Journal of Software Engineering Research and Development, 2025, 13:9, doi: 10.5753/jserd.2025.4084
This work is licensed under a Creative Commons Attribution 4.0 International License..

Advancing Test Data Selection by Leveraging Decision Tree Structures: An Investigation into Decision Tree Coverage and Mutation Analysis

Beatriz N. C. Silveira [**Universidade de São Paulo** | ***beatrizncs@usp.br***]
Vinicius H. S. Durelli [**Universidade Federal de São Carlos** | ***vinicius.durelli@ufscar.br***]
Sebastião H. N. Santos [**Universidade de São Paulo** | ***sebastiaohns@usp.br***]
Rafael S. Durelli [**Universidade Federal de Lavras** | ***rafael.durelli@ufla.br***]
Marcio E. Delamaro [**Universidade de São Paulo** | ***delamaro@icmc.usp.br***]
Simone R. S. Souza [**Universidade de São Paulo** | ***srocio@icmc.usp.br***]

Abstract Over the past decade, there has been a significant surge in interest regarding the application of machine learning (ML) across various tasks. Due to this interest, the adoption of ML-based systems has gone mainstream. It turns out that it is imperative to conduct thorough software testing on these systems to ensure that they behave as expected. However, ML-based systems present unique challenges for software testers who are striving to enhance the quality and reliability of these solutions. To cope with these testing challenges, we propose novel test adequacy criteria centered on decision tree models. Our criteria diverge from the conventional method of manually collecting and labeling data. Instead, our criteria relies on the inherent structure of decision tree models to inform the selection of test inputs. Specifically, we introduce decision tree coverage (DTC) and boundary value analysis (BVA) as approaches to systematically guide the creation of effective test data that exercises key structural elements of a given decision tree model. Additionally, we also propose a mutation-based criterion to support the validation of ML-based systems. Essentially, this approach involves applying mutation analysis to the decision tree structure. The resulting mutated trees are then used as a reference for selecting test data that can effectively identify incorrect classifications in ML models. To evaluate these criteria, we carried out an experiment using 16 datasets. We measured the effectiveness of test inputs in terms of the difference in model's behavior between the test input and the training data. According to the results of the experiment, our criteria can be used to improve the test data selection for ML applications by guiding the generation of diversified test data that negatively impact the prediction performance of models.

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

1 Introduction

In recent years, due to the growing abundance of available data, machine learning (ML) algorithms have emerged as an alternative data-driven approach for solving a myriad of problems (Durelli et al., 2019; Aniche et al., 2022). Given the escalating popularity of ML algorithms, researchers and practitioners alike have been exploring ways to evaluate and improve the reliability and quality of ML-based software systems (Braiek and Khomh, 2020; Zhang et al., 2022).

Before deploying ML models, testers need to approach these models like conventional software and carry out testing efforts to uncover potential issues. This testing phase is an essential part of the training and deployment process. However, despite its importance, uncovering problems in ML-based applications presents significant challenges. Many challenges arise due to the inherent differences between traditional software and ML-based software, which is more statistically-oriented and inherently less deterministic. The behavior of ML-based software is derived from a data-driven process: ML algorithms are used to model and understand complex datasets, and the process of deriving decision logic from data through training can vary greatly depending on the specific ML algorithm employed. To address the challenges associated with testing ML-based systems, researchers have started adopting proven methods and approaches, such as structural coverage criteria and mutation testing (Braiek and Khomh, 2020). Drawing inspiration from white-box testing approaches, researchers have been probing into how the internal structure of ML models can be used to generate test cases (Li et al., 2019; Pei et al., 2019). However, it is worth noting that the lion's share of current testing techniques are not directly applicable to software representations of ML models. For example, traditional white-box testing methods, based on a software's internal elements as branches and conditions do not carry over into the context of testing ML models because behavior in these models is not encoded as control flow structures.

In an effort to cope with the shortcomings of some of the recently introduced approaches to testing ML-based software, we developed two coverage criteria that leverage the internal structure of decision tree models to help testers devise test cases. These tree coverage criteria are sufficiently fine-grained so as to allow for testers to quantify the effectiveness of test suites by counting the amount of leaves/decisions activated by data inputs. Moreover, we devised an approach that capitalizes on mutation testing to create effective test suites for uncovering faults in ML models. Specifically, our approach is centered around mutating some elements of the internal structure of decision tree models, allowing for the selection of inputs (i.e., test cases) that are sensitive enough to reveal discrepancies in the ensuing predictions (i.e., outcomes).

A decision tree algorithm involves segmenting the predictor space into non-overlapping regions. The splitting rules used to segment the predictor space are summarized in a model that resembles a tree structure in which internal nodes represent branching decisions and leaf nodes represent the result of a series of decisions (James et al., 2013). Decision tree algorithms primarily benefit from their simplicity and interpretability. Internal nodes represent feature relationships, while leaf nodes indicate potential outcomes, making the resulting models easily understandable.

Given a decision tree, the rationale behind our two tree coverage criteria is to interpret the resulting model as a tree structure and come up with test cases that cover all leaf nodes. Such rationale is similar to that used to evaluate the adequacy of test data for "conventional" programs, in which an example of adequacy criteria is to cover all branches of the program under test. Accordingly, our decision tree coverage criteria evaluate a test set for a given decision tree in terms of how effectively the test cases reach and cover the decisions represented by the tree's leaf nodes.

In conventional software testing settings, mutation testing entails repeatedly making subtle changes in the form of small syntactic deviations (i.e., *mutations*) to the software being tested. The purpose of this is to examine if any test case fails when executed against the mutated program. A mutated program is referred to as a *mutant* and is said to be *killed* when at least one test case detects the introduced mutation (i.e., fails due to the mutation). When a test case is sensitive enough to kill a mutant, the main implication is that the test suite is effective. Thus, the primary goal of mutation testing is to evaluate the quality of the test suite by fostering the creation of test cases that uncover problems that stem from the mutants (i.e., the small syntactic deviations made to the program under test).

Our decision tree mutation approach focuses on mutating the decision nodes in tree models. We achieve this by applying relational and constant mutation operators to the decision nodes. Subsequently, we examine the dataset for instances where the classification provided by the mutated tree model diverged from that produced by the original decision tree. This process is performed in hopes of coming up with a dataset specifically tailored for killing the mutated tree models. In a manner akin to mutation analysis in traditional testing, new test cases are then generated, thereby enhancing the original dataset. The process, as illustrated in Figure 1, results in a test set that can be used to evaluate other models trained with the original dataset

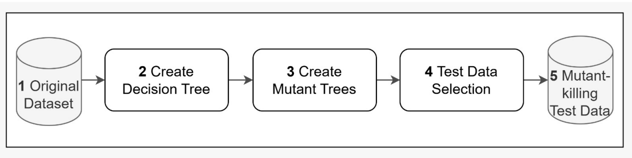

Figure 1. An overview of the proposed mutation-based approach to testing ML-based models.

We conducted an experimental study to evaluate the effectiveness of test data produced by our decision tree based approaches. We compare our approaches with k-fold cross-validation, a common technique for assessing ML models. We explored whether our approaches could improve the choice of a high-quality test dataset. This paper builds upon the criteria and findings presented in two previous studies (Santos et al., 2021; Silveira et al., 2023). The scope of our experiment now includes a two-fold comparison, utilizing the criteria presented in our earlier research. These improvements are aimed at bolstering the robustness and comprehensiveness of our experimental results. The contributions presented in this paper are as follows:

- We propose three decision tree based criteria. Two test adequacy criteria based on different aspects of the internal representation of decision tree models. Additionally, we propose an approach that applies mutation testing to decision tree models. This approach uses relational and constant mutation operators to mimic faults in the internal representation of decision tree models. It is worth mentioning that of the three proposed decision tree-based criteria, the first two were initially presented in our previous study, and thus represent an extension in this expanded version of our article.
- We conduct a comparative analysis between the two coverage criteria and the decision tree mutation criterion, providing detailed insights into the implications of these comparative results.
- We demonstrate how these three criteria may be used to choose test data samples that are more effective than randomly selected test data. Here, effectiveness is defined by how much the test data can decrease prediction scores.

The remainder of this paper is organized as follows. Related work is presented in Section 2. Our decision tree-based criteria are described in Section 3. The experimental study for evaluating our testing criteria is presented in Section 4. Section 5 describes the experimental results, including the statistical analysis of our experimental study. Section 6 presents a discussion of the results of our experiment. Threats to validity are covered in Section 7, and concluding remarks are presented in Section 8.

2 Related Work

Some papers have presented literature reviews on software testing for ML-based programs (Sherin et al., 2019; Riccio et al., 2020; Braiek and Khomh, 2020; Zhang et al., 2022; Ogrizović et al., 2024). Panichella and Liem (2021) raise questions about the extent and rigor with which mutation testing techniques have been applied to Deep Learning (DL)-based programs, arguing that these approaches do not always align with the classical interpretation of mutation testing. Based on these literature reviews, we have selected papers dealing with mutation testing for ML-based programs.

MuNN is a method for applying mutation analysis to neural networks (Shen et al., 2018). It provides five mutation operators designed to build deep-learning models with potential bugs. Given a test set, the mutation score can be calculated as the fitness of the test set. In this context, the results from the mutation analysis can guide test generation and neural network test optimization. The experimental results indicate that mutation analysis when applied to neural networks is strongly influenced by domain-specific characteristics, highlighting the need for domain-specific mutation operators to improve the effectiveness of mutation analysis in the context of DL-based programs. Additionally, according to the results, the depth of a neuron is a sensitive factor in DL mutation analysis.

DeepMutation is also a framework for applying mutation testing techniques to DL systems (Ma et al., 2018). This paper proposes a source-level mutation testing technique for training data and training programs. To this end, a set of source-level mutation operators are designed. A model-level mutation testing technique is also proposed, defining a set of mutation operators that inject faults directly into DL models, and mutation testing metrics are proposed to measure test data quality. A follow-up to this study resulted in the Deepmutation++ framework (Hu et al., 2019): a mutation testing framework for Feed-Forward Neural Networks (FNNs) and Recurrent Neural Networks (RNNs). DeepMutation++ incorporates eight model-level operators for FNN models introduced in DeepMutation (Ma et al., 2018) and introduces nine new specialized operators for RNN models.

The study by Jahangirova and Tonella (2020) also investigates how mutation testing can be used to drive the test data generation for DL-based software. The authors focus on configurations that render mutation operators non-equivalent and non-trivial. Specifically, the authors propose a new definition of killed mutants that accounts for the stochastic nature of DL systems. They compare this approach to the traditional threshold-based drop in accuracy, demonstrating its inconsistency across different runs.

DeepCrime (Humbatova et al., 2021) also proposes mutation operators for DL systems; However, unlike the previously mentioned approaches, its set of mutation operators is grounded in real-world DL faults. Humbatova et al. define 35 DL mutation operators and implement 24 DL mutation operators in DeepCrime. To evaluate the tool, Humbatova et al. compared the sensitivity of the tool to changes in the test data quality with that of DeepMutation++ (Hu et al., 2019). The results show that DeepCrime produces killable and non-trivial mutants, can effectively discriminate a weak test set from a strong one, and significantly outperforms DeepMutation++ in these aspects. In Humbatova et al. (2023), the authors expanded this work, making new evaluations on the artifacts of three large-scale studies focusing on real faults in DL systems. The authors conclude that DeepCrime generates meaningful mutants that are able to effectively distinguish between weak and strong test suites.

Riccio et al. (2022) propose DeepMetis (Riccio et al., 2022), an automated approach to increasing existing test sets with inputs that kill mutants generated by DeepCrime (Humbatova et al., 2021). The goal of this test set augmentation approach is to increase the mutation score of a test set by generating new entries that kill mutants not killed by the original test set. DeepMetis is a search-based test generator for DL systems that uses mutation fitting as a guideline. An experiment was carried out to evaluate the approach, the results would seem to indicate that DeepMetis is effective in augmenting a given test set by adding tests that detect, on average, 63% of the mutants. The experimental evaluation shows that the augmented test suite can expose hard-to-uncover mutants, which simulates the occurrence of undetected faults. DeepMetis differs from existing approaches in that its goal is to increase the ability to kill mutants. With the advent of DL mutation frameworks such as DeepMutation (Ma et al., 2018), MuNN (Shen et al., 2018) and DeepCrime (Humbatova et al., 2021), the problem of achieving a high mutation score is increasingly important, especially when mutants simulate real faults, as is the case with DeepCrime (Humbatova et al., 2021).

Lu et al. (2022) describe a mutation testing technique specifically designed for unsupervised learning (UL) systems. In this study, Lu et al. detail the mutation operators they developed to simulate unstable scenarios and potential errors that UL systems may encounter. The proposed approach incorporates an autoencoder to generate contradictory samples, demonstrating its feasibility across three datasets.

Another paper focused on Deep Learning proposes a Probabilistic Mutation Testing (PMT) approach, which enables more consistent decisions on whether a mutant is killed. This approach was evaluated using three models and eight mutation operators (Tambon et al., 2023). The authors also analyze the trade-off between the approximation error and the cost of the method, showing that a relatively small error can be achieved for a manageable cost. The authors argue that PMT is the first step in this direction that effectively removes the lack of consistency between test runs of previous methods caused by the stochasticity of DNN training.

Exploring a somewhat different topic, the approach FAIRER (Li et al., 2023) addresses fairness in machine learning models. The authors propose aligning model decisions with principles of equity, introducing a metric to assess and interpret classifier impartiality. Practical methods are presented for testing models' fairness, identifying, and correcting potential biases. The approach is validated experimentally, demonstrating how decision alignment can enhance fairness across various classification contexts.

Another research avenue that has been drawing significant attention focuses on enhancing sample selection for training ML models. Rittler and Chaudhuri (2023) propose an active learning approach within the context of the k-NN algorithm (Rittler and Chaudhuri, 2023). The approach splits the process into two stages: an initial selection of samples to maximize learning efficiency, followed by a refinement phase to improve model precision. According to the results, the approach enhances both efficiency and accuracy, enabling the model to learn from fewer labeled data. The approach was validated on multiple datasets, showing superior performance in sample selection. Another effort related to improving sample selection introduces an approach that considers the model's sensitivity ("sharpness") to small data variations (Kim et al., 2023). SAAL selects samples that improve the model's generalization capabilities, making it more robust and efficient. This optimizes data selection for labeling while maintaining high performance on unseen scenarios. Empirical results indicate that SAAL outperforms existing active learning methods in terms of precision and efficiency.

Also in context of DL systems, Ahmed and Makedonski (2024) draw an analogy between the software development process using formal methods and the ML development process. They argue that the two processes are similar when considering the data processing required to generate both the training data and the model. The authors compare the proposed state-of-the-art mutant selection criteria, evaluating the metrics used, the extra computational cost incurred, and the criteria for filtering out trivial, redundant, and equivalent mutants. They conclude that there is a lack of realism in existing model-level mutation operators and emphasize the importance of rethinking the hypotheses of mutation testing for ML.

It is important to note that, although our decision tree-based coverage criteria were originally designed for test data selection, they can be adapted to enhance the quality of training data selection as well. However, this article does not explore the application of our approaches for training data sampling in depth. Table 1 provides an overview of the related work discussed in this section, offering a comparative analysis of the related work and our decision tree-based approaches to testing ML-based programs.

3 Decision Tree based Criteria

In the following two subsections, we describe the three criteria based on decision tree models we propose for enhancing the testing of machine learning models. The first subsection introduces the decision tree coverage criteria. A more in-depth discussion of these criteria and an extensive analysis of their effectiveness in selecting test data is available in our prior work (Santos et al., 2021). Following this, Subsection 3.2 then outlines our third criterion, which revolves around the application of mutation testing to decision nodes.

3.1 Decision Tree Coverage Criteria

Decision trees are learning algorithms that create tree-like graphical models, known for their human interpretability (James et al., 2013). The core concept of our decision tree criteria is to use the tree-like internal structure of these models and the information in decision nodes to guide input selection. We posit that sampling test inputs that increase leaf/decision coverage is likely to yield more effective test cases. Additionally, given the intrinsic interpretability of decision tree models, leveraging their internal structure for guiding test input selection not only improves coverage but also provides testers with clear insights into the extent of how much of the model has been covered by the test inputs, offering an improvement over traditional manual, *ad hoc* testing methods for ML-based software. The criteria we propose are defined as below.

- **Decision Tree Coverage (DTC) Definition:** given a decision tree model M, a test set T is considered DTC-adequate if it includes tests that traverse the tree from its root to all leaf nodes at least once.
- **Boundary Value Analysis (BVA) criterion:** test cases are designed to cover valid boundary values of decision nodes. Specifically, when designing test cases, this criterion requires the selection of values that explore either the lower or the upper boundary of each decision. This criterion is based on the common assumption in software testing that test cases that explore *boundary conditions* are needed for effective testing (Myers et al., 2011).

When applying DTC each root-to-leaf path represents a test requirement. Given that in decision tree models there is only one path from the root to each leaf node, the number of test requirements for DTC is the number of root-to-leaf paths in a given decision tree. It is worth emphasizing that by traversing a decision tree from its root to all leaves BVA places special emphasis on the decision boundaries (i.e., branches of the tree), which stem from combinations of thresholding rules inferred from the dataset to represent the most important features.

When examining the two criteria as a whole, two approaches are used to generated test data: *(i)* for DTC, we design test cases that reach all leaves in the models under test (i.e., test cases that traverse root-to-leaf paths) and *(ii)* for BVA, we include test data with either the lower or upper limit value concerning decision boundaries of each internal node along the path to a given leaf node. From a test requirement perspective, both test case generation approaches are aimed at satisfying the same number of test requirements: all root-to-leaf paths in the tree under test.

As mentioned, these criteria are premised on the notion that the internal structure of decision tree models can help testers select test data. As an additional benefit of exploring the internal structure of models, the testing criteria also allow testers to quantify the effectiveness of test data by analyzing the number of outputs/decisions covered by such test data.

3.2 Decision Tree Mutation Testing

Mutants are created through mutation operators defined to specific programming languages or specification techniques. These operators are designed to mimic the most common mistakes typically made by programmers or to fulfill specific testing objectives (Ammann and Offutt, 2016). Therefore, mutation operators consist of rules that specify the changes to be made in the program under test. Applying small changes to the software under test encourages the tester to produce test cases that reveal the defects inserted in mutant programs, improving the quality of the test case set (DeMillo et al., 1978)

Our Decision Tree Mutation Testing (DTMT) was developed drawing inspiration from mutation operators specifically designed for the C language (Agrawal et al., 1989). Specifically, the following mutation operators are used in our definition:

- **ORRN**: this operator replaces every occurrence of a relational operator ($<$, $>$, $<=$, $>=$ and $==$) by another possible relational operator. For instance, `original`: $a < b$; `mutant`: $a <= b$.
- **Cccr**: this operator replaces every occurrence of a constant with another possible constant. For instance, given

Table 1. Overview of the related work on testing ML- and DL-based software.

Study	Year	Focus	Type of Model Tested	Type of Testing	Experimental Evaluation
Shen et al. (2018)	2018	Mutation testing of neural networks	Neural networks	Mutation testing	Yes
Ma et al. (2018)	2018	Mutation testing of deep learning systems	Deep learning models	Mutation testing	Yes
Hu et al. (2019)	2019	Mutation testing framework for deep learning	Deep learning models	Mutation testing	Yes
Jahangirova and Tonella (2020)	2020	Evaluation of mutation operators in deep learning	Deep learning models	Mutation testing	Yes
Humbatova et al. (2021)	2021	Real-fault-based mutation testing for deep learning	Deep learning models	Mutation testing with real faults	Yes
Santos et al. (2021)	2021	Decision tree coverage criteria for ML model testing	K-NN	Decision tree-based coverage testing	Yes
Riccio et al. (2022)	2022	Augmentation of test sets to improve mutation score	Deep learning models	Mutation-based test set augmentation	Yes
Lu et al. (2022)	2022	Mutation testing of unsupervised learning systems	Unsupervised learning systems	Mutation testing	Yes
Tambon et al. (2023)	2023	Probabilistic framework for mutation testing in DNNs	Deep neural networks	Probabilistic mutation testing	Yes
Li et al. (2023)	2023	Fairness evaluation and alignment in ML models	Machine learning classifiers	Fairness testing and evaluation	Yes
Rittler and Chaudhuri (2023)	2023	Active learning algorithm for k-NN classification	k-Nearest Neighbors	Active learning for improved training	Yes
Kim et al. (2023)	2023	Sharpness-aware active learning for model robustness	Various ML models	Active learning considering model sharpness	Yes
Silveira et al. (2023)	2023	Mutation testing applied to decision tree models	K-NN	Decision tree-based mutation testing	Yes
Ahmed and Makedonski (2024)	2024	Mutation testing of deep learning systems	Deep learning models	Mutation testing	Yes

a set of constants $[7.5, 2.3, 3.14]$; `original` $a = 7.5$; `mutant`: $a = 3.14$.

Based on these mutation operators, DTMT is defined as follows: given a decision tree model, a test set is considered suitable for DTMT if it includes tests that generate a different classification result between the original and all mutant tree models. The mutant tree models are mutations of tree models where the operators ORRN and Cccr are applied to the intermediate/decision nodes of the original tree.

In the context of DTMT, each mutant tree represents a test requirement. To satisfy a given test requirement, it is essential to identify rows in the dataset that yield a different classification between the mutant and the original model.

The number of mutant trees depends on the number of intermediate nodes in the original tree. Thus, when applying the ORRN operator, we have four mutant trees for each node: one for each possible relational operator. For instance, consider the decision tree in Figure 2. This tree was generated from the Iris dataset.[1] In this tree, eight nodes have relational operators; therefore, it is possible to generate 32 mutant decision trees for ORRN operator. Figure 3 shows an example of a mutant decision tree, highlighting the mutated node.

To apply the Cccr operator, each tree node containing a comparison between a feature and a constant value is selected, and the constant value is replaced by another constant

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

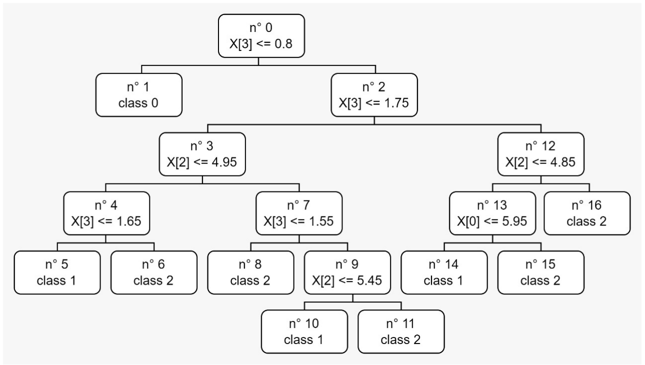

Figure 2. Decision tree model generated from the Iris dataset. In this model, the features are represented as follows: x[0] corresponds to sepal length, x[1] to sepal width, x[2] to petal length, and x[3] to petal width.

value used in another relational operation involving the same feature. The mutation is applied to all decision nodes in our approach; however, we have imposed a limitation of up to two changes per feature to prevent an overwhelming number of mutants. The number of mutant trees can become impractical, especially when the original decision tree has a large number of features.

Figure 4 shows an example of a mutant tree generated by applying the Cccr operator. Considering that the decision tree has eight decision nodes including relational operators and eight different constant values, applying the Cccr operator to generate up to two mutants for each decision node results in

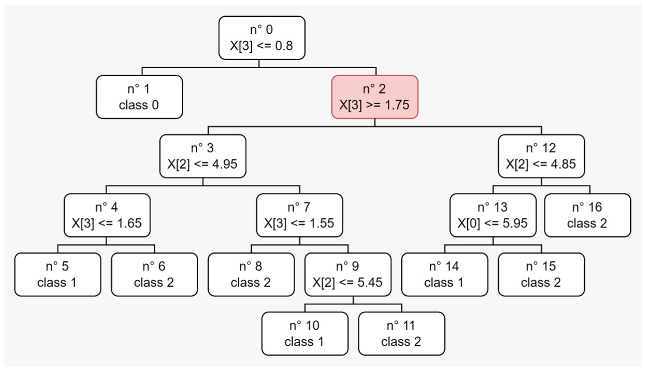

Figure 3. An example of applying the ORRN operator to a tree node.

a total of 10 mutant trees. The number of mutants generated from a specific decision node is determined by the frequency with which the same feature is compared to different constant values in other decision nodes. For example, if a feature is compared with only one constant, no mutant is generated. In cases where a feature is part of two different comparisons, each with a distinct constant, only one mutant is generated for the current decision node, where the constant is swapped with the value used in the other decision node's comparison. Thus, it is only when a feature is compared to three or more unique constants across various decision nodes that we cap the number of mutants to two.

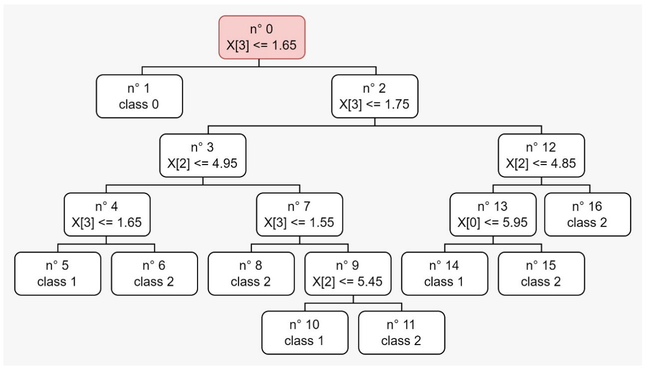

Figure 4. Example of mutant tree generated from applying the Cccr operator to node n^o 0.

Similar to DTC and BVA, DTMT is also used for generating test data. However, in terms of test requirements, it differs from the DTC and BVA approaches, which focus on satisfying the same set of test requirements: covering all paths from the root to the leaf in the tree under test. Conversely, DTMT shifts this focus. Instead of path coverage, DTMT emphasizes creating test inputs that effectively kill mutant trees (i.e., lead to classification results that differ from the original tree).

We apply DTMT to the canonical Iris flower dataset (Fisher, 1936). Applying the decision tree algorithm to the dataset resulted in a tree model with nine leaf nodes and eight decision nodes, as shown in Figure 2. Using DTMT to guide the generation of test cases, we generated a set of 10 test cases. Given that a single test case has the potential to kill multiple mutants, it is not often the case that the quantity of test cases directly corresponds to the number of testing requirements.

To show how a test case can be considered capable of killing a mutant tree, we use the original tree presented in Figure 2, and the mutant tree presented in Figure 3. In this mutant tree, node n^o 2 had its relational expression changed from $X[3]$ $<=$ 1.75 to $X[3] \underline{>=}$ 1.75. Considering a test case in which X[3] = 1 in the original tree, the path would follow to the left child node (n^o 3) and would end at leaf node n^o 5, whereas in the mutant tree, it would be directed to the right child node (n^o 12) and would end at leaf node n^o 16. Thus, achieving different classification outcomes indicates that the mutant under analysis was killed. This implies that the fault mimicked by this change is not accounted for by the decision tree. Therefore, this criterion offers the added benefit of improving the quality of the test suite.

4 Experimental Evaluation

We set out to compare the effectiveness of our three decision tree based criteria through a two-fold experiment. Given that mutation testing is widely regarded as the "gold standard" against which all other types of coverage are measured, it was hypothesized that DTMT would potentially outperform DTC and BVA. Thus, the order in which the criteria were compared is based on this notion of how effective these criteria are at generating test data. Initially, we compared our two decision tree coverage criteria with cross-validation. This is grounded on the conjecture that our decision tree coverage criteria are more suitable to generate test suites for ML-based programs whose ML component was created using the decision tree algorithm, compared to a random methodology. Our hypothesis is based on the fact that our criteria leverage the internal structure and decision information of decision tree models during the identification of test requirements. The assumption is that decision nodes play a key role in determining the model's behavior. Consequently, more effective test data can be derived by capitalizing on the inherent decision-making data of the model under test. Following this initial comparison, the most effective decision tree coverage criterion was then benchmarked against DTMT to further ascertain relative performance. Therefore, we designed the experiment to answer the following research questions (RQs):

RQ$_1$: *How does the effectiveness of the test suites derived from DTC and BVA compare with random test cases from a 10-fold cross-validation?*

RQ$_2$: *How does the effectiveness of the test suites derived from DTMT compare to our best-performing decision tree coverage criterion?*

In the context of this experimental study, effectiveness is quantified using the ML performance metrics mentioned in Subsection 4.2.

4.1 Scoping

The scope of our experiment is defined by setting its goals, which is based on the GQM template (Wohlin et al., 2012) as follows:

Analyze three decision tree based criteria (i.e., DTC, BVA and DTMT)
for the purpose of evaluation

with respect to their effectiveness
from the point of view of the researcher
in the context of testing ML-based programs.

4.2 Hypotheses Formulation

To enable the application of statistical tests, we further refined our RQs into specific hypotheses. We defined our prediction for $\mathbf{RQ}_1$ as: our decision tree coverage criteria (DTCC) are more effective than random testing. Thus, $\mathbf{RQ}_1$ was turned into the following hypotheses:

Null hypothesis, $\mathbf{H}_{0-DTCC\times random}$**:** there is no difference in effectiveness between our decision tree coverage criteria and random testing.

Alternative hypothesis, $\mathbf{H}_{1-DTCC\times random}$**:** there is a significant difference in effectiveness between our decision tree coverage criteria and random testing.

As for $\mathbf{RQ}_2$, we defined our prediction as: our mutation-based criterion is more effective than our best-performing decision tree coverage criterion. Thus, our $\mathbf{RQ}_2$ was turned into the following hypotheses:

Null hypothesis, $\mathbf{H}_{0-DTMT\times DTCC}$**:** there is no difference in effectiveness between the best-performing decision tree coverage criterion and DTMT.

Alternative hypothesis, $\mathbf{H}_{1-DTMT\times DTCC}$**:** there is a significant difference in effectiveness between the best-performing decision tree coverage criterion and DTMT.

The main goal of this study is to explore the effectiveness of DTMT in selecting test data, particularly in comparison to the outcomes provided by our decision tree coverage criteria. To evaluate this assumption, we set out to derive test suites that satisfy our decision tree coverage criteria and DTMT when applied to a decision tree model. In our evaluation, these test suites are then used to assess the performance of models generated from other ML algorithms, such as k-Nearest Neighbors (k-NN). Specifically, we first train models using the k-NN algorithm on the original dataset (i.e., the training data); these models are both trained and tested using 10-fold cross-validation. We then compare the results from this 10-fold cross-validation with the outcomes from applying the generated test suites. Figures 5, 6, 7 and 8 give an overview of the evaluation process.

In the context of traditional software testing, the effectiveness of a criterion is determined by its ability to generate test inputs that can reveal more faults. Hence, a criterion C_1 is deemed more effective than another criterion C_2 if the former leads to test inputs that can uncover more faults than those derived from applying the latter. This concept also applies to our study. For a given model M, if the performance is poorer with test inputs derived from C_1 than those generated from C_2, then we say that C_1 is more effective than C_2. Given a metric, denoted as f, to gauge the quality of model M and a test suite, denoted as T. The score of running test suite T against model M, according to metric f, is expressed as $f(M(T))$. Therefore:

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

indicates that T_1 is more effective than T_2 according to f because T_1 revealed more cases in which M *fails* than T_2 did.

Many approaches have been devised to evaluating and comparing ML models. In our comparative analysis, we decided to take into account several multiple widely used ML performance metrics. As such, we selected precision, recall, accuracy, and F_1 metrics. Thus, the dependent variable that is key in answering our RQs is *effectiveness*: within the framework of this experimental study, effectiveness is quantified using the ML performance metrics mentioned previously.

4.3 Instrumentation

Prior to conducting our experiment, we created experimental objects. These objects are primarily divided into two categories: Python scripts for loading and processing datasets, including test case generation, and scripts for the analysis of results. We employed Google Colaboratory (also known as Colab)[2], a cloud-based environment that facilitates the execution of Python scripts via a browser. Google Colab enables users to create notebooks, which are essentially Jupyter notebooks, combining rich text and executable Python code within a single document.

For data handling and analysis, we utilized the `pandas`[3] library. All the ML algorithms were implemented using `scikit-learn`, a leading Python library for machine learning (Müller and Guido, 2016). we used Google Drive to store Predictive Model Markup Language (PMML)[4] files, which were employed for generating the mutant decision trees.[5].

4.4 Execution

To assess the effectiveness of our mutation-based approach when compared to our two decision tree coverage criteria, we chose a sample of 16 publicly available datasets. These datasets are widely utilized for training machine learning models. We selected these particular datasets due to their simplicity compared to others. Furthermore, these datasets are frequently applied to solve more straightforward problems; consequently, inputting these datasets into a decision tree algorithm yields smaller, more interpretable trees. A summary of the datasets used in our experiment is provided in Table 2.

We loaded all datasets into the Google Colab (i.e, Python) environment. To this end, we used `pandas` library's methods for loading external data. Several datasets had `NA` values, so we performed some data cleansing (also employing the `pandas` library) to create more consistent datasets. After tidying up the data, we split the datasets into two parts: training data (i.e., `X`) and their corresponding outputs or labels (i.e., `y`).

The next step was to build decision tree models from the training data. To fit the models to the training data we used `scikit-learn`, which employs the classification and regression tree (CART) algorithm to train decision trees (Géron,

[2]https://colab.research.google.com/
[3]https://pandas.pydata.org
[4]https://www.ibm.com/docs/pt-br/db2/11.1?topic=analytics-pmml-markup-language-data-mining
[5]Experiment repository: https://drive.google.com/drive/folders/14XpZ3yoFlTW55FVHE6q2-RryqunLYFDJ?usp=sharing

Table 2. Overview of the datasets used in the experiment.

Datasets	Samples	Attributes	Classes	Samples per Class
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]

2019). DTs in `scikit-learn` are implemented in such a way that the resulting models also contain auxiliary information that we used in later steps of the experiment execution. Specifically, these models also keep information regarding all nodes, including the rules in decision nodes that resemble `if-else` code rules from conventional programming languages.

4.4.1 DTC and BVA

The first step in the initial phase of our experiment which involved a comparative analysis of our decision tree coverage criteria against a random methodology, was building decision tree models for the training data. As mentioned, to fit the models to the training data we used `scikit-learn`. Decision trees in `scikit-learn` are implemented in such a way that the resulting models also contain ancillary information, which was instrumental in the later stages of the experiment execution. Specifically, these models also keep information regarding the number of nodes in a given tree, left and right children of each node, the thresholds for all decision nodes, and a list of all leaf nodes.

Following the generation of the models, we devised a recursive approach aimed at identifying distinct paths from the root to every leaf node (Algorithm 1). Given that each leaf node is connected to the root by a unique path, there is a unique representation of each root-to-leaf path. As a result, starting from the root and applying post-order traversal (visiting all the nodes of the right subtree followed by all the nodes of the left subtree), our recursive implementation keeps track of each unique path from root to leaf. Internally, we use arrays to keep path-related information.

When selecting test data that satisfies the DTC criterion, we randomly select from the dataset inputs that cover each unique path from root to leaf. As a result, the algorithm is designed to generate test inputs that effectively cover each leaf of the tree model under examination.

When applying BVA, we adopt a more robust approach to test data selection: the algorithm, while traversing each root-to-leaf path, analyzes decision nodes and generates test inputs that are specifically tailored to be slightly greater or lesser (i.e., at boundary values) than the actual values at the nodes. Consequently, the test inputs for features encountered along the root-to-leaf paths may comprise values that are unlikely to be found in the original dataset. This methodical inclusion of boundary values aims to provide a thorough evaluation of the decision tree's behavior under diverse data conditions.

Algorithm 1 Function getPaths

```
1:  function getPaths(node, is_leaves, right_children, left_children, paths=None, current_path=None)
2:      if $paths$ is None then
3:          Initialize $paths$ as an empty list
4:      end if
5:      if $current\_path$ is None then
6:          Initialize $current\_path$ as an empty list
7:      end if
8:      Add $node$ to $current\_path$
9:      if $is\_leaves[node]$ is true then ▷ If the current node is a leaf
10:         Add $current\_path$ to $paths$
11:     else                                          ▷ Explore right subtree
12:         Call getPaths($right\_children[node]$, $is\_leaves$, $right\_children$, $left\_children$, $paths$, copy of $current\_path$)
                                                      ▷ Explore left subtree
13:         Call getPaths($left\_children[node]$, $is\_leaves$, $right\_children$, $left\_children$, $paths$, copy of $current\_path$)
14:     end if
15:     return $paths$
16: end function
```

The original datasets were employed for the training and testing of k-NN models: training and testing were carried out using *k-fold cross-validation*. Subsequently, the resulting k-NN models were subjected to testing using test suites appropriate for DTC and BVA. The outcomes of these tests were then compared with the results derived from the *k-fold cross-validation* conducted previously.

As mentioned, our evaluation is twofold: *(i)* we evaluate our decision tree coverage criteria in the context of a straightforward train-test split procedure and *(ii)* we also investigate the effectiveness of our criteria within the context of a stratified k-fold cross-validation procedure. In the former, we fit a decision tree model on the whole dataset. The DTC-adequate test data is based on the resulting tree model. More specifically, DTC-adequate test data are produced by extracting from the dataset one observation to traverse each path of the resulting tree model. BVA-adequate test data are computed based on the DTC-adequate test data. We then fit a k-NN model (using $k = 3$) on the remaining observations, which were not used for the DTC-adequate test data generation, and evaluate its performance on the DTC- and BVA-adequate test data. Thus, it is worth emphasizing that the DTC-adequate test data set is a part of the training set that is held out (the k-NN model is not trained on it). Moreover, BVA-adequate test data is computed from this held out data. Figure 5 gives an overview of this procedure. Using the stratified k-fold cross-validation procedure, the training set is split into 10 different subsets (i.e., folds), which are used to build and evaluate the decision tree model 10 times: a different subset is used for evaluation and the decision tree is trained on the other nine folds. In each iteration of the 10-fold cross-validation procedure, the nine training subsets are used to generate DTC- and BVA-adequate test data (as shown in Figure 6). During the evaluation (i.e., testing) step, the k-NN model is tested on the DTC- and BVA-adequate test data as well as the test data from the cross-validation split (as shown in Figure 7). In these two comparisons, we are interested in the results from applying the DTC- and BVA-adequate test suites to the aforementioned k-NN model.

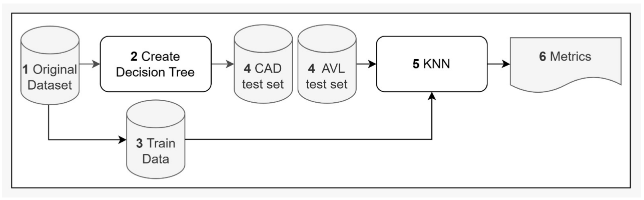

Figure 5. Train-test split steps conducted in the context of the experiment.

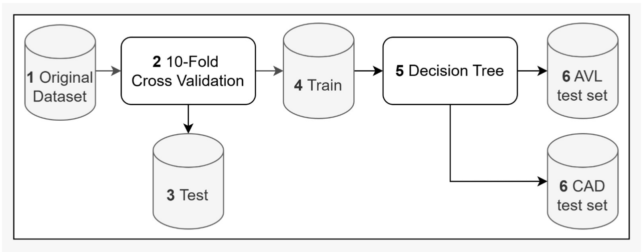

Figure 6. Overview of the test data generation process for the two decision tree coverage criteria.

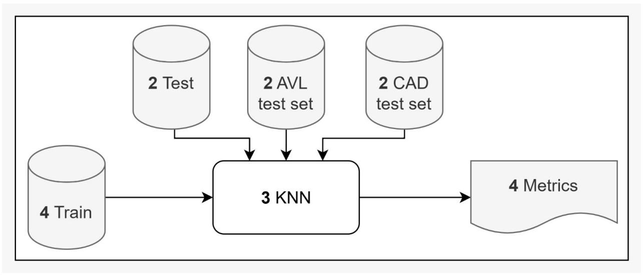

Figure 7. Overview of the test suite evaluation process.

4.4.2 DTMT

Figure 8 highlights the main steps entailed in the application of the DTMT. Essentially, applying DTMT is a two-pronged process. Initially, the process involves the construction of decision tree models, which is succeeded by the generation of mutant trees. Subsequently, test inputs capable of killing these mutant trees are gleaned from the dataset. The latter phase encompasses the training and evaluation of a kNN model, utilizing the mutant-killing test data extracted from the dataset in the initial phase. The process culminates with an analytical comparison of the results obtained from the 10-fold cross-validation using both the conventional dataset and the curated mutant-killing test suite developed throughout the process.

After creating decision tree models by fitting the complete datasets to the decision tree algorithm, the resulting models are rendered into XML-based format files, specifically predictive model markup language (PMML). These PMML

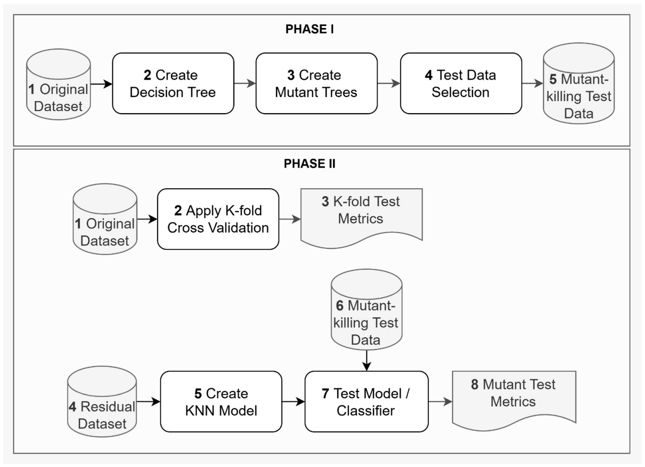

Figure 8. Experiment flow illustration.

files are then used during mutant generation. Given the XML-based structure of PMML files, the parsing of each PMML model starts at the root and decision nodes are mutated as the tree is traversed. To extract model information from the resulting XML Schema in PMML files, we used BeautifulSoup, a Python library for parsing XML and HTML files.

After the generation of mutant trees, our method involves the random selection of instances (i.e., rows) from the dataset. This is followed by a comparison between the predictions made by the original tree and those of each mutant tree. More precisely, for each mutant tree, our approach entails a random traversal through the dataset to identify an instance that effectively kills the mutant under analysis.

This focus on model predictions (i.e., outcomes) to compare DTs emphasizes the selection of instances that contain values that cause mutant trees to make predictions that differ from the predictions made by the original tree model: mutants that result in different predictions are deemed *dead*, while those that generate predictions matching those of the original model are classified as *alive*.

Our mutant-based criterion then carries out an exhaustive examination of the dataset to ascertain if there exists at least one instance capable of killing the live mutant tree being analyzed. Alternatively, it determines if the dataset is devoid of any instance that can achieve this. In scenarios where an instance is found that can kill the mutant, it is selected. Conversely, if no such instance is present, the mutant tree is considered equivalent to the original tree. Through this process of identifying and selecting mutant-killing instances, while simultaneously eliminating duplicates, we derive a subset of the dataset. This subset, composed exclusively of mutant-killing instances, is presumed to be a high-quality test dataset. It serves as an ideal foundation for the validation of other ML models.

We initially used the original datasets to train and test k-NN models, employing a 10-fold cross-validation process. Subsequently, we generated new k-NN models that were trained on data from the original datasets, that is, excluding the data selected for the test set generated by mutation (i.e., a subset of the dataset that does not contain instances that kill mutants). The outcomes of these tests were then compared with the results from the previously run 10-fold cross-validation.

5 Experimental Results

In this section, we examine the effectiveness of our criteria by looking at the performance of models when they are cross-validated versus when tested with test suites tailored for each criterion. Then, we compared the result of the best-performing decision tree coverage criterion with the results of applying DTMT.

5.1 Decision Tree Coverage Criteria: Objects Characteristics

Table 3 gives an overview of the basic properties of the decision tree models generated from the datasets. As shown in Table 3, the resulting model with the most nodes was generated from the `Cancer Prediction` dataset: 2,161 nodes out of which 1,181 are leaf nodes. `Iris` is the dataset that led to the creation of the simplest model, which is comprised of 17 nodes out of which nine are leaves.

Given the outliers in our data regarding the height of the resulting tree models (i.e., amount of nodes), we consider the median values in Table 3 to be a more accurate measure of central tendency than the mean. Thus, on average (median), the models in our experiment have 49 nodes, of which 25 are leaves.

Table 3. Number of nodes in the resulting decision tree models and the number of test cases generated from applying our criteria to these models.

Datasets	Nodes	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
	Descriptive Statistics	
Max	2,161	1,181
Min	17	9
Mean	278.75	139.88
Median	49.00	25.00
Std Dev	562.52	281.26

Each root-to-leaf path within a decision tree model constitutes a testing criterion. Consequently, the volume of test cases generated by our decision tree coverage criteria is directly proportional to the quantity of leaf nodes present in a given model. Thus, the application of our criteria to the ensuing models has, on average (median), generated approximately 25 test requirements, as shown in Table 3.

5.2 DTMT: Objects Characteristics

Our experiment took into account 12,333 mutant trees generated from the 16 chosen datasets. Table 4 shows an overview of the amount of mutants generated for each dataset. The table presents, for each dataset, the amount of non-terminal DT nodes, the number of relational mutants created (ORRN Mut), the number of constant mutants generated (Cccr Mut), and the total number of mutants generated (Total Mut). It also includes the quantity of test data used to kill the relational (ORRN Test Data) and constant mutants (Cccr Test Data). Additionally, Table 4 lists the total volume of test data, which is the combination of "ORRN Test Data" and "Cccr Test Data" after removing duplicates. Furthermore, it provides the proportion of the resulting test set concerning the number of instances in the original dataset (represented as "Test Data/Dataset") and, finally, the amount of mutants that have not been killed ("Live Mut") and the Score Mutation ("Score Mut"). As shown in Table 4, most mutants were generated from the decision tree model of the Cancer Prediction dataset (totaling 5,508 mutants). The Iris dataset led to the simplest tree model, which resulted in 42 mutants.

Given the outliers in our data regarding the number of mutants and test data generated, we consider the median values to be a more accurate measure of central tendency than the mean. Thus, on average (median), the models in our experiment have 125 mutants.Regarding the size of our test set, we have an average of 34 test data for each dataset, which corresponds to 7.21% of the original dataset. As for the number of mutants that were not killed, we have an average of 27.5, and this implies that using our test dataset, we achieved a median mutation score of 79.42%.

As each mutant represents a test requirement, the amount of test cases generated by our criteria is proportional to the number of decision nodes in a given model. Therefore, applying our criteria to the resulting models, on average (median), approximately 125 test requirements.

5.2.1 Analysis of Live Mutants

In our analysis of live mutants, we observed that mutants generated by the Relational Operator Replacement (ORRN) Mutator, specifically those where a relational operator was replaced with <, remained live across all models. A closer examination of the decision tree algorithms reveals that altering the relational operator from <= to < typically results in an equivalent mutant, as this modification does not impact the data split or the final classification that occurs at decision nodes.

This behavior is particularly evident in decision tree algorithms like CART and C4.5, where cut-off points for numerical attributes are determined by intermediate values (such as medians or means) calculated to maximize child node purity. Consequently, when a decision node compares a feature using either <= or < with a specific value, both operators generally produce identical effects on data separation. This is because, in practice, no data point exactly matches the cut-off value, leading to the same split for both conditions. This applies to both continuous data, where the distinction between <= and < does not alter the partition, and discrete data, where

Table 4. Overview of data generated in the experiment.

Datasets	Decision Nodes	ORRN Mut	Cccr Mut	Total Mut	ORRN Test Data	Cccr Test Data	Total Test Data	Test Data/Dataset (%)	Live Mut	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
	Descriptive Statistics									
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
Mean	141.94	567.75	203.06	770.81	120.13	101.94	150.88	10.83	151.56	78.68
Median	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

the split remains unchanged for either condition (Breiman et al., 1983). The Cccr mutation operator did not produce equivalent mutants.

5.3 Hypothesis Testing

Consistent with the methodology employed in our previous experiments (Santos et al., 2021; Silveira et al., 2023), to investigate which decision tree-based criterion leads to the selection of test data that are more effective in evaluating the performance of ML models, we used parametric tests (i.e., two-tailed unpaired two-sample t-test) and non-parametric tests to assess differences in the mean value of the metrics used in our experiment (i.e., precision, recall, accuracy, and F_1).

We checked for normality using the Shapiro-Wilk test before running the tests. According to the results of the Shapiro-Wilk test, all distributions of the results of applying the DTC, BVA and DTMT test data are normal. Nevertheless, the results of the Shapiro-Wilk test also show that all distributions of the results of the 10-fold cross-validation approach deviate from normality (as shown in Table 5). Consequently, we resorted to employing a non-parametric test, specifically the Wilcoxon signed-rank test, for the purpose of comparing this approach with the other criteria.

The statistical analysis summarized in Table 6 presents a comparison of the performance metrics between DTC, BVA, and 10-fold cross-validation using the Wilcoxon signed-rank test.

The results for precision indicate statistically significant differences when comparing both DTC and BVA against 10-fold cross-validation, with W values of 5.0 ($p \leq 0.0018$) and 7.0 ($p \leq 0.0006$), respectively. These low p-values suggest a strong significance in the differences of precision scores between the criteria when compared to 10-fold cross validation.

For recall, the analysis also seems to suggest that there is a difference, especially between DTC and 10-fold cross-validation ($W = 0.0$, $p \leq 0.0007$), and also between BVA and 10-fold cross-validation ($W = 5.0$, $p \leq 0.0003$).

Similar to recall, the accuracy metric shows a significant difference in performance between DTC ($W = 0.0$, $p \leq 0.0007$) and BVA ($W = 5.0$, $p \leq 0.0003$) compared to 10-fold cross-validation. The F_1 scores also reflect a significant disparity, with DTC exhibiting a W value of 1.0 ($p \leq 0.0008$) and BVA a W value of 5.0 ($p \leq 0.0003$) when compared against 10-fold cross-validation. These results further corroborate the significant differences observed in the other metrics.

Overall, the statistical tests reveal that both DTC and BVA exhibit significant differences in performance metrics when compared to 10-fold cross-validation. The results indicate a consistent trend of significant differences across these approaches, suggesting that DTC and BVA may offer distinct advantages over traditional 10-fold cross-validation in terms of model evaluation.

The null hypothesis, $H_{0-DTCC \times random}$, posits no difference in effectiveness between our decision tree coverage criteria and random testing. However, the p-values across all metrics for both DTC and BVA against cross-validation strongly suggest rejecting this null hypothesis. The alternative hypothesis, $H_{1-DTCC \times random}$, which asserts that there is a significant difference in effectiveness is borne out by the results of the comparisons we carried out. The results for precision, recall, accuracy, and F_1 score indicate that both DTC and BVA are significantly different (and likely more effective) than random testing in evaluating the performance of ML models.

The statistical test outcomes presented in Table 7, comparing DTMT with BVA, provide insightful data regarding the effectiveness of these two decision tree based criteria.

Firstly, the paired samples t-test results for accuracy, recall, and F_1 scores indicate that there are no statistically significant differences in terms of these metrics between DTMT and BVA. This suggests that, in terms of accuracy, recall, and F_1 scores, DTMT and BVA perform comparably. However, the precision metric shows a different trend. With a p-value of 0.041, there is a statistically significant difference between DTMT and BVA. This suggests that, in terms of precision, DTMT and BVA do not perform equally, and according to the results one criterion is more effective than the other.

In the context of your research question and hypotheses,

Table 5. Results in terms of the evaluated metrics for the DTC, BVA, DTMT approach and 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
	Descriptive Statistics and Tests of Normality															
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
Mean	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
Median	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

Table 6. Summary of the results from the statistical tests DTC, BVL and 10-fold cross-validation.

	Wilcoxon Signed-Rank Test (W)	
Metric	**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
F_1	W = 1.0, valor-p $\leq$ 0.0008	W = 5.0, valor-p $\leq$ 0.0003

these results offer nuanced insights. The null hypothesis, $H_{0-DTMT \times DTCC}$, posits no difference in effectiveness between the best-performing decision tree coverage criterion (i.e., BVA) and DTMT. Considering that BVA is the best-performing decision tree coverage criterion according to the results of the first comparison, the results for accuracy, recall, and F_1 do not provide enough evidence to reject the null hypothesis – indicating that DTMT does not significantly outperform BVA. However, the results for precision do allow for the rejection of the null hypothesis, suggesting that there is a significant difference in effectiveness between DTMT and BVA in terms of precision. This aligns with the alternative hypothesis, $H_{1-DTMT \times DTCC}$, indicating that there is a significant difference between DTMT and BVA. It is worth noting that these results imply that BVA outperforms DTMT in precision; a lower performance of the resulting model using the generated test data is indicative of a more effective testing approach.

Overall, the results present a nuanced contradiction to the prediction that DTMT would surpass the best-performing decision tree coverage criterion in effectiveness. Although DTMT does not seem to perform as well as BVA in terms of precision, it demonstrates comparable performance in accuracy, recall, and F_1 scores. This pattern suggests that DTMT's effectiveness, when compared to other decision tree coverage criteria, may vary depending on the specific metric under consideration.

Table 7. Statistical test outcomes comparing BVA and DTMT.

	Paired Samples Test (t)
Metric	**BVA x DTMT**
Accuracy	t = -1.75, p-value = 0.091
Recall	t = -1.75, p-value = 0.091
F_1	t = -1.51, p-value = 0.142
Precision	t = -2.13, p-value = 0.041

6 Discussion

To address our **RQ**$_1$, we looked into the effectiveness of our decision tree coverage criteria at selecting test inputs capable of negatively impacting the performance of the model under evaluation. Our hypothesis was grounded in the potential to select stronger test data using our criteria since this test data selection is based on the internal structure of decision tree models, as opposed to random test data selection.

In this context, we measure how *effective* the test input is for an ML model against the data the model was trained on: as mentioned, effectiveness was assessed by the impact of test data on the predictive performance of the model under evaluation. The more detrimental the test data was to the model's performance, the more effective it was considered. In the context of **RQ**$_1$, random testing is a 10-fold cross-validation, a technique that involves dividing the dataset into 10 parts and repeating the model training and testing 10 times, resulting in a general performance estimate.

The results from the comparison between decision tree coverage criteria (DTC and BVA) and 10-fold cross-validation reveal significant differences in performance metrics. Notably, both DTC and BVA demonstrate superior precision, recall, accuracy, and F1 scores compared to 10-fold cross-validation. This suggests that decision tree coverage criteria, particularly DTC and BVA, may offer more robust mechanisms for evaluating ML models than traditional cross-validation methods. Furthermore, our results appear to indicate that BVA outperforms DTC in selecting test inputs capable of negatively impacting the performance of the evaluated model.

Upon analyzing the outcomes of the first part of our experiment, it was established that BVA represents the best-performing decision tree coverage criterion. Consequently, to answer our **RQ**$_2$, we proceeded to conduct a comparative analysis between BVA and DTMT. The experimental results of comparing DTMT with BVA, particularly in terms of precision, offer a multifaceted view of the effectiveness of a decision tree coverage criterion and a mutation-based criterion. While DTMT and BVA exhibit comparable performance in accuracy, recall, and F_1 scores, a significant difference is observed in precision. This suggests that BVA may have an edge over DTMT in precision-focused scenarios. This finding partially contradicts the notion that DTMT could be more effective than the best-performing decision tree coverage criterion.

Figure 9 provides an overview of the effectiveness of the four approaches. According to the results shown in Figure 9, 10-fold cross-validation showed the worst performance in selecting stronger test cases, whereas DTMT and DTC achieved similar performances. On the other hand, BVA stood out as the most effective at identifying stronger test cases.

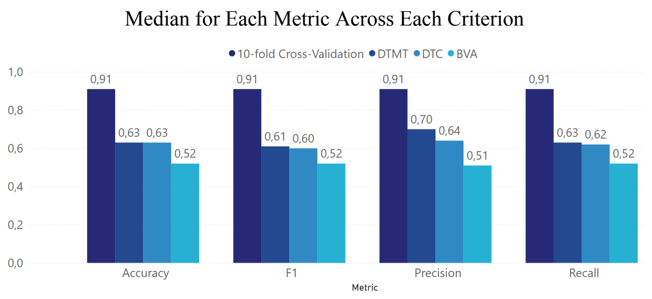

Figure 9. Overview of the effectiveness of the four approaches.

6.1 Comparative Analysis: From Initial Findings to Current Results

Table 8 provides a comparative summary of the results from our early research and results from the current study. A detailed analysis reveals that the metrics of maximum and minimum precision, recall, and accuracy maintain similar patterns across the different approaches. The low standard deviation observed in all approaches indicates minimal variation in the results, reflecting the stability of the evaluated methods.

It is important to mention that our use of random selection was not intended as a comparison with sequential selection but rather as a design choice aimed at introducing diversity in the initial selection process. Therefore, we understand that random selection is not strictly necessary, given that we subsequently perform an exhaustive search to identify an instance (i.e., dataset entry) capable of killing the mutant. We hypothesize this approach facilitates a broader distribution of entries at the outset.

The results suggest that the approaches proposed in our early research show consistency, and the overall performance of the approaches is stable.

7 Threats to Validity

We took several precautions to alleviate potential threats to the validity of our experiment and the findings thereof. However, as is typical with most experimental studies, it is impossible to remedy all threats to validity completely. A potential threat to validity is that we employed the same metrics used in our previous research. As a result, all threats associated with these elements also carry over into our study.

While we have enhanced our sample by incorporating four new datasets, there remains a potential threat to external validity. This is due to the possibility that our dataset sample might not sufficiently represent the target population. Our chosen datasets are smaller and cleaner than those commonly encountered in practice. Therefore, we cannot dismiss the possibility that the results could have differed if larger datasets had been selected. The measures used in the experiment may be considered a potential threat to construct validity as they may not be adequate to assess the effects we set out to investigate. Specifically, precision, recall, accuracy, and F_1 score may not be key predictors of test data suitability for ML-based programs. However, it is worth mentioning that these four measures are widely used to evaluate ML models, which mitigates the risk of this threat.

A fundamental limitation of our criteria is that they assume that only numerical values appear at decision nodes. In addition, regarding the implementation of the decision tree coverage criteria, the exhaustive search in the original dataset is currently done sequentially, which impacts performance when creating our set of test cases.

8 Concluding Remarks

ML-based systems have gained popularity due to their success in various domains. Despite their widespread adoption, ML classifiers are not without issues, which can lead to significant practical consequences. This brings to the limelight the need for approaches tailored to evaluating and improving the quality of ML-based systems. Since this is a relatively new research area, testing ML-based systems remains a substantial challenge. In response to this challenge, adapting criteria that have been effective in testing traditional software systems to the context of ML-based systems has emerged as a promising strategy. To this end, in previous work we propose novel test adequacy criteria for ML-based systems. Two of

Table 8. Comparison of results from early research and the current study.

Approach	Metric	Prior Research				Current Research			
		Precision	Recall	Accuracy	F_1	Precision	Recall	Accuracy	F_1
	Max	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	0.21	0.33	0.33	0.28
	Mean	0.61	0.57	0.57	0.54	0.63	0.61	0.61	0.57
DTC	Median	0.63	0.59	0.59	0.53	0.64	0.62	0.63	0.60
	Std Dev	0.19	0.17	0.17	0.19	0.17	0.17	0.17	0.18
	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=0.937, p=0.315	W=0.949, p=0.481	W=0.949, p=0.477	W=0.961, p=0.678
	Max	0.69	0.67	0.67	0.67	0.99	0.97	0.97	0.98
	Min	0.15	0.39	0.39	0.22	0.15	0.14	0.14	0.22
	Mean	0.48	0.53	0.53	0.49	0.51	0.54	0.54	0.52
BVA	Median	0.52	0.53	0.53	0.51	0.51	0.52	0.52	0.52
	Std Dev	0.17	0.09	0.09	0.14	0.20	0.17	0.17	0.18
	Shapiro-Wilk (W)	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=0.949, p=0.473	W=0.905, p=0.096	W=0.905, p=0.096	W=0.925, p=0.200
	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	0.13	0.29	0.29	0.17
	Mean	0.63	0.61	0.61	0.57	0.66	0.65	0.65	0.62
DTMT	Median	0.70	0.62	0.62	0.58	0.70	0.63	0.63	0.61
	Std Dev	0.22	0.17	0.17	0.20	0.20	0.18	0.18	0.20
	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=0.928, p=0.224	W=0.957, p=0.606	W=0.957, p=0.606	W=0.967, p=0.792
	Max	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Min	0.35	0.48	0.48	0.40	0.35	0.48	0.48	0.40
	Mean	0.82	0.83	0.83	0.82	0.84	0.85	0.85	0.84
10-fold Cross-Validation	Median	0.88	0.88	0.88	0.87	0.91	0.91	0.91	0.91
	Std Dev	0.19	0.16	0.16	0.18	0.17	0.15	0.15	0.16
	Shapiro-Wilk (W)	W=0.843, p=0.030	W=0.885, p=0.102	W=0.885, p=0.102	W=0.871, p=0.068	W=0.807, p=0.003	W=0.839, p=0.010	W=0.840, p=0.010	W=0.829, p=0.007

these criteria draw upon the traditional notion that the underlying structure of models can be explored in such a way as to help testers sample test data that increases structural coverage. Specifically, DTC and BVA are premised on the notion that the internal structure of decision tree models can be utilized to sample test data that is more effective than randomly selected training data. We also propose a mutation-based approach, grounded on the premise that it is possible to utilize the internal structure of decision tree models. This approach allows for the creation of mutant trees, which can then guide the selection of test data more effectively compared to the use of randomly selected test data.

We believe that our research adds to the relatively limited literature on how researchers and practitioners can leverage the knowledge of the internal structure of ML models to devise test data.

To probe into the effectiveness of these criteria and also gauge how they compare with each other, we designed and conducted a two-part experiment. In the first part, we evaluated our two decision tree coverage criteria against random testing (specifically, 10-fold cross-validation) to identify the best-performing decision tree coverage criterion. In the second part, we explored how this decision tree coverage criterion fares in comparison to DTMT. It turns out that the effectiveness of DTC, BVA, and DTMT at generating test data that is able to negatively impact ML models is borne out by the results of our experiment. **Our findings indicate that BVA stands out as the best-performing criterion among the three.** We surmise that BVA's superior performance compared to the other two criteria can be ascribed to its approach of generating new test inputs: currently, BVA achieves this by randomly selecting values that exceed the upper limit or fall below the lower limit of conditions in decision trees. In contrast, applying DTC and DTMT primarily involves selecting existing input values from the dataset. In a follow-up experiment, we plan to look into the implications of creating new test data when applying BVA.

Acknowledgements

Beatriz N. C. Silveira was funded by CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), process number 133469/2020−4. Sebastião H. N. Santos was funded by CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior). The authors also thank the support of FAPESP, process number 2019/06937−0, and CNPq, under processes number 308636/2021−0 and 308445/2021−0.

References

Agrawal, H., Demillo, R. A., Hathaway, B., Hsu, W., Hsu, W., Krauser, E. W., Martin, R. J., Mathur, A. P., and Spafford, E. H. (1989). *Design Of Mutant Operators For The C Programming Language*. W. Lafayette, IN 47907, Software Engineering Research Center Department of Computer Sciences Purdue University.

Ahmed, Z. and Makedonski, P. (2024). Exploring the fundamentals of mutations in deep neural networks. In *Proceedings of the ACM/IEEE 27th International Conference* *on Model Driven Engineering Languages and Systems*, MODELS Companion '24, page 227–233, New York, NY, USA. Association for Computing Machinery.

Ammann, P. and Offutt, J. (2016). *Introduction to software testing*. Cambridge University Press, USA, 2nd edition.

Aniche, M., Maziero, E., Durelli, R., and Durelli, V. H. S. (2022). The effectiveness of supervised machine learning algorithms in predicting software refactoring. *IEEE Transactions on Software Engineering*, 48(4):1432–1450.

Braiek, H. B. and Khomh, F. (2020). On testing machine learning programs. *Journal of Systems and Software*, 164:110 – 542.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1983). *Classification and Regression Trees*. The Wadsworth Statistics/Probability Series. Wadsworth International Group, Belmont, CA, 1st edition.

DeMillo, R., Lipton, R., and Sayward, F. (1978). Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41.

Durelli, V. H. S., Durelli, R. S., Borges, S. S., Endo, A. T., Eler, M. M., Dias, D. R. C., and Guimarães, M. P. (2019). Machine learning applied to software testing: A systematic mapping study. *IEEE Transactions on Software Engineering*, 68(3):1189–1212.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188.

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly, 2nd edition.

Hu, Q., Ma, L., Xie, X., Yu, B., Liu, Y., and Zhao, J. (2019). Deepmutation++: A mutation testing framework for deep learning systems. In *34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1157–1161.

Humbatova, N., Jahangirova, G., and Tonella, P. (2021). 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*, ISSTA 2021, page 67–78, New York, NY, USA. Association for Computing Machinery.

Humbatova, N., Jahangirova, G., and Tonella, P. (2023). Deepcrime: from real faults to mutation testing tool for deep learning. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 68–72.

Jahangirova, G. and Tonella, P. (2020). An empirical evaluation of mutation operators for deep learning systems. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pages 74–84.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics.

Kim, Y.-Y., Cho, Y., Jang, J., Na, B., Kim, Y., Song, K., Kang, W., and Moon, I.-C. (2023). Saal: sharpness-aware active learning. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Li, T., Guo, Q., Liu, A., Du, M., Li, Z., and Liu, Y. (2023). Fairer: fairness as decision rationale alignment. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Li, Z., Ma, X., Xu, C., and Cao, C. (2019). Structural coverage criteria for neural networks could be misleading. In *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pages 89–92. IEEE.

Lu, Y., Shao, K., Sun, W., and Sun, M. (2022). Mtul: Towards mutation testing of unsupervised learning systems. In *Dependable Software Engineering. Theories, Tools, and Applications: 8th International Symposium, SETTA 2022, Beijing, China, October 27-29, 2022, Proceedings*, page 22–40, Berlin, Heidelberg. Springer-Verlag.

Ma, L., Zhang, F., Sun, J., Xue, M., Li, B., Juefei-Xu, F., Xie, C., Li, L., Liu, Y., Zhao, J., and Wang, Y. (2018). Deepmutation: Mutation testing of deep learning systems. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, pages 100–111.

Müller, A. C. and Guido, S. (2016). *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media.

Myers, G. J., Sandler, C., and Badgett, T. (2011). *The Art of Software Testing*. Wiley, 3rd edition.

Ogrizović, M., Drašković, D., and Bojić, D. (2024). Quality assurance strategies for machine learning applications in big data analytics: an overview. *Journal of Big Data*, 11(156):1–48.

Panichella, A. and Liem, C. C. S. (2021). 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)*, pages 66–70.

Pei, K., Cao, Y., Yang, J., and Jana, S. (2019). Deepxplore: Automated whitebox testing of deep learning systems. *Communications of the ACM*, 62(11):137–145.

Riccio, V., Humbatova, N., Jahangirova, G., and Tonella, P. (2022). 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*, ASE '21, pages 355–367. IEEE Press.

Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., and Tonella, P. (2020). Testing machine learning based systems: a systematic mapping. *Empirical Software Engineering*, 25:1573–7616.

Rittler, N. and Chaudhuri, K. (2023). A two-stage active learning algorithm for k-nearest neighbors. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Santos, S., Silveira, B., Durelli, V. H. S., Durelli, R., Souza, S., and Delamaro, M. (2021). On using decision tree coverage criteria fortesting machine learning models. In *Proceedings of the 6th Brazilian Symposium on Systematic and Automated Software Testing*, SAST '21, page 1—9. ACM.

Shen, W., Wan, J., and Chen, Z. (2018). Munn: Mutation analysis of neural networks. In *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 108–115.

Sherin, S., khan, M. U., and Iqbal, M. Z. (2019). A systematic mapping study on testing of machine learning programs.

Silveira, B., Durelli, V. H. S., Santos, S., Durelli, R., Delamaro, M., and Souza, S. (2023). 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*, SAST '23', pages 38–46. ACM.

Tambon, F., Khomh, F., and Antoniol, G. (2023). A probabilistic framework for mutation testing in deep neural networks. *Inf. Softw. Technol.*, 155(C).

Wohlin, C., Runeson, P., Hst, M., Ohlsson, M. C., Regnell, B., and Wessln, A. (2012). *Experimentation in Software Engineering*. Springer.

Zhang, J. M., Harman, M., Ma, L., and Liu, Y. (2022). Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, 48(1):1–36.