árvore de decisão para os dados de treinamento: conforme mencionado, para ajustar os modelos aos dados de treinamento que se fez uso do scikit-learning. Árvores de decisão scikit-learning são implementadas de forma que os modelos resultantes também contenham informações auxiliares que serão usadas em etapas posteriores da execução do experimento. Especificamente, esses modelos também mantêm informações sobre o número de nós em uma determinada árvore, filhos esquerdo e direito de cada nó, o limite de todos os nós de decisão e uma lista de todos os nós folha.

Após a geração do modelo, foi desenvolvido um método recursivo com o objetivo de encontrar caminhos exclusivos da raiz para cada nó de folha. Uma vez que existe um caminho único da raiz para cada folha, há uma representação única dos caminhos da raiz para cada folha. Portanto, começa-se da raiz e aplicando a travessia de pós-ordem (visitando todos os nós da subárvore direita seguida por todos os nós da subárvore esquerda), essa implementação recursiva mantém o controle de cada caminho único da raiz para a folha.

Na seleção de dados de teste para satisfazer o DTC, é realizado uma busca aleatória no *dataset* original para encontrar entradas de teste que cobrem cada caminho único da raiz à folha. Como resultado, o algoritmo produz dados de teste que cobrem cada folha do modelo de árvore em teste. Com a geração de dados mais forte ao aplicar BVA, ao percorrer cada caminho raiz para folha, ao atingir os nós de decisão, o algoritmo cria entradas de teste que são ligeiramente maiores ou menores (ou seja, valores de limite) do que os valores presentes nos nós de decisão. Consequentemente, as entradas de teste para os recursos que aparecem ao longo dos caminhos raiz para folha podem incluir valores que provavelmente não estarão presentes no *dataset* original.

Os *datasets* originais foram usados para treinar e testar modelos k-NN: o treinamento e o teste foram realizados usando *k-fold cross validation*. Finalmente, os modelos k-NN resultantes foram testados com suítes de teste adequadas para DTC e BVA, e os resultados dos testes foram comparados com os resultados de *k-fold cross validation* executados anteriormente.

## 5.2 Resultados do Experimento sobre Critérios de Cobertura de Árvores de Decisão

Nesta seção serão descritos os resultados experimentais para os 12 *datasets* que foram analisados. Primeiro, é discutido algumas estatísticas descritivas e, em seguida, é apresentado o teste de hipótese. Para permitir a replicação do estudo, foi disponibilizado os dados, código-fonte,

resultados e scripts de análise de dados online[5].

### 5.2.1 Tabela de Características

A Tabela 11 oferece uma visão geral das propriedades básicas dos modelos de árvore de decisão gerados a partir dos *datasets*. Conforme mostrado na Tabela 11, o modelo Phonemed foi o que apresentou a maior quantidades de nós: 1.041 nós, dos quais 521 são nós de folhas. O Iris é o *dataset* que levou à criação do modelo mais simples, que é composto de 17 nós, dos quais nove são folhas.

Os outliers dos dados relativos à altura dos modelos de árvore resultante (ou seja, quantidade de nós), considera os valores medianos na Tabela 11 como uma medida mais precisa da tendência central do que a média. Assim, em média (mediana), os modelos no experimento têm 61 nós, dos quais 31 são folhas. Além disso, também devido a outliers, o desvio absoluto mediano (MAD) é uma medida mais robusta de dispersão estatística do que o desvio padrão (ver Tabela 11).

Tabela 11 – Número de nós nos modelos de árvore de decisão resultantes e número de casos de teste gerados a partir da aplicação dos critérios DTC e BVA a esses modelos.

| ***Datasets*** | **Nós** | **DTC/BVA** |
|---|---|---|
| Phoneme | 1,041 | 521 |
| Mammography | 319 | 160 |
| Pima Indians Diabetes | 255 | 128 |
| Haberman's Survival | 207 | 104 |
| Cleveland Heart Disease | 99 | 50 |
| Oil Spill | 69 | 35 |
| Banknote Authentication | 53 | 27 |
| Ionosphere | 45 | 23 |
| Sonar, Mines vs. Rocks | 45 | 23 |
| Breast Cancer Wisconsin | 43 | 22 |
| Wine Recognition | 23 | 12 |
| Iris | 17 | 9 |
| | **Estatísticas descritivas** | |
| **Max** | 1,041 | 521 |
| **Min** | 17 | 9 |
| **Média** | 184.67 | 92.83 |
| **Mediana** | 61 | 31 |
| **Std Dev** | 287.34 | 143.67 |
| **MAD**‡ | 180.55 | 90.28 |

‡MAD significa desvio absoluto mediano.

Como cada caminho raiz para folha representa um requisito de teste, a quantidade de casos de teste gerados pelos critérios propostos é proporcional ao número de nós folha em um determinado modelo. Portanto, a aplicação dos critérios aos modelos resultantes gerou, em média (mediana), aproximadamente 31 requisitos de teste (Tabela 11) .

[5] <https://doi.org/10.5281/zenodo.4772950>

### 5.2.2 Teste de hipóteses

A fim de investigar se os critérios propostos levam à criação de dados de teste que são mais eficazes na avaliação do desempenho dos modelos de AM, foram usados t-testes de duas amostras não pareados bi-caudais para avaliar as diferenças no valor médio da métrica usada no experimento (ou seja, Precision, Recall, Accuracy e $F_1$). Foi verificado a normalidade usando o teste de Shapiro-Wilk antes dos testes de execução. De acordo com os resultados do Shapiro-Wilktest, todas as distribuições dos resultados da aplicação dos dados do teste DTC e BVA a um modelo k-NN, bem como os resultados de *k-fold cross validation*, são normais (conforme mostrado na Tabela 12).

Tabela 12 – Resultados da aplicação de dados de teste DTC e BVA a modelos k-NN gerados a partir de cada *dataset*.

| ***Dataset*** | **Casos de Teste DTC** | | | | **Casos de Teste BVA** | | | | ***k-fold cross validation*** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **Accuracy** | **$F_1$** | **Precision** | **Recall** | **Accuracy** | **$F_1$** | **Precision** | **Recall** | **Accuracy** | **$F_1$** |
| Phoneme | 0.61 | 0.61 | 0.61 | 0.60 | 0.52 | 0.52 | 0.52 | 0.52 | 0.89 | 0.89 | 0.89 | 0.89 |
| Mammography | 0.63 | 0.62 | 0.63 | 0.57 | 0.58 | 0.59 | 0.59 | 0.55 | 0.99 | 0.99 | 0.99 | 0.99 |
| Pima Indians Diabetes | 0.45 | 0.46 | 0.46 | 0.45 | 0.52 | 0.53 | 0.53 | 0.50 | 0.70 | 0.69 | 0.69 | 0.68 |
| Haberman's Survival | 0.56 | 0.56 | 0.56 | 0.48 | 0.63 | 0.64 | 0.64 | 0.62 | 0.69 | 0.69 | 0.69 | 0.68 |
| Cleveland Heart Disease | 0.45 | 0.44 | 0.44 | 0.44 | 0.59 | 0.56 | 0.56 | 0.55 | 0.59 | 0.57 | 0.57 | 0.57 |
| Oil Spill | 0.21 | 0.40 | 0.40 | 0.28 | 0.24 | 0.49 | 0.49 | 0.32 | 0.93 | 0.95 | 0.95 | 0.94 |
| Banknote Authentication | 1.00 | 1.00 | 1.00 | 1.00 | 0.47 | 0.48 | 0.48 | 0.47 | 0.99 | 0.99 | 0.99 | 0.99 |
| Ionosphere | 0.78 | 0.48 | 0.48 | 0.39 | 0.15 | 0.39 | 0.39 | 0.22 | 0.86 | 0.84 | 0.84 | 0.83 |
| Sonar, Mines vs. Rocks | 0.63 | 0.64 | 0.64 | 0.61 | 0.47 | 0.50 | 0.50 | 0.48 | 0.83 | 0.81 | 0.81 | 0.81 |
| Breast Cancer Wisconsin | 0.65 | 0.64 | 0.64 | 0.64 | 0.64 | 0.64 | 0.64 | 0.64 | 0.93 | 0.93 | 0.93 | 0.93 |
| Wine Recognition | 0.64 | 0.33 | 0.33 | 0.32 | 0.29 | 0.42 | 0.42 | 0.34 | 0.74 | 0.70 | 0.70 | 0.69 |
| Iris | 0.69 | 0.67 | 0.67 | 0.67 | 0.69 | 0.67 | 0.67 | 0.67 | 0.97 | 0.96 | 0.96 | 0.96 |
| | **Estatísticas descritivas e teste de normalidade** | | | | | | | | | | | |
| **Max** | 1.00 | 1.00 | 1.00 | 1.00 | 0.69 | 0.67 | 0.67 | 0.67 | 0.99 | 0.99 | 0.99 | 0.99 |
| **Min** | 0.21 | 0.33 | 0.33 | 0.28 | 0.15 | 0.39 | 0.39 | 0.22 | 0.59 | 0.57 | 0.57 | 0.57 |
| **Média** | 0.61 | 0.57 | 0.57 | 0.54 | 0.48 | 0.53 | 0.53 | 0.49 | 0.84 | 0.84 | 0.84 | 0.83 |
| **Mediana** | 0.53 | 0.59 | 0.63 | 0.59 | 0.51 | 0.53 | 0.52 | 0.53 | 0.86 | 0.87 | 0.88 | 0.87 |
| **Std Dev** | 0.19 | 0.17 | 0.17 | 0.19 | 0.17 | 0.09 | 0.09 | 0.14 | 0.13 | 0.14 | 0.14 | 0.14 |
| **Shapiro-Wilk (W)** | W=0.94, p=0.49 | W=0.89, p=0.13 | W=0.89, p=0.13 | W=0.92, p=0.29 | W=0.90, p=0.18 | W=0.96, p=0.81 | W=0.96, p=0.81 | W=0.94, p=0.47 | W=0.92, p=0.25 | W=0.91, p=0.20 | W=0.91, p=0.20 | W=0.91, p=0.22 |

Para testar a hipótese de que há uma diferença significativa na eficácia entre os critérios propostos e o teste aleatório, utilizou-se t-teste de duas amostras não pareados bi-caudais. Este parametrismo é uma abordagem bem conhecida para comparar amostras (ou seja, *datasets*) medidos nas mesmas variáveis dependentes (ou seja, as métricas de desempenho escolhidas), mas sob diferentes níveis de uma variável independente (ou seja, DTC, BVA e *k-fold cross validation*): De acordo com os resultados dos testes, os critérios são capazes de gerar dados de teste que diferem significativamente dos dados de teste de *k-fold cross validation*. Da mesma forma, os resultados também parecem sugerir que os dados do teste de BVA também diferem significativamente dos dados originais em todas as métricas. Os resultados são representados por pontuações t e valores p na Tabela 13.

### 5.2.3 Discussão dos Resultados do Experimento sobre Critérios de Cobertura de Árvores de Decisão

Para responder ao $\mathbf{RQ}_1$ nos concentramos em verificar se nossos critérios podem ser usados para orientar a criação sistemática de entradas diversificadas que vão desde aquelas semelhantes aos dados de treinamento até entradas nos limites dos nós de decisão. Dado que o objetivo dos nossos critérios é gerar dados de teste que difiram sistematicamente dos dados de treinamento, conjecturamos que dados de teste adequados para DTC e BVA resultam em diminuição no desempenho. Postulamos que é possível criar sistematicamente diversos dados de teste (quando comparados aos dados de treinamento), aproveitando a estrutura interna dos modelos de árvore de decisão. Além disso, dado que os modelos de AM são provavelmente mais propensos a erros (ou seja, não conseguem prever corretamente) quando expostos a entradas desconhecidas, ou seja, diversas.

Nesse contexto, medimos o quão *eficaz* é a entrada de teste para um modelo de AM em relação aos dados com os quais o modelo foi treinado: aqui a eficácia foi medida em termos de quão prejudiciais os dados do teste afetaram o desempenho preditivo do modelo em avaliação: quanto mais eficazes os dados do teste, pior o desempenho preditivo do modelo. Portanto, o teste aleatório aqui tem a ver com a avaliação do modelo usando *k-fold cross validation* (ou seja, aprendendo com os dados de treinamento disponíveis e depois testando e avaliando o modelo usando os dados de teste).

De acordo com os resultados mostrados nas tabelas 12 e 13, ao usar nossos dados de teste para testar modelos k-NN avaliados por meio de *k-fold cross validation*, esses modelos mostram diminuições na predição desempenho em todas as métricas. Nossa resposta a $\mathbf{RQ}_1$ está centrada nas métricas mencionadas no Capítulo 2), que foram usadas como proxies para a eficácia dos dados de teste resultantes. Considerando os valores de $F_1$, os resultados parecem sugerir que os dados de teste gerados usando nossos critérios levam a entradas mais desafiadoras, o que por sua vez resulta em uma diminuição significativa de desempenho em relação à linha de base ($F_1$ de *k-fold cross validation*).

Tabela 13 – Resumo dos resultados dos testes estatísticos entre DTC, BVA e *k-fold cross validation*.

| | **Teste de Amostras Emparelhadas ($t$)** | |
|---|---|---|
| **Métrica** | **DTC x *k-fold cross validation*** | **BVA$^{\ddagger}$ x *k-fold cross validation*** |
| **Precision** | $t$ = -3.47, valor-p $\leq$ 0.01 | $t$ = -5.76, valor-p $\leq$ 0.001 |
| **Recall** | $t$ = -4.11, valor-p $\leq$ 0.001 | $t$ = -6.25, valor-p $\leq$ 0.001 |
| **Accuracy** | $t$ = -4.11, valor-p $\leq$ 0.001 | $t$ = -6.25, valor-p $\leq$ 0.001 |
| **$F_1$** | $t$ = -4.23, valor-p $\leq$ 0.001 | $t$ = -5.98, valor-p $\leq$ 0.001 |

Por exemplo, para o modelo k-NN gerado a partir do *dataset Oil Spill*, enquanto no *k-fold cross validation* o modelo rendeu um valor $F_1$ de 0,94, ele teve um desempenho significativamente pior no teste adequado de DTC dados: 0,28. Da mesma forma, os dados do teste BVA resultaram em diminuição no desempenho preditivo para o mesmo modelo: 0,32. Para o modelo k-NN gerado a partir do *dataset Ionosphere*, o valor $F_1$ para *k-fold cross validation* foi 0,83 e houve uma diminuição significativamente mais pronunciada no desempenho quando o modelo foi testado com dados de teste BVA (0,22 ) e uma degradação considerável do desempenho para dados adequados para DTC (0,39). Em termos de $F_1$ há uma diferença estatisticamente significativa entre DTC (média = 0,54) e *k-fold cross validation* (média = 0,83): $t$ = -4,23, valor p $\leq$ 0,001 (Tabela 13). Há também uma diferença estatisticamente significativa entre o BVA (média = 0,49) e *k-fold cross validation*: $t$ = -5,98, valor p $\leq$ 0,001 (Tabela 13).

Portanto, conforme mostrado na Tabela 13 e na Figura 9, há uma diferença estatisticamente significativa entre DTC e *k-fold cross validation*, bem como entre BVA e *k-fold cross validation* para todas as métricas consideradas. Argumentamos que isso indica que o DTC e o BVA podem ser usados para orientar a geração de novos dados de teste a partir dos existentes (que são renderizados na estrutura interna dos modelos de árvore de decisão) com o objetivo de detectar comportamentos errôneos (ou seja, não conseguir prever diversos entradas de teste) em modelos de AM.

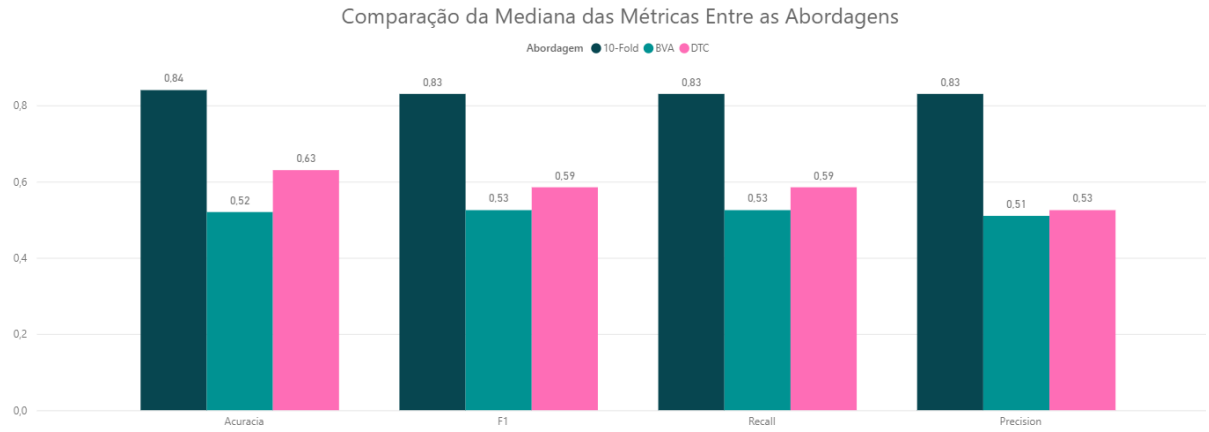



Figura 9 – Comparação dos resultados obtidos para as métricas avaliadas com DTC, BVA e *k-fold cross validation*

## 5.3 Configuração do Experimento sobre Teste de Mutação aplicado a árvores de Decisão

Para avaliar a eficácia de nossa abordagem de teste de mutação de árvore de decisão, pretendemos compará-la com a abordagem *k-fold cross validation*. Nossa hipótese é que a aplicação de mutação a modelos de árvore de decisão tende a resultar em conjuntos de teste

de maior qualidade, isto é, conjuntos de teste capazes de encontrar mais problemas (falhas) em aplicações de AM, ou seja, exemplos em que os modelos presentem classificações incorretas.

A hipótese se baseia no fato de que DTMT aproveita a estrutura interna e as informações de decisão dos modelos de árvore de decisão durante a identificação dos requisitos de teste. Foi conjecturado que a análise de mutação é um método eficaz para direcionar a seleção de casos de teste a partir do *dataset* original, utilizando versões defeituosas da árvore de decisão (mutantes) para verificar discrepâncias na classificação entre a árvore original e suas variantes mutantes.

Supõem-se que realizando mutações os nós não terminais (ou seja, nós de decisão) que desempenham um papel fundamental no comportamento do modelo, é possível selecionar casos de teste mais capazes em identificar falhas em um modelos e AM. Um experimento foi projetado para responder à seguinte Questão de Pesquisa (QP):

***RQ*$_1$*: Como a eficácia dos conjuntos de teste derivados do DTMT se compara aos casos de teste aleatórios de k-fold cross validation?***

### 5.3.1 Escopo

O escopo de nosso experimento é definido de acordo com a configuração dos objetivos, feita baseada no modelo Objetivo/Pergunta/Métrica (Goal/Question/Metric (GQM) (WOHLIN *et al.*, 2012)) da seguinte forma:

**Analisar** a abordagem DTMT
**com o objetivo de** avaliação
**no que diz respeito à sua** eficácia
**do ponto de vista** do pesquisador
**no contexto de** testar programas baseados em AM.

### 5.3.2 Formulação de Hipóteses

Para permitir a aplicação de testes estatísticos, refinamos a QR em hipóteses específicas. Definimos nossa previsão para **RQ** como: nosso critério baseado em mutação é mais eficaz que testes aleatórios. Assim, nosso QR foi transformado nas seguintes hipóteses:

**Hipótese nula, H$_0$:** não há diferença na eficácia entre DTMT e testes aleatórios.

**Hipótese alternativa, H$_1$:** há uma diferença na eficácia entre o DTMT e o teste aleatório.

Nosso principal objetivo é investigar se o DTMT leva à seleção de dados de teste mais eficazes. Para avaliar esta suposição, pretendemos derivar um conjunto de teste que satisfaça o DTMT quando aplicado a um modelo de árvore de decisão. Posteriormente, usamos o conjunto

de teste adequado ao DTMT resultante para avaliar o desempenho de modelos gerados a partir de outros algoritmos de AM, como kNN.

A Figura 10 oferece uma visão geral do processo, o qual está dividido em duas fases distintas: I) A primeira fase consiste na criação das árvores de decisão, seguida pela geração das árvores mutantes, e posteriormente, na seleção dos registros do *dataset* que matam os mutantes. Esses registros compõem o conjunto de casos de teste da abordagem DTMT. II) A segunda fase compreende o treinamento e teste do modelo KNN utilizando a abordagem DTMT e *k-fold cross validation*. Em seguida, é realizada um comparação entre os resultados de *k-fold cross validation* com os resultados da aplicação dos conjuntos de teste DTMT, neste momento métricas são coletadas para fins de comparação estatística. Detalhes mais abrangentes sobre este processo são fornecidos de maneira minuciosa na subseção 5.3.4.

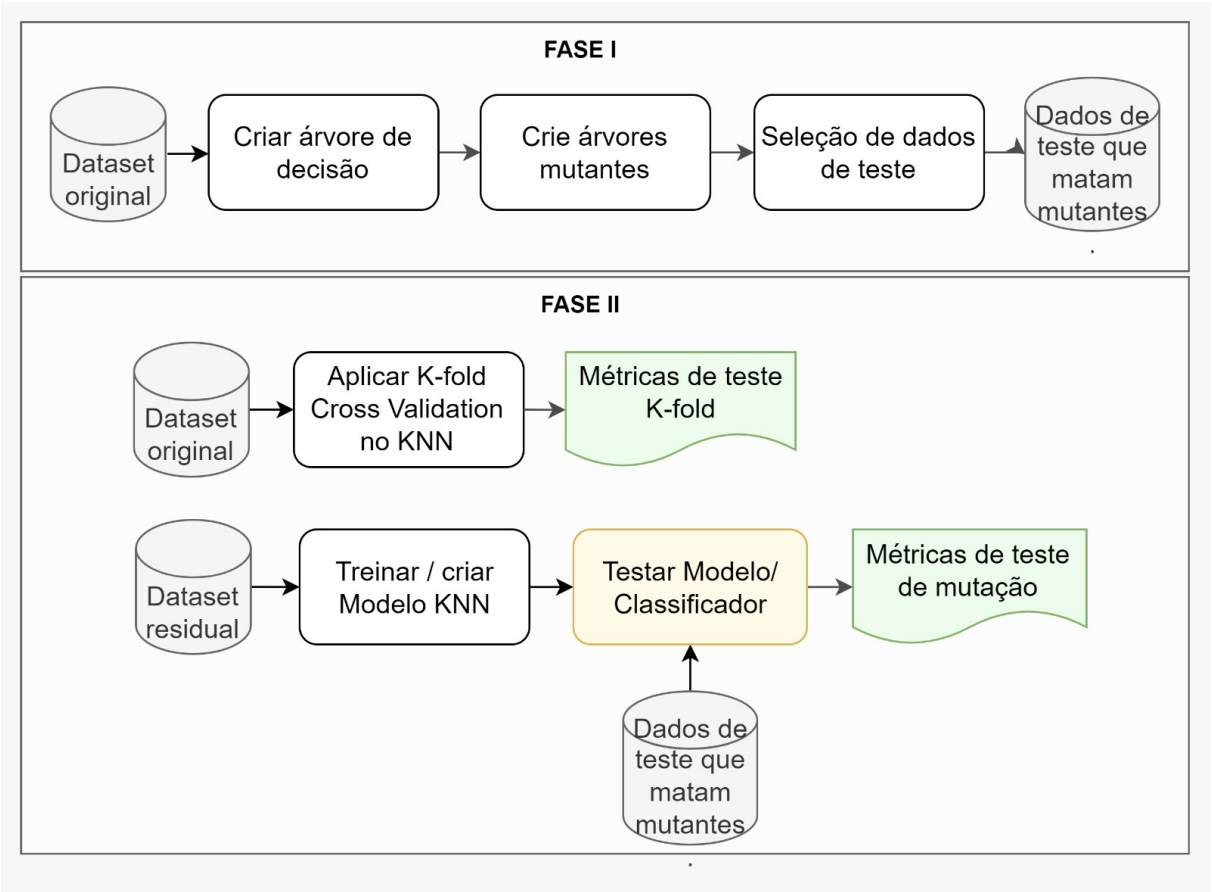



Figura 10 – Ilustração do fluxo do experimento

No contexto de teste de software tradicional, a eficácia de um critério é determinada pela sua capacidade de gerar entradas de teste que podem revelar mais falhas. Portanto, um critério $C_1$ é considerado mais eficaz do que outro critério $C_2$ se o primeiro levar a entradas de teste que possam descobrir mais falhas do que aquelas derivadas da aplicação do último. Este conceito também se aplica ao nosso estudo. Para um determinado modelo $M$, se a capacidade de classificar corretamente novas instâncias for pior com entradas de teste derivadas de $C_1$ do que aquelas geradas a partir de $C_2$, então dizemos que $C_1$ é mais eficaz que $C_2$. Dada uma métrica,

denotada como $f$, para avaliar a qualidade do modelo $M$ e um conjunto de teste, denotado como $T$. A pontuação da execução do conjunto de teste $T$ em relação ao modelo $M$, de acordo com a métrica $f$, é expressa como $f(M(T))$. Portanto:

$$f(M(T_1)) < f(M(T_2))$$

indica que $T_1$ é mais eficaz que $T_2$ de acordo com $f$ porque $T_1$ revelou mais casos em que $M$ *falha* do que $T_2$, ou seja, mais casos em que é realizada uma classificação incorreta, logo os valores analisados nas métricas são menores.

Muitas abordagens foram desenvolvidas para avaliar e comparar modelos de AM. Em nossa análise comparativa, decidimos levar em consideração várias métricas de desempenho de AM amplamente utilizadas. Como tal, selecionamos Precision, Recall, Accuracy e $F_1$. Assim, a variável dependente que é fundamental para responder ao nosso QR é *eficácia*. Logo, no âmbito deste estudo experimental, a avaliação da eficácia dos modelos será realizada avaliando diferentes métricas, pois elas são relevantes para avaliar diferentes aspectos de um problema.

Neste experimento, focamos em um único fator com dois tratamentos distintos. O fator em consideração é a abordagem de teste, enquanto os tratamentos são nossos critérios de adequação e testes aleatórios (ou seja, *k-fold cross validation*).

### 5.3.3 Instrumentação

Antes de conduzir nosso experimento, criamos objetos experimentais. Esses objetos são divididos principalmente em duas categorias: scripts Python para carregamento e processamento de *datasets*, incluindo geração de casos de teste, e scripts para análise de resultados. Empregamos o Google Colaboratory (também conhecido como Colab)[6], um ambiente baseado em nuvem que facilita a execução de scripts Python por meio de um navegador. O Google Colab permite que os usuários criem notebooks, que são essencialmente notebooks Jupyter, combinando texto e código Python executável em um único documento.

Para manipulação e análise de dados, utilizamos a biblioteca `pandas`[7]. Todos os algoritmos de AM foram implementados usando `scikit-learn`, uma biblioteca Python líder para aprendizado de máquina (MÜLLER; GUIDO, 2016). Para armazenar os arquivos em linguagem de marcação para predição de dados (Predictive Model Markup Language (PMML)[8]), que foram empregados para gerar as árvores de decisão mutantes, usamos o Google Drive[9].

[6] <https://colab.research.google.com/>
[7] <https://pandas.pydata.org>
[8] <https://www.ibm.com/docs/pt-br/db2/11.1?topic=analytics-pmml-markup-language-data-mining>
[9] Repositório do experimento: <https://drive.google.com/drive/folders/14XpZ3yoFlTW55FVHE6q2-RryqunLYFDJ?usp=sharing>

### 5.3.4 Execução

Para avaliar a eficácia da nossa abordagem baseada em mutações, foram selecionados os *datasets* utilizados no experimento anterior (consultar Tabela 10). Para preparar os dados para o experimento, carregamos todos os *datasets* no ambiente do Google Colab (ou seja, Python). Para isso, utilizamos os métodos da biblioteca *pandas* para carregamento de dados externos. Vários *datasets* tinham valores *NA*, então realizamos uma limpeza de dados (também empregando a biblioteca *pandas*) com o objetivo de criar *datasets* mais consistentes. Depois de organizar os dados, dividimos os *datasets* em duas partes: dados de treinamento (ou seja, $X$) e suas saídas ou rótulos correspondentes (ou seja, $Y$).

O próximo passo foi construir modelos de árvore de decisão a partir dos dados de treinamento: como mencionado, para ajustar os modelos aos dados de treinamento usamos `scikit-learn`, que emprega o algoritmo de árvore de classificação e regressão (CART) para treinar árvores de decisão (GÉRON, 2019). As árvores de decisão no `scikit-learn` são implementadas de tal forma que os modelos resultantes também contêm informações auxiliares que usamos nas etapas posteriores da execução do experimento. Especificamente, esses modelos também guardam informações sobre todos os nós, incluindo as regras nos nós de decisão que se assemelham às regras de código `if-else` das linguagens de programação convencionais.

Após aplicar o algoritmo de árvore de decisão a todos *datasets* completos, os modelos resultantes são armazenados em um formato baseado em XML: linguagem de marcação de modelo preditivo (PMML). Os modelos PMML para cada classificador são então usados na etapa de geração de mutação. Dada a estrutura baseada em XML dos arquivos PMML, a análise de cada modelo PMML começa na raiz e os nós de decisão sofrem mutação à medida que a árvore é atravessada. Para extrair informações do modelo do esquema XML resultante em arquivos PMML, usamos BeautifulSoup, uma biblioteca Python para análise de arquivos XML e HTML.

Uma vez geradas as árvores de decisão mutantes, nossa abordagem seleciona aleatoriamente instâncias (ou seja, linhas) do *dataset* e, em seguida, realiza uma comparação entre as previsões geradas pela árvore original e a previsão de cada árvore mutante. Especificamente, para cada árvore mutante, nossa abordagem percorre o *dataset* de forma aleatória até que encontre uma instância que mata o mutante em análise.

Esse foco nas previsões do modelo (ou seja, resultados) para comparar árvores de decisão enfatiza a seleção de instâncias que contêm valores que fazem com que as árvores mutantes façam previsões diferentes das previsões feitas pelo modelo de árvore original: mutantes que produzem previsões diferentes são classificados como mortos. No outro extremo, os mutantes que produzem previsões que correspondem às produzidas pelo modelo original são classificados como vivos.

Esta análise completa do *dataset* pode determinar se há pelo menos uma instância que pode matar a árvore mutante viva sob análise ou, inversamente, se o *dataset* não possui tal

instância. Se pelo menos uma instância puder matar o mutante, essa instância será escolhida. Se tal instância não existir, a árvore mutante em análise continua viva. Ao selecionar instâncias que matam mutantes e depois de excluir duplicatas, nossa abordagem produz um subconjunto do *dataset* contendo instâncias que matam mutantes (que assumimos ser um conjunto de teste de alta qualidade e um bom ponto de partida para validar outros modelos de classificação de AM).

A segunda fase do experimento consiste em avaliar a eficácia de modelos de AM quando testados com nosso conjunto de teste em comparação com *k-fold cross validation*. Inicialmente usamos os *datasets* originais para treinar e testar modelos k-NN, empregando um processo de *k-fold cross validation*. Posteriormente, geramos novos modelos k-NN que foram treinados nos dados dos *datasets* originais, ou seja, excluindo os dados selecionados para o conjunto de teste gerado por mutação (ou seja, um subconjunto do *dataset* que não contém instâncias que matam mutantes). Os resultados desses testes foram então comparados com os resultados de *k-fold cross validation* executada anteriormente.

## 5.4 Resultados do Experimento sobre Teste de Mutação aplicado a árvores de Decisão

Nesta seção, examinamos a eficácia de nossa abordagem baseada em mutação observando o desempenho de modelos k-NN quando são testados com *k-fold cross validation* em relação a quando são testados com nosso conjuntos de teste que matam mutantes. Realizamos esse estudo em 12 *datasets* (Tabela 10).

### *5.4.1 Características dos Objetos*

Nosso experimento considerou 6.634 modelos mutantes de árvores de decisão gerados a partir dos 12 *datasets* escolhidos. Tabela 14 mostra uma visão geral dos mutantes gerados para cada *dataset*. A tabela apresenta, para cada *dataset*, a quantidade de nós não terminais da DT, o número de mutantes relacionais criados (sob ORRN Mut), o número de mutantes constantes gerados (sob Cccr Mut), e o número total de mutantes gerados (em Total Mut). Também inclui a quantidade de dados de teste usados para eliminar os mutantes relacionais (dados de teste ORRN) e constantes (dados de teste Cccr). Além disso, lista o volume total de dados de teste, que é a combinação de "ORRN Test Data"e "Cccr Test Data"após a remoção de duplicatas. Além disso, fornece a proporção do *dataset* resultante em relação ao número de instâncias no *dataset* original (representado como Dados de Teste/*Dataset*) e, por fim, a quantidade de mutantes que não foram mortos (Live Mut) e a Mutação de Pontuação (Score Mut ). Conforme mostrado na Tabela 14, a maioria dos mutantes foi gerada a partir do modelo DT do *dataset* Phoneme (totalizando 3.116 mutantes). O *dataset* Iris levou ao modelo de árvore mais simples, o que resultou em 42 mutantes.

Dados os valores discrepantes em nossos dados em relação à quantidade de mutantes e aos dados de teste gerados, consideramos os valores medianos uma medida de tendência central mais precisa do que a média. Assim, em média (mediana), os modelos do nosso experimento possuem 155 mutantes, dos quais aproximadamente 120 correspondem à mutação do operador e 35 à mutação constante. Em relação ao tamanho do nosso conjunto de teste, temos uma média de 42 dados de teste para cada *dataset*, o que corresponde a 7,64% do *dataset* original. Quanto ao número de mutantes que não foram mortos, temos uma média de 33,5, e isso implica que usando nosso conjunto de teste, alcançamos uma pontuação mediana de mutação de 79,09%.

Como cada mutante representa um requisito de teste, a quantidade de casos de teste gerados pelos nossos critérios é proporcional ao número de nós de decisão em um determinado modelo. Portanto, em média (mediana), aplicando nossos critérios aos modelos, obtém-se aproximadamente 155 requisitos de teste (mutantes).

A segunda etapa de nosso estudo, foi avaliar o desempenho do modelo KNN testando-os com *k-fold cross validation* e com o nosso conjunto de teste.

### 5.4.2 Hipóteses de Teste

Para investigar se o DTMT leva à seleção de dados de teste mais eficazes na avaliação do desempenho de modelos AM, usamos testes paramétricos (ou seja, teste t bicaudal não pareado de duas amostras) para avaliar diferenças no valor médio das métricas usadas em nosso experimento (ou seja, Precision, Recall, Accuracy e $F_1$).

Verificamos a normalidade usando o teste de Shapiro-Wilk antes de executar os testes. De acordo com os resultados do teste Shapiro-Wilk, todas as distribuições dos dados do teste DTMT são normais. No entanto, os resultados do teste Shapiro-Wilk também mostram que os resultados de Precision da abordagem *k-fold cross validation* desviam-se significativamente da normalidade: W = 0,843, p=0,030 (conforme mostrado na Tabela 20). Portanto, empregamos um teste não paramétrico (teste de Wilcoxon) para analisar essa métrica.

Para Accuracy, Recall e $F_1$, usamos testes t bicaudais não pareados para duas amostras. Este teste paramétrico é uma abordagem bem conhecida para comparar amostras (ou seja, *datasets*) medidas nas mesmas variáveis dependentes (ou seja, métricas de desempenho escolhidas), mas sob diferentes níveis de uma variável independente (ou seja, DTMT e *k-fold cross validation*). Usamos um teste não paramétrico (teste de Wilcoxon) para a métrica Precision, conforme indicado pelo teste de Shapiro-Wilk, um desvio da normalidade. De acordo com os resultados do teste, nossos critérios podem gerar dados de teste que diferem significativamente dos dados de teste de *k-fold cross validation* na maioria dos casos. Os resultados sugerem que os dados do teste DTMT diferem significativamente dos dados originais em quase todas as métricas, exceto na Precision. Os resultados são representados por estatística $W$, pontuações $t$ e valores p na Tabela 22.

### 5.4.3 Discussão dos Resultados do Experimento sobre Teste de Mutação aplicado a árvores de Decisão

Para responder ao nosso **RQ**, examinamos a viabilidade de usar nossa abordagem baseada em mutação para conduzir a seleção de casos de teste para modelos de AM. Nosso objetivo foi selecionar entradas que matassem o maior número possível de mutantes, visto que cada mutante é um requisito de teste. A motivação geral para isso é chegar a um conjunto de casos de teste que impacte negativamente o desempenho do modelo em teste. Postulamos que é possível selecionar dados de teste mais fortes quando pretendemos matar os mutantes gerados aproveitando a estrutura interna dos modelos de árvore de decisão.

Nesse contexto, medimos o quão *eficaz* é a entrada de teste para um modelo de AM em relação aos dados nos quais o modelo foi treinado: A eficácia foi avaliada pelo impacto dos dados dos testes no desempenho preditivo do modelo em avaliação. Quanto mais prejudiciais os dados do teste foram para o desempenho do modelo, mais eficaz ele foi considerado. Aqui, o teste aleatório é *k-fold cross validation*, uma técnica que envolve dividir o *dataset* em 10 partes e repetir o treinamento e o teste do modelo 10 vezes, resultando em uma estimativa geral de desempenho.

De acordo com os resultados mostrados na Tabela 20, Tabela 22 e Figura 12, na avaliação de modelos k-NN usando nossos dados de teste gerados, nós observaram uma diminuição significativa no desempenho preditivo em quase todas as métricas avaliadas. Nossa resposta ao **RQ** centra-se nas métricas mencionadas na Subseção 5.4.2, que foram usadas como representantes para a eficácia dos dados de teste resultantes. Considerando os valores de $F_1$, os resultados sugerem que os dados de teste gerados usando nossa abordagem levam a entradas mais desafiadoras, resultando em uma diminuição significativa no desempenho em comparação com a linha de base ($F_1$ de *k-fold cross validation*).

Por exemplo, para o modelo k-NN gerado a partir do *dataset* `Oil Spill`, durante a execução de *k-fold cross validation*, o modelo produziu um valor $F_1$ de 0,94, verifica-se que este modelo teve um desempenho significativamente pior quando avaliado em relação aos dados do teste MTDT: 0,46. Para o modelo k-NN gerado a partir do *dataset* `Ionosphere`, o valor $F_1$ para *k-fold cross validation* foi 0,84 e houve uma diminuição significativamente mais pronunciada no desempenho quando o modelo foi testado com dados de teste DTMT ( 0,45). Em termos de $F_1$, há uma diferença estatisticamente significativa entre DTMT (média = 0,57) e *k-fold cross validation* (média = 0,82): $W$ = -3,12, valor p $=$ 0,005 (Tabela 22).

Os resultados apresentados na Tabela 20 e na Figura 12 demonstram uma diferença estatisticamente significativa entre DTMT e *k-fold cross validation* em todas as métricas avaliadas. Isso sugere que o DTMT pode ser empregado como uma abordagem eficaz para gerar dados de teste a partir de *datasets* existentes para detectar comportamento errôneo (ou seja, falha na previsão de múltiplas entradas de teste) em modelos de AM.

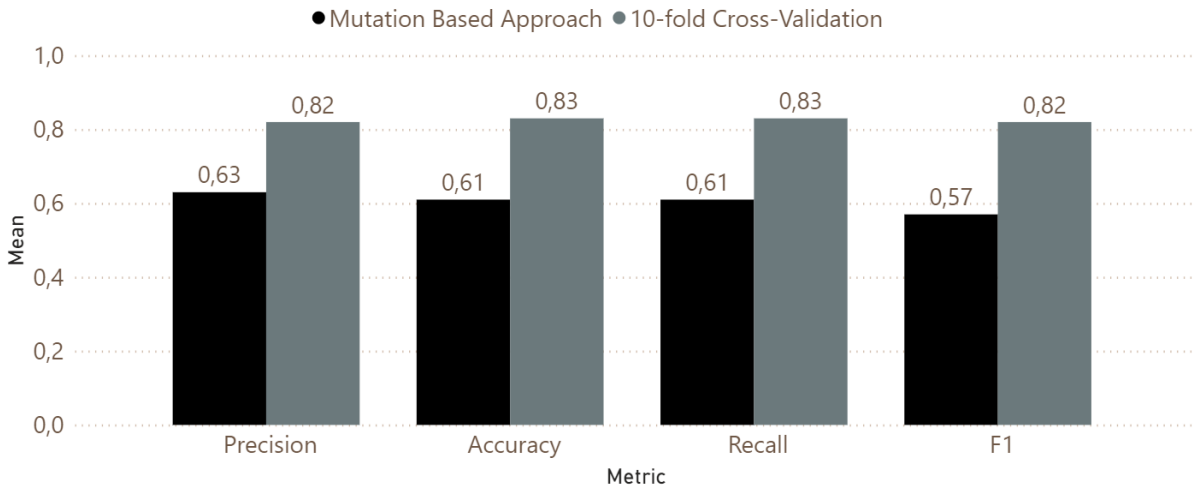



Figura 11 – Comparação dos resultados obtidos para as métricas avaliadas com DTMT e *k-fold cross validation*

## 5.5 Configuração do Experimento que compara DTC, BVA e DTMT

Decidimos comparar a eficácia de nossos três critérios baseados em árvores de decisão por meio de um experimento duplo. Dado que o teste de mutação é amplamente considerado como o "padrão ouro" em comparação aos outros tipos de teste. Levantou-se a hipótese de que o DTMT teria potencialmente um desempenho superior ao DTC e ao BVA. Assim, a ordem em que os critérios foram comparados baseia-se nesta noção de quão eficazes são estes critérios na geração de dados de teste. Inicialmente, foi comparado os dois critérios de cobertura da árvore de decisão com *k-fold cross validation*.

Isto se baseia na hipótese de que os critérios de cobertura de árvore de decisão são mais adequados para gerar conjuntos de teste para programas baseados em AM cujo componente de AM foi criado usando o algoritmo de árvore de decisão, em comparação com uma metodologia aleatória. Esta hipótese é baseada no fato de que os critérios aproveitam a estrutura interna e as informações de decisão dos modelos de árvore de decisão durante a identificação dos requisitos de teste.

A suposição é que os nós de decisão desempenham um papel fundamental na determinação do comportamento do modelo. Consequentemente, dados de teste mais eficazes podem ser obtidos aproveitando os dados de tomada de decisão inerentes ao modelo em teste. Após esta comparação inicial, o critério de cobertura da árvore de decisão mais eficaz foi então comparado com o DTMT para determinar melhor o desempenho relativo. Portanto, o experimento foi projetado para responder às seguintes questões de pesquisa (RQs):

**$RQ_1$:** *Como a eficácia dos conjuntos de testes derivados do DTC e do BVA se compara aos casos de testes aleatórios de k-fold cross validation?*

**$RQ_2$:** *Como a eficácia dos conjuntos de testes derivados do DTMT se compara ao critério de cobertura de árvore de decisão de melhor desempenho?*

No contexto deste estudo experimental, a eficácia é quantificada usando as mesmas métricas de desempenho de AM mencionadas nos experimentos anteriores.

### 5.5.1 Escopo

O escopo deste experimento é definido de acordo com a configuração dos objetivos, que é baseado no modelo GQM (WOHLIN *et al.*, 2012) da seguinte forma:

**Analisar** três critérios baseados em árvore de decisão (ou seja, DTC, BVA e DTMT)
**para fins de** avaliação
**com relação à sua** eficácia
**do ponto de vista** do pesquisador
**no contexto de** testar programas baseados em AM.

### 5.5.2 Formulação de Hipóteses

Para permitir a aplicação de testes estatísticos, refinamos ainda mais nossos QRs em hipóteses específicas. Definimos a previsão para $\mathbf{RQ}_1$ como: nossos critérios de cobertura de árvore de decisão (DTCC) são mais eficazes que testes aleatórios. Assim, $\mathbf{RQ}_1$ foi transformado nas seguintes hipóteses:

**Hipótese nula, $H_{0-DTCC \times aleatrio}$:** não há diferença na eficácia entre os critérios de cobertura da árvore de decisão e testes aleatórios.

**Hipótese alternativa, $H_{1-DTCC \times aleatrio}$:** há uma diferença significativa na eficácia entre os critérios de cobertura da árvore de decisão e os testes aleatórios.

Quanto a $\mathbf{RQ}_2$, definimos sa previsão como: o critério baseado em mutação é mais eficaz do que o critério de cobertura de árvore de decisão de melhor desempenho. Assim, $\mathbf{RQ}_2$ foi transformado nas seguintes hipóteses:

**Hipótese nula, $H_{0-DTMT \times DTCC}$:** não há diferença na eficácia entre o critério de cobertura da árvore de decisão com melhor desempenho e o DTMT.

**Hipótese alternativa, $H_{1-DTMT \times DTCC}$:** há uma diferença significativa na eficácia entre o critério de cobertura da árvore de decisão com melhor desempenho e o DTMT.

### 5.5.3 Instrumentação e Execução

Para instrumentação, foram utilizadas os mesmo recursos dos experimento anteriores, ver subseções 5.1.3 e 5.3.3.

Para avaliar a eficácia da abordagem baseada em mutações quando comparada com os dois critérios de cobertura da árvore de decisão, foi selecionada uma amostra de 16 conjuntos de dados disponíveis publicamente (ver Tabela 17).

Para a primeira etapa deste experimento duplo, a execução seguiu conforme descrito na Subseção 5.1.4. Em seguida, na segunda parte do experimento duplo, comparamos o critério de cobertura de árvore de decisão com o DTMT, semelhante ao que foi feito na Subseção 5.3.4, porém comparando com o melhor critério de cobertura de árvore de decisão em vez do *k-fold cross validation*. Os resultados desta análise são discutidos na seção a seguir, Seção 5.6.

## 5.6 Resultados do Experimento que Compara DTC, BVA e DTMT

Nesta seção, examinamos a eficácia dos critérios observando o desempenho dos modelos quando eles são validados aleatoriamente versus quando testados com conjuntos de testes adaptados para cada critério. Em seguida, comparamos o resultado do critério de cobertura da árvore de decisão de melhor desempenho com os resultados da aplicação da DTMT.

### 5.6.1 Critérios de cobertura da árvore de decisão: características dos objetos

A Tabela 18 fornece uma visão geral das propriedades básicas dos modelos de árvore de decisão gerados a partir dos conjuntos de dados. Conforme mostrado na Tabela 18, o modelo resultante com o maior número de nós foi gerado a partir do *dataset* `Cancer Prediction`: 2.161 nós, dos quais 1.181 são nós folha. `Iris` é o *dataset* que levou à criação do modelo mais simples, que é composto por 17 nós, dos quais 9 são folhas.

Dados os valores discrepantes em nossos dados em relação à altura dos modelos de árvore resultantes (ou seja, quantidade de nós), consideramos os valores medianos da Tabela 18 uma medida mais precisa de tendência central do que a média. Assim, em média (mediana), os modelos do nosso experimento possuem 49 nós, dos quais 25 são folhas.

Cada caminho raiz-folha dentro de um modelo de árvore de decisão constitui um critério de teste. Consequentemente, o volume de casos de teste gerados pelos critérios de cobertura da árvore de decisão é diretamente proporcional à quantidade de nós folha presentes em um determinado modelo. Assim, a aplicação dos critérios aos modelos, em média (mediana), gerou aproximadamente 25 requisitos de teste, conforme mostrado na Tabela 18.

### 5.6.2 DTMT: Características dos Objetos

O experimento levou em consideração 12.333 árvores mutantes geradas a partir dos 16 conjuntos de dados escolhidos. A tabela 19 mostra uma visão geral da quantidade de mutantes gerados para cada conjunto de dados. A tabela apresenta, para cada *dataset*, a quantidade de nós não terminais, o número de mutantes relacionais criados (ORRN Mut), o número de mutantes constantes gerados (Cccr Mut), e o número total de mutantes gerados (Total Mut). Também inclui a quantidade de dados de teste usados para eliminar os mutantes relacionais (dados de teste ORRN) e constantes (dados de teste Cccr). Além disso, Table 19 lista o volume total de dados de teste, que é a combinação de "ORRN Test Data"e "Cccr Test Data"após a remoção de duplicatas. Além disso, fornece a proporção do conjunto de testes resultante em relação ao número de instâncias no conjunto de dados original (representado como "Test Data/*Dataset*") e, finalmente, a quantidade de mutantes que não foram mortos ("Live Mut") e a Mutação de Pontuação ("Score Mut"). Conforme mostrado na Tabela 19, a maioria dos mutantes foi gerada a partir do modelo de árvore de decisão do *dataset* `Cancer Prediction` (totalizando 5.508 mutantes). O *dataset* `Iris` levou ao modelo de árvore mais simples, o que resultou em 42 mutantes.

Dados os valores discrepantes nos dados em relação ao número de mutantes e aos dados de teste gerados, foram considerados os valores medianos uma medida de tendência central mais precisa do que a média. Assim, em média (mediana), os modelos do experimento possuem 125 mutantes. Em relação ao tamanho dos conjuntos de teste, há uma média de 34 dados de teste para cada *dataset*, o que corresponde a 7,21% do conjunto de dados original. Quanto ao número de mutantes que não foram mortos, obteve-se uma média de 27,5, e isso implica que usando o conjunto de dados de teste de DTMT, é possível alcançar um score de mutação mediano de 79,42%.

Como cada mutante representa um requisito de teste, a quantidade de casos de teste gerados pelo critério DTMT é proporcional ao número de nós de decisão em um determinado modelo. Portanto, aplicando este critério aos modelos resultantes, obtém-se em média (mediana), aproximadamente 125 requisitos de teste.

### 5.6.3 Teste de Hipóteses

Consistente com a metodologia empregada em nossos experimentos anteriores, para investigar qual critério baseado em árvore de decisão leva à seleção de dados de teste que são mais eficazes na avaliação do desempenho de modelos de AM, usamos testes paramétricos (ou seja, teste t bicaudal não pareado de duas amostras) e testes não paramétricos para avaliar diferenças no valor médio das métricas usadas em nosso experimento (ou seja, Precision, Recall, Accuracy e $F_1$).

Verificamos a normalidade usando o teste de Shapiro-Wilk antes de executar os testes. De acordo com os resultados do teste Shapiro-Wilk, todas as distribuições dos resultados da

aplicação dos dados dos testes DTC, BVA e DTMT são normais. No entanto, os resultados do teste de Shapiro-Wilk também mostram que todas as distribuições dos resultados da abordagem 10-fold cross-validation se desviam da normalidade (como mostrado na Tabela 20). Consequentemente, recorremos ao emprego de um teste não paramétrico, especificamente o teste dos postos sinalizados de Wilcoxon, com o propósito de comparar esta abordagem com os demais critérios.

A análise estatística resumida na Tabela 21 apresenta uma comparação das métricas de desempenho entre DTC, BVA e 10-fold cross-validation usando o teste de postos sinalizados de Wilcoxon.

Os resultados de Precision indicam diferenças estatisticamente significativas ao comparar DTC e BVA com 10-fold cross-validation, com valores de $W$ de 5,0 ($p \leq 0{,}0018$) e 7,0 ($p \leq 0{,}0006$), respectivamente. Esses baixos valores de p sugerem uma forte significância nas diferenças de pontuações de Precision entre os critérios quando comparados à validação cruzada de 10 vezes10-fold cross-validation.

Para Recall, a análise também parece sugerir que há uma diferença, especialmente entre DTC e 10-fold cross-validation (W = 0,0, $p \leq 0{,}0007$), e também entre BVA e 10-fold cross-validation (W = 5,0, $p \leq 0{,}0003$).

Semelhante ao recall, a métrica de Accuracy mostra uma diferença significativa no desempenho entre DTC (W = 0,0, $p \leq 0{,}0007$) e BVA (W = 5,0, $p \leq 0{,}0003$) em comparação com a 10-fold cross-validation. $F_1$ também reflete uma disparidade significativa, com DTC exibindo um valor W de 1,0 ($p \leq 0{,}0008$) e BVA um valor W de 5,0 ($p \leq 0{,}0003$) quando comparado com o resultado de 10-fold cross-validation. Esses resultados corroboram ainda mais as diferenças significativas observadas nas demais métricas.

No geral, os testes estatísticos revelam que tanto o DTC quanto o BVA apresentam diferenças significativas nas métricas de desempenho quando comparados à 10-fold cross-validation. Os resultados indicam uma tendência consistente de diferenças significativas entre essas abordagens, sugerindo que o DTC e o BVA podem oferecer vantagens distintas em relação à tradicional 10-fold cross-validation em termos de avaliação do modelo.

A hipótese nula, $H_{0-DTCC \times aleatrio}$, não postula nenhuma diferença na eficácia entre nossos critérios de cobertura da árvore de decisão e testes aleatórios. No entanto, os valores p em todas as métricas para DTC e BVA contra 10-fold cross-validation sugerem fortemente a rejeição desta hipótese nula. A hipótese alternativa, $H_{1-DTCC \times aleatrio}$, que afirma que existe uma diferença significativa na eficácia é corroborada pelos resultados das comparações realizadas. Os resultados de Precision, Recall, Accuracy e $F_1$ indicam que tanto o DTC quanto o BVA são significativamente diferentes (e provavelmente mais eficazes) do que os testes aleatórios na avaliação do desempenho dos modelos de AM.

Os resultados dos testes estatísticos apresentados na Tabela 22, comparando DTMT com BVA, fornecer dados perspicazes sobre a eficácia desses dois critérios baseados em árvores de

decisão.

Primeiramente, os resultados do teste t de amostras pareadas para Recall, Accuracy e $F_1$ indicam que não há diferenças estatisticamente significativas em termos dessas métricas entre DTMT e BVA. Isso sugere que, em termos de Recall, Accuracy e $F_1$, DTMT e BVA têm desempenho comparável. No entanto, a métrica Precision mostra uma tendência diferente. Com um valor p de 0,041, há uma diferença estatisticamente significativa entre DTMT e BVA. Isto sugere que, em termos de Precision, DTMT e BVA não têm desempenho igual, e de acordo com os resultados um critério é mais eficaz que o outro.

No contexto da sua questão e hipóteses de pesquisa, esses resultados oferecem insights diferenciados. A hipótese nula, $H_{0-DTMT \times DTCC}$, não postula nenhuma diferença na eficácia entre o critério de cobertura da árvore de decisão de melhor desempenho (ou seja, BVA) e o DTMT. Considerando que o BVA é o critério de cobertura da árvore de decisão com melhor desempenho de acordo com os resultados da primeira comparação, os resultados de Recall, Accuracy e $F_1$ não fornecem evidências suficientes para rejeitar a hipótese nula - indicando que o DTMT não supera significativamente o BVA. No entanto, os resultados de Precision permitem a rejeição da hipótese nula, sugerindo que há uma diferença significativa na eficácia entre DTMT e BVA em termos de Precision. Isso se alinha com a hipótese alternativa, $H_{1-DTMT \times DTCC}$, indicando que há uma diferença significativa entre DTMT e BVA. Vale a pena notar que estes resultados implicam que o BVA supera o DTMT em Precision, um desempenho inferior do modelo resultante usando os dados de teste gerados é indicativo de uma abordagem de teste mais eficaz.

No geral, os resultados apresentam uma contradição sutil com a previsão de que o DTMT superaria o critério de cobertura da árvore de decisão de melhor desempenho em eficácia. Embora o DTMT não pareça ter um desempenho tão bom quanto o BVA em termos de Precision, demonstra desempenho comparável em Recall, Accuracy e $F_1$. Este padrão sugere que a eficácia do DTMT, quando comparado com outros critérios de cobertura da árvore de decisão, pode variar dependendo da métrica específica em consideração.

## 5.7 Ameaças à Validade

Ao longo dos experimentos, foram tomadas várias precauções para aliviar ameaças potenciais à validade do nosso experimento e às suas descobertas. No entanto, como é típico na maioria dos estudos experimentais, é impossível remediar completamente todas as ameaças à validade.

Os *datasets* podem potencialmente ameaçar a validade externa, uma vez que podem não representar adequadamente a população-alvo. Os *datasets* escolhidos são menores e mais limpos do que aqueles comumente encontrados na prática. Portanto, não podemos descartar a possibilidade de que os resultados pudessem ter sido diferentes se *datasets* maiores tivessem sido selecionados. As medidas utilizadas no experimento podem ser consideradas uma ameaça

potencial à validade de construção, pois podem não ser adequadas para avaliar os efeitos que nos propusemos a investigar. Especificamente, Precision, Recall, Accuracy e $F_1$ podem não ser os principais preditores da adequação dos dados de teste para programas baseados em AM. Porém, vale ressaltar que essas quatro medidas são amplamente utilizadas para avaliar modelos de AM, o que mitiga o risco dessa ameaça.

Uma limitação fundamental da nossa abordagem é que ela assume que apenas valores numéricos aparecem nos nós de decisão. Além disso, uma ameaça específica do experimento sobre teste de mutação aplicado a árvores de decisão é que a busca exaustiva no *dataset* original atualmente é feita de forma sequencial, o que impacta o desempenho na criação do conjunto de casos de teste de DTMT.

## 5.8 Considerações Finais

Neste capítulo foram apresentados três experimentos distintos que avaliam o uso de árvores de decisão para selecionar dados de teste para AM. Cada experimento foi planejado e direcionado para investigar questões de pesquisas específicas, proporcionando uma ampla gama de dados para análise. No entanto, nas considerações finais deste capítulo, optou-se por concentrar a discussão no último experimento apresentado. Este último experimento foi selecionado devido à sua capacidade de englobar os resultados obtidos nos dois experimentos anteriores, consolidando assim uma visão mais abrangente dos critérios de testes baseados na estrutura interna de modelos de árvores de decisão.

Para abordar a $\mathbf{RQ}_1$, analisamos a eficácia dos nossos critérios de cobertura da árvore de decisão em selecionar entradas de teste capazes de impactar negativamente o desempenho do modelo em avaliação. Nossa hipótese foi fundamentada no potencial de selecionar dados de teste mais fortes usando nossos critérios, uma vez que a seleção de dados de teste é baseada na estrutura interna dos modelos de árvore de decisão, em oposição à seleção aleatória de dados de teste.

Nesse contexto, medimos o quão *eficaz* é a entrada de teste para um modelo de AM em relação aos dados nos quais o modelo foi treinado: como mencionado, a eficácia foi avaliada pelo impacto dos dados de teste no desempenho preditivo do modelo em avaliação. Quanto mais prejudiciais os dados de teste foram para o desempenho do modelo, mais eficaz foi considerado. No contexto de $\mathbf{RQ}_1$, o teste aleatório é 10-fold cross-validation, uma técnica que envolve dividir o conjunto de dados em 10 partes e repetir o treinamento e o teste do modelo 10 vezes, resultando em uma estimativa geral de desempenho.

Os resultados da comparação entre os critérios de cobertura da árvore de decisão (DTC e BVA) e 10-fold cross-validation revelam diferenças significativas nas métricas de desempenho. Nitidamente, tanto o DTC quanto o BVA demonstram resultados das métricas Precision, Recall, Accuracy e $F_1$ superiores em comparação com 10-fold cross-validation. Isto sugere que os crité-

rios de cobertura da árvore de decisão, particularmente DTC e BVA, pode oferecer mecanismos mais robustos para avaliar modelos de AM do que os métodos tradicionais de validação cruzada. Além disso, os resultados parecem indicar que o BVA supera o DTC na seleção de insumos de teste capazes de impactar negativamente o desempenho do modelo avaliado.

Ao analisar os resultados da primeira parte do nosso experimento, foi estabelecido que o BVA representa o critério de cobertura da árvore de decisão com melhor desempenho. Consequentemente, para responder a **RQ**$_2$, procedeu-se à realização de uma análise comparativa entre BVA e DTMT. Os resultados experimentais da comparação do DTMT com o BVA, especialmente em termos de Precision, oferecem uma visão multifacetada da eficácia de um critério de cobertura de árvore de decisão e de um critério baseado em mutação. Embora DTMT e BVA apresentem desempenho comparável em Recall, Accuracy e $F_1$, uma diferença significativa é observada em Precision. Isto sugere que o BVA pode ter uma vantagem sobre o DTMT em cenários focados na Precision. Esta descoberta contradiz parcialmente a noção de que o DTMT poderia ser mais eficaz do que o critério de cobertura da árvore de decisão com melhor desempenho.

A Figura 12 fornece uma visão geral da eficácia das quatro abordagens. De acordo com os resultados mostrados na Figura 12, 10-fold cross-validation mostrou o pior desempenho na seleção de casos de teste mais fortes, enquanto DTMT e DTC obtiveram desempenhos semelhantes. Por outro lado, O BVA destacou-se como o mais eficaz na identificação de casos de teste mais fortes.

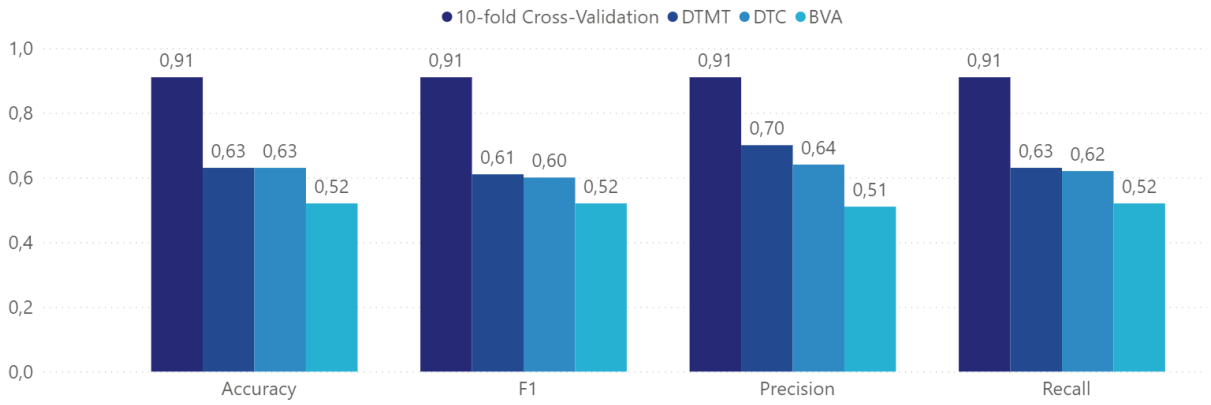



Figura 12 – Visão geral da eficácia das quatro abordagens com a mediana para cada métrica.

Na próxima seção são apresentadas as conclusões desta dissertação.

Tabela 14 – Visão geral dos mutantes e do conjunto de teste gerado no experimento DTMT.

| *Datasets* | Nós Decisão | Mut ORRN | Mut Cccr | Mut Total | Teste ORRN | Teste Cccr | Teste Total | Teste/*Dataset* (%) | Mut Vivo | Score Mut |
|---|---|---|---|---|---|---|---|---|---|---|
| Phoneme | 521 | 2084 | 1032 | 3116 | 516 | 465 | 624 | 11.55 | 532 | 82.93 |
| Mammography | 159 | 636 | 306 | 942 | 151 | 165 | 205 | 1.83 | 173 | 81.63 |
| Pima Indians Diabetes | 126 | 504 | 235 | 739 | 107 | 113 | 156 | 20.31 | 140 | 81.06 |
| Haberman's Survival | 101 | 404 | 187 | 591 | 69 | 92 | 111 | 36.27 | 116 | 80.37 |
| Cleveland Heart Disease | 100 | 400 | 148 | 548 | 79 | 75 | 103 | 34.68 | 118 | 78.47 |
| Oil Spill | 34 | 136 | 26 | 162 | 33 | 17 | 39 | 4.16 | 38 | 76.54 |
| Banknote Authentication | 26 | 104 | 44 | 148 | 33 | 26 | 45 | 3.28 | 29 | 80.41 |
| Ionosphere | 22 | 88 | 14 | 102 | 22 | 7 | 26 | 7.41 | 26 | 74.51 |
| Sonar, Mines vs. Rocks | 22 | 88 | 4 | 92 | 20 | 3 | 21 | 10.10 | 22 | 76.09 |
| Breast Cancer Wisconsin | 21 | 84 | 14 | 98 | 24 | 11 | 29 | 5.10 | 23 | 76.53 |
| Wine Recognition | 11 | 44 | 10 | 54 | 12 | 8 | 14 | 7.87 | 11 | 79.63 |
| Iris | 8 | 32 | 10 | 42 | 7 | 6 | 10 | 6.67 | 8 | 80.95 |
| | **Estatísticas Descritivas** | | | | | | | | | |
| **Max** | 521 | 2,084 | 1,032 | 3116 | 516 | 465 | 624 | 36.27 | 532 | 82.93 |
| **Min** | 8 | 32 | 4 | 42 | 7 | 3 | 10 | 1.83 | 8 | 74.51 |
| **Média** | 95.92 | 383.67 | 169.17 | 552.83 | 89.42 | 82.33 | 115.25 | 12.44 | 103 | 79.09 |
| **Mediana** | 30 | 120 | 35 | 155 | 33 | 21.5 | 42 | 7.64 | 33.5 | 80.00 |
| **Std Dev** | 143.28 | 573.13 | 290.86 | 863.73 | 141.23 | 131.50 | 171.91 | 11.79 | 146.72 | 2.62 |

Tabela 15 – Resultados obtidos pelas métricas avaliadas para a abordagem DTMT e *k-fold cross validation*.

| ***Dataset*** | **Abordagem Baseada em Mutação** | | | | ***k-fold cross validation*** | | | |
|---|---|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **Accuracy** | **$F_1$** | **Precision** | **Recall** | **Accuracy** | **$F_1$** |
| Phoneme | 0.64 | 0.64 | 0.64 | 0.63 | 0.89 | 0.90 | 0.90 | 0.89 |
| Mammography | 0.70 | 0.70 | 0.70 | 0.67 | 0.99 | 0.99 | 0.99 | 0.99 |
| Pima Indians Diabetes | 0.48 | 0.49 | 0.49 | 0.48 | 0.68 | 0.68 | 0.68 | 0.68 |
| Haberman's Survival | 0.62 | 0.62 | 0.62 | 0.57 | 0.66 | 0.69 | 0.69 | 0.67 |
| Cleveland Heart Disease | 0.13 | 0.29 | 0.29 | 0.17 | 0.35 | 0.48 | 0.48 | 0.40 |
| Oil Spill | 0.39 | 0.56 | 0.56 | 0.46 | 0.93 | 0.95 | 0.95 | 0.94 |
| Banknote Authentication | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| Ionosphere | 0.80 | 0.50 | 0.50 | 0.45 | 0.87 | 0.85 | 0.85 | 0.84 |
| Sonar, Mines vs. Rocks | 0.69 | 0.62 | 0.62 | 0.59 | 0.84 | 0.83 | 0.83 | 0.82 |
| Breast Cancer Wisconsin | 0.71 | 0.69 | 0.69 | 0.69 | 0.93 | 0.93 | 0.93 | 0.93 |
| Wine Recognition | 0.71 | 0.50 | 0.50 | 0.49 | 0.70 | 0.69 | 0.69 | 0.67 |
| Iris | 0.73 | 0.70 | 0.70 | 0.71 | 0.97 | 0.96 | 0.97 | 0.96 |
| | **Estatísticas Descritivas e Teste de Normalidade** | | | | | | | |
| **Max** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| **Min** | 0.13 | 0.29 | 0.29 | 0.17 | 0.35 | 0.48 | 0.48 | 0.40 |
| **Mediana** | 0.63 | 0.61 | 0.61 | 0.57 | 0.82 | 0.83 | 0.83 | 0.82 |
| **Std Dev** | 0.22 | 0.17 | 0.17 | 0.20 | 0.19 | 0.16 | 0.16 | 0.18 |
| **Shapiro-Wilk (W)** | W=0.908, p=0.202 | W=0.925, p=0.330 | W=0.926, p=0.336 | W=0.939, p=0.486 | W=0.843, p=0.030 | W=0.885, p=0.102 | W=0.885, p=0.102 | W=0.871, p=0.068 |

Tabela 16 – Resumo dos resultados dos testes estatísticos para DTMT e 10-fold cross validation.

| | **Teste de Amostras Pareadas ($t$)** |
|---|---|
| **Métrica** | **DTMT x *k-fold cross validation*** |
| **Accuracy** | $t$ = -3.24, p-value = 0.003 |
| **Recall** | $t$ = -3.23, p-value = 0.003 |
| **$F_1$** | $t$ = -3.12, p-value = 0.005 |
| | **Teste de Wilcoxon para Amostras Pareadas ($W$)** |
| **Métrica** | **DTMT x *k-fold cross validation*** |
| **Precision** | $W$ = 23, p-value = 0.374 |

Tabela 17 – Visão geral dos dados gerados no experimento.

| ***Datasets*** | **Exemplos** | **Atributos** | **Classes** | **Exemplos por Classe** |
|---|---|---|---|---|
| Cancer Prediction | 10,001 | 5 | 2 | [9,036, 965] |
| Phoneme | 5,404 | 6 | 2 | [3,818, 1,586] |
| Mammography | 11,183 | 7 | 2 | [10,923, 260] |
| Pima Indians Diabetes | 768 | 9 | 2 | [500, 268] |
| Haberman's Survival | 306 | 4 | 2 | [225, 81] |
| Cleveland Heart Disease | 297 | 15 | 2 | [160, 137] |
| Oil Spill | 937 | 50 | 2 | [896, 41] |
| Banknote Authentication | 1,372 | 5 | 2 | [762, 610] |
| Ionosphere | 351 | 35 | 2 | [225, 126] |
| Sonar, Mines vs. Rocks | 208 | 61 | 2 | [97, 111] |
| Breast Cancer Wisconsin | 569 | 31 | 2 | [212, 357] |
| Penguins | 345 | 7 | 3 | [152, 69, 124] |
| Hawks | 909 | 19 | 3 | [71, 577, 261] |
| Wheat Seeds | 200 | 7 | 3 | [66, 68, 66] |
| Wine Recognition | 178 | 14 | 3 | [59, 71, 48] |
| Iris | 150 | 5 | 3 | [50, 50, 50] |

Tabela 18 – Número de nós nos modelos de árvore de decisão resultantes e número de casos de teste gerados pela aplicação dos critérios DTC e BVA.

| ***Datasets*** | **Nós** | **DTC/BVA** |
|---|---|---|
| Cancer Prediction | 2,161 | 1,081 |
| Phoneme | 1,041 | 521 |
| Mammography | 319 | 160 |
| Pima Indians Diabetes | 255 | 128 |
| Haberman's Survival | 207 | 104 |
| Cleveland Heart Disease | 99 | 50 |
| Oil Spill | 69 | 35 |
| Banknote Authentication | 53 | 27 |
| Ionosphere | 45 | 23 |
| Sonar, Mines vs. Rocks | 45 | 23 |
| Breast Cancer Wisconsin | 43 | 22 |
| Penguins | 31 | 6/16 |
| Hawks | 27 | 7/14 |
| Wheat Seeds | 25 | 13 |
| Wine Recognition | 23 | 12 |
| Iris | 17 | 9 |
| | **Estatísticas Descritivas** | |
| **Max** | 2,161 | 1,181 |
| **Min** | 17 | 9 |
| **Média** | 278.75 | 139.88 |
| **Mediana** | 49.00 | 25.00 |
| **Std Dev** | 562.52 | 281.26 |

Tabela 19 – Visão geral dos dados gerados no experimento.

| *Datasets* | Nós Decisão | Mut ORRN | Mut Cccr | Mut Total | Teste ORRN | Teste Cccr | Teste Total | Teste/*Dataset* (%) | Mut Vivo | Score Mut |
|---|---|---|---|---|---|---|---|---|---|---|
| Cancer Prediction | 1,080 | 4,320 | 1,188 | 5,508 | 808 | 626 | 984 | 9.84 | 1,145 | 79.21 |
| Phoneme | 521 | 2,084 | 1,032 | 3,116 | 516 | 465 | 624 | 11.55 | 532 | 82.93 |
| Mammography | 159 | 636 | 306 | 942 | 151 | 165 | 205 | 1.83 | 173 | 81.63 |
| Pima Indians Diabetes | 126 | 504 | 235 | 739 | 107 | 113 | 156 | 20.31 | 140 | 81.06 |
| Haberman's Survival | 101 | 404 | 187 | 591 | 69 | 92 | 111 | 36.27 | 116 | 80.37 |
| Cleveland Heart Disease | 100 | 400 | 148 | 548 | 79 | 75 | 103 | 34.68 | 118 | 78.47 |
| Oil Spill | 34 | 136 | 26 | 162 | 33 | 17 | 39 | 4.16 | 38 | 76.54 |
| Banknote Authentication | 26 | 104 | 44 | 148 | 33 | 26 | 45 | 3.28 | 29 | 80.41 |
| Ionosphere | 22 | 88 | 14 | 102 | 22 | 7 | 26 | 7.41 | 26 | 74.51 |
| Sonar, Mines vs. Rocks | 22 | 88 | 4 | 92 | 20 | 3 | 21 | 10.10 | 22 | 76.09 |
| Breast Cancer Wisconsin | 21 | 84 | 14 | 98 | 24 | 11 | 29 | 5.10 | 23 | 76.53 |
| Penguins | 15 | 60 | 15 | 75 | 18 | 9 | 20 | 5.80 | 17 | 77.33 |
| Hawks | 13 | 52 | 4 | 56 | 12 | 2 | 13 | 1.43 | 15 | 73.21 |
| Wheat Seeds | 12 | 48 | 12 | 60 | 11 | 6 | 14 | 7.00 | 12 | 80.00 |
| Wine Recognition | 11 | 44 | 10 | 54 | 12 | 8 | 14 | 7.87 | 11 | 79.63 |
| Iris | 8 | 32 | 10 | 42 | 7 | 6 | 10 | 6.67 | 8 | 80.95 |
| | **Estatísticas Descritivas** | | | | | | | | | |
| **Max** | 1,080 | 4,320 | 1,188 | 5,508 | 808 | 626 | 984 | 36.27 | 1,145 | 82.93 |
| **Min** | 8 | 32 | 4 | 42 | 7 | 2 | 10 | 1.43 | 8 | 73.21 |
| **Média** | 141.94 | 567.75 | 203.06 | 770.81 | 120.13 | 101.94 | 150.88 | 10.83 | 151.56 | 78.68 |
| **Mediana** | 24 | 96 | 20.50 | 125 | 28.50 | 14 | 34 | 7.21 | 27.50 | 79.42 |
| **Std Dev** | 280.57 | 1,122.29 | 367.51 | 1,476.88 | 221.80 | 182.08 | 269.48 | 10.61 | 295.32 | 2.73 |

Tabela 20 – Resultados das métricas avaliadas para as abordagens DTC, BVA, DTMT approach e 10-fold cross-validation.

| *Dataset* | DTC | | | | BVA | | | | DTMT | | | | 10-fold Cross-Validation | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Precision** | **Recall** | **Accuracy** | **$F_1$** | **Precision** | **Recall** | **Accuracy** | **$F_1$** | **Precision** | **Recall** | **Accuracy** | **$F_1$** | **Precision** | **Recall** | **Accuracy** | **$F_1$** |
| Cancer Prediction | 0.76 | 0.85 | 0.85 | 0.80 | 0.99 | 0.97 | 0.97 | 0.98 | 0.84 | 0.88 | 0.88 | 0.84 | 0.82 | 0.88 | 0.88 | 0.85 |
| Phoneme | 0.61 | 0.61 | 0.61 | 0.60 | 0.52 | 0.52 | 0.52 | 0.52 | 0.64 | 0.64 | 0.64 | 0.63 | 0.89 | 0.90 | 0.90 | 0.89 |
| Mammography | 0.63 | 0.62 | 0.63 | 0.57 | 0.58 | 0.59 | 0.59 | 0.55 | 0.70 | 0.70 | 0.70 | 0.67 | 0.99 | 0.99 | 0.99 | 0.99 |
| Pima Indians Diabetes | 0.45 | 0.46 | 0.46 | 0.45 | 0.52 | 0.53 | 0.53 | 0.50 | 0.48 | 0.49 | 0.49 | 0.48 | 0.68 | 0.68 | 0.68 | 0.68 |
| Haberman's Survival | 0.56 | 0.56 | 0.56 | 0.48 | 0.63 | 0.64 | 0.64 | 0.62 | 0.62 | 0.62 | 0.62 | 0.57 | 0.66 | 0.69 | 0.69 | 0.67 |
| Cleveland Heart Disease | 0.45 | 0.44 | 0.44 | 0.44 | 0.59 | 0.56 | 0.56 | 0.55 | 0.13 | 0.29 | 0.29 | 0.17 | 0.35 | 0.48 | 0.48 | 0.40 |
| Oil Spill | 0.21 | 0.40 | 0.40 | 0.28 | 0.24 | 0.49 | 0.49 | 0.32 | 0.39 | 0.56 | 0.56 | 0.46 | 0.93 | 0.95 | 0.95 | 0.94 |
| Banknote Authentication | 1.00 | 1.00 | 1.00 | 1.00 | 0.47 | 0.48 | 0.48 | 0.47 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| Ionosphere | 0.78 | 0.48 | 0.48 | 0.39 | 0.15 | 0.39 | 0.39 | 0.22 | 0.80 | 0.50 | 0.50 | 0.45 | 0.87 | 0.85 | 0.85 | 0.84 |
| Sonar, Mines vs. Rocks | 0.63 | 0.64 | 0.64 | 0.61 | 0.47 | 0.50 | 0.50 | 0.48 | 0.69 | 0.62 | 0.62 | 0.59 | 0.84 | 0.83 | 0.83 | 0.82 |
| Breast Cancer Wisconsin | 0.65 | 0.64 | 0.64 | 0.64 | 0.64 | 0.64 | 0.64 | 0.64 | 0.71 | 0.69 | 0.69 | 0.69 | 0.93 | 0.93 | 0.93 | 0.93 |
| Penguins | 0.67 | 0.67 | 0.67 | 0.61 | 0.50 | 0.50 | 0.50 | 0.33 | 0.86 | 0.90 | 0.90 | 0.88 | 0.97 | 0.97 | 0.97 | 0.97 |
| Hawks | 0.74 | 0.71 | 0.71 | 0.70 | 0.36 | 0.14 | 0.14 | 0.57 | 0.67 | 0.69 | 0.69 | 0.67 | 0.95 | 0.95 | 0.95 | 0.95 |
| Wheat Seeds | 0.58 | 0.62 | 0.62 | 0.59 | 0.48 | 0.62 | 0.62 | 0.51 | 0.56 | 0.57 | 0.57 | 0.56 | 0.92 | 0.92 | 0.92 | 0.92 |
| Wine Recognition | 0.64 | 0.33 | 0.33 | 0.32 | 0.29 | 0.42 | 0.42 | 0.34 | 0.71 | 0.50 | 0.50 | 0.49 | 0.70 | 0.69 | 0.69 | 0.67 |
| Iris | 0.69 | 0.67 | 0.67 | 0.67 | 0.69 | 0.67 | 0.67 | 0.67 | 0.73 | 0.70 | 0.70 | 0.71 | 0.97 | 0.96 | 0.97 | 0.96 |
| | **Estatísticas Descritivas e Teste de Normalidade** | | | | | | | | | | | | | | | |
| **Max** | 1.00 | 1.00 | 1.00 | 1.00 | 0.99 | 0.97 | 0.97 | 0.98 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| **Min** | 0.21 | 0.33 | 0.33 | 0.28 | 0.15 | 0.14 | 0.14 | 0.22 | 0.13 | 0.29 | 0.29 | 0.17 | 0.35 | 0.48 | 0.48 | 0.40 |
| **Média** | 0.63 | 0.61 | 0.61 | 0.57 | 0.51 | 0.54 | 0.54 | 0.52 | 0.66 | 0.65 | 0.65 | 0.62 | 0.84 | 0.85 | 0.85 | 0.84 |
| **Mediana** | 0.64 | 0.62 | 0.63 | 0.60 | 0.51 | 0.52 | 0.52 | 0.52 | 0.70 | 0.63 | 0.63 | 0.61 | 0.91 | 0.91 | 0.91 | 0.91 |
| **Std Dev** | 0.17 | 0.17 | 0.17 | 0.18 | 0.20 | 0.17 | 0.17 | 0.18 | 0.20 | 0.18 | 0.18 | 0.20 | 0.17 | 0.15 | 0.15 | 0.16 |
| **Shapiro-Wilk (W)** | W=0.937, p=0.315 | W=0.949, p=0.481 | W=0.949, p=0.477 | W=0.961, p=0.678 | W=0.949, p=0.473 | W=0.905, p=0.096 | W=0.905, p=0.096 | W=0.925, p=0.200 | W=0.928, p=0.224 | W=0.957, p=0.606 | W=0.957, p=0.606 | W=0.967, p=0.792 | W=0.807, p=0.003 | W=0.839, p=0.010 | W=0.840, p=0.010 | W=0.829, p=0.007 |

Tabela 21 – Resumo dos resultados dos testes estatísticos para DTC, BVL e 10-fold cross-validation.

| | **Teste de Wilcoxon para Amostras Pareadas ($W$)** | |
|---|---|---|
| **Métrica** | **DTC x Cross-validation** | **BVA x Cross-validation** |
| **Precision** | $W$ = 5.0, valor-p $\leq$ 0.0018 | $W$ = 7.0, valor-p $\leq$ 0.0006 |
| **Recall** | $W$ = 0.0, valor-p $\leq$ 0.0007 | $W$ = 5.0, valor-p $\leq$ 0.0003 |
| **Accuracy** | $W$ = 0.0, valor-p $\leq$ 0.0007 | $W$ = 5.0, valor-p $\leq$ 0.0003 |
| $\mathbf{F}_1$ | $W$ = 1.0, valor-p $\leq$ 0.0008 | $W$ = 5.0, valor-p $\leq$ 0.0003 |

Tabela 22 – Resumo dos resultados dos testes estatísticos para BVA e DTMT.

| | **Teste de Amostras Pareadas ($t$)** |
|---|---|
| **Métrica** | **BVA x DTMT** |
| **Accuracy** | $t$ = -1.75, p-value = 0.091 |
| **Recall** | $t$ = -1.75, p-value = 0.091 |
| $\mathbf{F}_1$ | $t$ = -1.51, p-value = 0.142 |
| **Precision** | $t$ = -2.13, p-value = 0.041 |

CAPÍTULO

# 6

# CONCLUSÃO

Neste capítulo são apresentadas as principais contribuições desta pesquisa de mestrado. São discutidas as contribuições da dissertação, principais achados, limitações e trabalhos futuros, incluindo os resultados obtidos em termos de contribuição científica e em termos de publicações.

## 6.1 Principais Contribuições

Este trabalho apresenta uma nova proposta de teste de mutação para sistemas de AM modelados por uma árvore de decisão chamada DTMT. A utilização do modelo de árvore de decisão é atrativa pela sua simplicidade e por seu poder de representação. No passado, o teste de mutação foi investigado no contexto de modelos de especificação, mostrando que é interessante explorar este critério de teste para modelos que representam o software em teste (JIA; HARMAN, 2010).

Esta abordagem baseia-se na técnica tradicional de teste de mutação, cujo objetivo é avaliar a qualidade dos casos de teste de um programa, verificando sua capacidade de detectar defeitos. Isto é feito por meio da introdução de mutações, ou seja, versões modificadas do programa original com falhas específicas. Propõe-se uma nova abordagem que aplica teste de mutação à estrutura da árvore de decisão.

Inspirada em um estudo preliminar que investigou a estrutura de árvores de decisão para desenvolver critérios de cobertura (SANTOS *et al.*, 2021), esta dissertação de mestrado também se baseia na exploração desta estrutura para a geração de mutantes. A partir desses mutantes, foram identificadas as instâncias do *dataset* que podem inativá-los. Dessa forma, é criado um conjunto de casos de teste que são mais eficazes do que os casos de teste selecionados aleatoriamente.

Em relação à contribuição para o estado da arte de teste para AM, diferente dos trabalhos relacionados que foram identificados, que exploram o uso de teste de mutação na área de

aprendizagem profunda, a abordagem DTMT centra-se na aplicação de teste de mutação em modelos de classificação.

A abordagem DTMT busca melhorar a robustez e a confiabilidade dos modelos de classificação. Ao aplicar teste de mutação em modelos de classificação, é possível revelar vulnerabilidades para fortalecer e melhorar o desempenho geral dos algoritmos de AM. Os resultados experimentais indicam que o DTMT é uma abordagem promissora para gerar dados de teste para detectar comportamentos incorretos em sistemas de AM.

### 6.1.1 Publicações Científicas

Durante esta pesquisa de mestrado, foram desenvolvidos três artigos científicos publicados em diferentes edições do SAST, nos quais a autora desta dissertação participou como coautora de dois e como autora principal no artigo que é o foco deste trabalho:

- An Experimental Study on Applying Metamorphic Testing in Machine Learning Applications (SANTOS *et al.*, 2020);
- On Using Decision Tree Coverage Criteria for Testing Machine Learning Models (SANTOS *et al.*, 2021);
- Test Data Selection Based on Applying Mutation Testing to Decision Tree Models (SILVEIRA *et al.*, 2023).

Esse último artigo foi premiado como The Best Paper do SAST 2023 e convidado para submissão de versão extendida ao Journal of Software Engineering Research and Development (JSERD). O artigo submetido compara os critérios de cobertura de árvores de decisão e DTMT.

## 6.2 Trabalhos Futuros

Acredita-se que o presente trabalho possa motivar novas proposições de técnicas de teste que explorem a estrutura interna dos modelos de AM. Em pesquisas futuras, é possível realizar uma análise de custos da abordagem DTMT e investigar sua eficácia em testar diferentes modelos de AM, examinando também o motivo pelo qual existem mutantes que permanecem vivos, seja por serem equivalentes à árvore original ou por possuírem caminhos inalcançáveis, e avaliando a viabilidade de criar casos de teste capazes de eliminá-los.

Vale a pena realizar análises qualitativas adicionais, explorar como adaptar a abordagem para selecionar instâncias a fim de criar bases de treinamento de qualidade, avaliar a possibilidade de utilizar mais operadores de mutação e aplicar a abordagem em conjuntos de dados maiores. Além disso, é possível explorar modelos Random Forest (BREIMAN, 2001) em vez de árvores de decisão para melhorar a qualidade dos dados de teste. Florestas aleatórias são uma técnica

que combina múltiplas árvores de decisão para mitigar o overfitting e aumentar a precisão da previsão.

# REFERÊNCIAS

AGRAWAL, H.; DEMILLO, R. A.; HATHAWAY, B.; HSU, W.; HSU, W.; KRAUSER, E. W.; MARTIN, R. J.; MATHUR, A. P.; SPAFFORD, E. H. **Design Of Mutant Operators For The C Programming Language**. [S.l.]: W. Lafayette, IN 47907, Software Engineering Research Center Department of Computer Sciences Purdue University, 1989. Citado nas páginas 31 e 53.

AMMANN, P.; OFFUTT, J. **Introduction to software testing**. [S.l.]: Cambridge University Press, 2016. Citado nas páginas 28, 29, 30 e 31.

Aniche, M.; Maziero, E.; Durelli, R. S.; Durelli, V. H. S. The Effectiveness of Supervised Machine Learning Algorithms in Predicting Software Refactoring. **IEEE Transactions on Software Engineering (Early Access)**, p. 1–15, 2020. Citado na página 23.

BRAIEK, H. B.; KHOMH, F. On Testing Machine Learning Programs. **Journal of Systems and Software**, v. 164, p. 110542, 2020. Citado nas páginas 23 e 45.

BREIMAN, L. Random forests. **Machine Learning**, v. 45, n. 1, p. 5–32, 2001. ISSN 1573-0565. Disponível em: <https://doi.org/10.1023/A:1010933404324>. Citado na página 88.

COPPIN, B. **Inteligência Artificial**. [S.l.]: Grupo Gen - LTC, 2010. Citado nas páginas 34, 37 e 38.

CREMONESI, P.; KOREN, Y.; TURRIN, R. Performance of recommender algorithms on top-n recommendation tasks. In: **Proceedings of the Fourth ACM Conference on Recommender Systems**. New York, NY, USA: Association for Computing Machinery, 2010. (RecSys '10), p. 39–46. ISBN 9781605589060. Disponível em: <https://doi.org/10.1145/1864708.1864721>. Citado na página 40.

DELAMARO, M. E.; MALDONADO, J. C. **Mutação de interface: um critério de adequação interprocedimental para o teste de integração**. Dissertação (Mestrado) — Universidade de São Paulo, 1997. Citado na página 32.

DEMILLO, R.; LIPTON, R.; SAYWARD, F. Hints on test data selection: Help for the practicing programmer. **Computer**, v. 11, n. 4, p. 34–41, 1978. Citado nas páginas 30 e 31.

Durelli, V. H. S.; Durelli, R. S.; Borges, S. S.; Endo, A. T.; Eler, M. M.; Dias, D. R. C.; Guimarães, M. P. Machine Learning Applied to Software Testing: A Systematic Mapping Study. **IEEE Transactions on Reliability**, v. 68, n. 3, p. 1189–1212, 2019. Citado nas páginas 23, 32 e 43.

FILHO, W. D. P. P. **ENGENHARIA DE SOFTWARE FUNDAMENTOS, MÉTODOS E PADRÕES**. [S.l.]: S.I.: LTC, 2009. Citado na página 27.

GÉRON, A. **Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems**. 2nd. ed. [S.l.]: O'Reilly, 2019. 856 p. Citado na página 69.

Hu, Q.; Ma, L.; Xie, X.; Yu, B.; Liu, Y.; Zhao, J. Deepmutation++: A mutation testing framework for deep learning systems. In: **34th IEEE/ACM International Conference on Automated Software Engineering (ASE)**. [S.l.: s.n.], 2019. p. 1157–1161. Citado nas páginas 32, 45 e 46.

HUMBATOVA, N.; JAHANGIROVA, G.; TONELLA, P. Deepcrime: Mutation testing of deep learning systems based on real faults. In: **Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis**. Association for Computing Machinery, 2021. (ISSTA 2021), p. 67–78. Disponível em: <https://doi.org/10.1145/3460319.3464825>. Citado nas páginas 32, 45 e 46.

IEEE Standard Glossary of Software Engineering Terminology. **IEEE Std 610.12-1990**, p. 1–84, 1990. Citado na página 28.

JAHANGIROVA, G.; TONELLA, P. An empirical evaluation of mutation operators for deep learning systems. In: **2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)**. [S.l.: s.n.], 2020. p. 74–84. Citado nas páginas 32, 45 e 46.

JAMES, G.; WITTEN, D.; HASTIE, T.; TIBSHIRANI, R. **An Introduction to Statistical Learning: with Applications in R**. [S.l.]: Springer Texts in Statistics, 2013. 426 p. Citado na página 24.

JIA, Y.; HARMAN, M. An analysis and survey of the development of mutation testing. **IEEE transactions on software engineering**, IEEE, v. 37, n. 5, p. 649–678, 2010. Citado na página 87.

LASZCZYK, M.; MYSZKOWSKI, P. B. Survey of quality measures for multi-objective optimization: Construction of complementary set of multi-objective quality measures. **Swarm and Evolutionary Computation**, v. 48, p. 109–133, 2019. ISSN 2210-6502. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2210650218305340>. Citado na página 40.

LU, Y.; SHAO, K.; SUN, W.; SUN, M. Mtul: Towards mutation testing of unsupervised learning systems. In: DONG, W.; TALPIN, J.-P. (Ed.). **Dependable Software Engineering. Theories, Tools, and Applications**. Cham: Springer Nature Switzerland, 2022. p. 22–40. Citado nas páginas 32, 45 e 47.

MA, L.; ZHANG, F.; SUN, J.; XUE, M.; LI, B.; JUEFEI-XU, F.; XIE, C.; LI, L.; LIU, Y.; ZHAO, J.; WANG, Y. Deepmutation: Mutation testing of deep learning systems. In: **2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)**. [S.l.: s.n.], 2018. p. 100–111. Citado nas páginas 32, 45 e 46.

MALDONADO, J. C.; JINO, M.; DELAMARO, M. E. Conceitos básicos. In: **Introdução ao Teste de Software**. 2. ed. [S.l.]: Elsevier Editora Ltda, 2016. cap. 1. Citado na página 27.

MALDONADO, J. C.; JINO, M.; DELAMARO, M. E.; OLIVEIRA, R. A. P.; BARBOSA, E. F. Teste de mutação. In: **Introdução ao Teste de Software**. 2. ed. [S.l.]: Elsevier Editora Ltda, 2016. cap. 5. Citado na página 30.

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 na página 34.

Marijan, D.; Gotlieb, A.; Kumar Ahuja, M. Challenges of Testing Machine Learning Based Systems. In: **IEEE International Conference On Artificial Intelligence Testing (AITest)**. [S.l.: s.n.], 2019. p. 101–102. Citado na página 23.

MULLER, A.; GUIDO, S. **Introduction to Machine Learning with Python: A Guide for Data Scientists**. O'Reilly Media, Incorporated, 2018. ISBN 9789352134571. Disponível em: <https://books.google.com.br/books?id=jGdXswEACAAJ>. Citado na página 58.

MÜLLER, A. C.; GUIDO, S. **Introduction to Machine Learning with Python: A Guide for Data Scientists**. [S.l.]: O'Reilly Media, 2016. 400 p. Citado nas páginas 60 e 68.

NASER, M. Z.; ALAVI, A. H. Error metrics and performance fitness indicators for artificial intelligence and machine learning in engineering and sciences. **Information and Software Technology**, Naser2021, p. 2730–9894, 2021. Citado na página 40.

PANICHELLA, A.; LIEM, C. C. S. What are we really testing in mutation testing for machine learning? a critical reflection. In: **2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)**. [S.l.: s.n.], 2021. p. 66–70. Citado na página 45.

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.). Elsevier, 2019, (Advances in Computers, v. 112). p. 275–378. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0065245818300305>. Citado na página 24.

PEI, K.; CAO, Y.; YANG, J.; JANA, S. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. **Communications of the ACM**, ACM, v. 62, n. 11, p. 137–145, 2019. Citado nas páginas 23 e 49.

PRESSMAN, R. S. **Software Engineering - A practitioner's approach**. [S.l.]: McGraw-Hill, 1997. Citado na página 30.

RICCIO, V.; HUMBATOVA, N.; JAHANGIROVA, G.; TONELLA, P. Deepmetis: Augmenting a deep learning test set to increase its mutation score. In: **Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering**. [S.l.]: IEEE Press, 2022. (ASE '21), p. 355—-367. Citado nas páginas 32, 45 e 46.

Riccio, V.; Jahangirova, G.; Stocco, A.; Humbatova, N.; Weiss, M.; Tonella, P. Testing Machine Learning Based Systems: A Systematic Mapping. **Empirical Software Engineering**, v. 25, n. 6, p. 5193–5254, 2020. Citado nas páginas 33, 34, 43 e 44.

RODRIGUES, D. S.; DELAMARO, M. E.; CORRêA, C. G.; NUNES, F. L. S. Using genetic algorithms in test data generation: A critical systematic mapping. **ACM Comput. Surv.**, Association for Computing Machinery, v. 51, n. 2, maio 2018. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3182659>. Citado na página 32.

ROKACH, L.; MAIMON, O. **Data mining with decision trees: Theory and applications**. [S.l.]: World Scientific Publishing Co, 2014. Citado nas páginas 34 e 35.

SANTOS, S. a.; SILVEIRA, B.; DURELLI, V.; DURELLI, R.; SOUZA, S.; DELAMARO, M. On using decision tree coverage criteria fortesting machine learning models. In: . [S.l.]: ACM, 2021. (SAST '21), p. 1—-9. Citado nas páginas 24, 48, 49, 87 e 88.

SANTOS, S. a. H. N.; SILVEIRA, B. N. C. da; ANDRADE, S. a. A.; DELAMARO, M.; SOUZA, S. R. S. An experimental study on applying metamorphic testing in machine learning applications. In: **Proceedings of the 5th Brazilian Symposium on Systematic and Automated Software Testing**. New York, NY, USA: Association for Computing Machinery, 2020. (SAST '20), p.

98–106. ISBN 9781450387552. Disponível em: <https://doi.org/10.1145/3425174.3425226>. Citado na página 88.

SCHMIDHUBER, J. Deep learning in neural networks: An overview. **Neural Networks**, v. 61, p. 85–117, 2015. ISSN 0893-6080. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0893608014002135>. Citado na página 34.

SCHMIDT, M. D.; LIPSON, H. Age-fitness pareto optimization. In: **Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation**. New York, NY, USA: Association for Computing Machinery, 2010. (GECCO '10), p. 543–544. ISBN 9781450300728. Disponível em: <https://doi.org/10.1145/1830483.1830584>. Citado na página 40.

SHEN, W.; WAN, J.; CHEN, Z. Munn: Mutation analysis of neural networks. In: **2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)**. [S.l.: s.n.], 2018. p. 108–115. Citado nas páginas 32, 45 e 46.

Sherin, S.; khan, M. U.; Iqbal, M. Z. A systematic mapping study on testing of machine learning programs. arXiv, 2019. Citado nas páginas 34, 43 e 44.

SILVA, K.; COTA, E. Predicting prime path coverage using regression analysis. In: **Proceedings of the XXXIV Brazilian Symposium on Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2020. (SBES '20), p. 263–272. ISBN 9781450387538. Disponível em: <https://doi.org/10.1145/3422392.3422413>. Citado na página 32.

SILVEIRA, B.; DURELLI, V.; SANTOS, S. a.; DURELLI, R.; DELAMARO, M.; SOUZA, S. Test data selection based on applying mutation testing to decision tree models. In: **Proceedings of the 8th Brazilian Symposium on Systematic and Automated Software Testing**. New York, NY, USA: Association for Computing Machinery, 2023. (SAST '23), p. 38–46. ISBN 9798400716294. Disponível em: <https://doi.org/10.1145/3624032.3624038>. Citado na página 88.

STRUG, J.; STRUG, B. Machine learning approach in mutation testing. In: NIELSEN, B.; WEISE, C. (Ed.). **Testing Software and Systems**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 200–214. ISBN 978-3-642-34691-0. Citado na página 47.

____. Evaluation of the prediction-based approach to cost reduction in mutation testing. In: ŚWIĄTEK, J.; BORZEMSKI, L.; WILIMOWSKA, Z. (Ed.). **Information Systems Architecture and Technology: Proceedings of 39th International Conference on Information Systems Architecture and Technology – ISAT 2018**. Cham: Springer International Publishing, 2019. p. 340–350. ISBN 978-3-319-99996-8. Citado na página 47.

TAMBON, F.; KHOMH, F.; ANTONIOL, G. A probabilistic framework for mutation testing in deep neural networks. **Information and Software Technology**, v. 155, p. 107–129, 2023. Citado nas páginas 32, 45 e 47.

VANDERSTOEP, S. W.; JOHNSON, D. D. **Research Methods for Everyday Life: Blending Qualitative and Quantitative Approaches**. [S.l.]: Jossey-Bass, 2008. 352 p. Citado na página 58.

WEYUKER, E. J. On testing non-testable programs. **The Computer Journal**, The British Computer Society, v. 25, n. 4, p. 465–470, 1982. Citado na página 34.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in Software Engineering**. [S.l.]: Springer, 2012. 236 p. Citado nas páginas 58, 66 e 74.

ZHANG, J.; ZHANG, L.; HARMAN, M.; HAO, D.; JIA, Y.; ZHANG, L. Predictive mutation testing. **IEEE Transactions on Software Engineering**, v. 45, n. 9, p. 898–918, 2019. Citado na página 47.

Zhang, J. M.; Harman, M.; Ma, L.; Liu, Y. Machine Learning Testing: Survey, Landscapes and Horizons. **IEEE Transactions on Software Engineering (Early Access)**, p. 1–37, 2020. Citado nas páginas 23, 34, 43 e 45.


ICMC
USP
SÃO CARLOS