Threshold Mutation Operators for Test Data Selection Applied to Machine Learning Testing

Waisman F. Braga
ICMC/USP
São Carlos, Brazil
waisman.braga@usp.br

Marcio E. Delamaro
ICMC/USP
São Carlos, Brazil
delamaro@icmc.usp.br

Simone R. S. Souza
ICMC/USP
São Carlos, Brazil
srocio@icmc.usp.br

Abstract

Machine learning (ML) models are commonly evaluated by sampling-based methods that estimate generalization performance but may fail to uncover fault-exposing inputs. Mutation-based test selection addresses that limitation by introducing controlled perturbations and selecting instances that distinguish the original model from mutated variants. Building on prior decision tree mutation testing (DTMT), this manuscript proposes two threshold-specific mutation operators for decision-tree-based test data selection, namely, local boundary threshold (LBT) and impurity-aware threshold (IAT). Both operators mutate a single internal-node threshold while preserving the original tree topology towards reducing mutant-generation cost and selecting challenging test inputs. Experimental results indicate that LBT and IAT generated fewer mutants than DTMT, selected more challenging test sets than standard sampling references, produced lower equivalent-mutant rates, and recovered around one-third of DTMT-selected instances. Such results suggest that LBT and IAT can serve as complementary, lower-cost test-selection strategies that expand the set of challenging instances identified by mutation-guided selection.

Keywords

Software Testing, Machine Learning, Decision Tree, Mutation Testing

1 Introduction

Over the past decade, machine learning (ML) has shifted from an experimental research tool to a practical technology deployed in real-world systems, driven by the growth of digital data, advances in computing infrastructure, and algorithmic improvements that have made ML models more accurate, scalable, and efficient to train and deploy [4, 7, 16].

Industry has adopted ML across a wide range of applications, including crime prevention, medicine, information technology, cloud analytics, cybersecurity, and chatbots, for detecting patterns, predicting outcomes, and reducing manual work [1, 2, 11, 13]. However, ML-based systems have raised societal and technical concerns (e.g., challenges in formally verifying safety and risk in critical applications, biased decision-making involving protected categories, and persuasive techniques that monitor or influence individuals at scale) [9, 16], thus motivating quantitative measures that reflect the desired behavior of an ML-based system [1].

ML models are commonly evaluated by sampling methods such as holdout and cross-validation, which estimate predictive accuracy and support model selection by measuring errors on data excluded from training [20]. However, those methods primarily assess generalization and do not actively identify inputs that expose model weaknesses or unintuitive behaviors.

ML models evaluated in isolation involve analyses that typically emphasize standard performance metrics such as accuracy, recall, precision, and F1-score. Since ML-based solutions are embedded within broader software systems, they must also satisfy functional requirements that reflect intended behavior in context [1], thus adapting established software testing techniques to ML-based systems towards better evaluations of expected behaviors.

Mutation testing (MT) assesses test adequacy by generating modified versions of a program, called mutants, and checking whether the existing test suite distinguishes their behavior from that of the original program. A test case kills a mutant if it produces a different outcome for the mutant than for the original program. A test suite is more adequate when it kills a higher proportion of non-equivalent mutants. Therefore, MT evaluates whether tests execute the program and reveal potential defects, indicating the test suite's fault-detection capability [14].

MT has been adapted into a mutation-based testing criterion for decision tree (DT) models towards guiding test data selection and supporting a more systematic evaluation of model behavior. Under that criterion, small modifications to internal decision nodes create mutant trees that can be contrasted with the original ones and test inputs are selected when they induce different classifications in original and mutant trees [21]. Such criteria provide test requirements that guide the construction of adequate test sets, rather than relying on arbitrary selection [14].

This manuscript offers four main contributions, emphasizing cost reduction for DTMT-style and mutation-guided test data selection. First, we propose two threshold-specific mutation operators for decision-tree-based selection, namely, local boundary threshold (LBT) and impurity-aware threshold (IAT). Second, we compare LBT and IAT with decision tree mutation testing (DTMT) in terms of mutant-generation cost and the observed equivalent-mutant rate. Third, we evaluate whether the resulting test sets are more challenging than standard holdout and cross-validation baselines across six tabular classification datasets and three target classifier models (KNN, SVM, and MLP). Fourth, we analyze the overlap among the test sets selected by the three criteria to determine whether the lower-cost operators reproduce DTMT selection or provide complementary hard instances.

The remainder of the manuscript is organized as follows: Section 2 presents background concepts and related work; Section 3 discusses the motivation for this research; Section 4 introduces the threshold mutation operators; Section 5 describes the empirical

study design; Section 6 reports and discusses the results, and Section 7 addresses threats to validity. Finally, Section 8 provides the conclusions and outlines directions for future work.

2 Background

In this study, mutation testing is not used to mutate the implementation of the learning algorithm but as a criterion to identify input instances that distinguish the behavior of the original model or selection structure from that of its mutated variants. The following sections present the background concepts necessary to understand that application.

2.1 Evaluation of Machine Learning Models

The validation of supervised machine learning (ML) models typically involves controlled experiments that demonstrate effectiveness on specific datasets. The procedures must ensure correctness, validity, and reproducibility of both experiments and conclusions [7]. Given the statistical nature of ML models, evaluation should focus on aggregate behavior across representative, unseen data rather than isolated predictions and consider standard performance metrics and whether the selected measures align with system requirements and whether the model maintains reliability under deployment conditions [1, 7].

Among the different sampling methods that estimate the performance of supervised ML models, commonly used approaches include holdout and cross-validation, as shown in Figures 1 and 2. Each method specifies a distinct procedure for generating training and testing subsets, thus directly affecting that estimation.

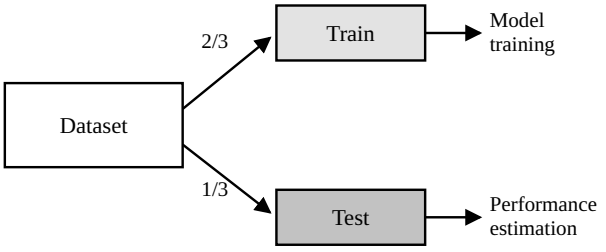


Figure 1: Holdout validation procedure used as a baseline criterion. Adapted from [7].

Figure 1 illustrates the holdout method, where the dataset is partitioned into distinct training and test sets. The approach may underestimate model accuracy, since models trained on the entire dataset typically outperform those trained on subsets [7]. That limitation is less significant for large datasets and no criterion has been established for a sufficiently large dataset [3]. Additionally, the holdout method does not evaluate performance variability across different train-test splits and may produce biased estimates if the test set contains easier instances [22].

Figure 2 illustrates the cross-validation (CV) method, where the dataset is partitioned into r equally sized folds and evaluated over r iterations. At each iteration, one fold is held out for validation and the remaining $r - 1$ folds are used for training. The final estimate is obtained by averaging across all folds [7, 20]. However, CV does not eliminate the need for a separate held-out test set for a final unbiased evaluation of the selected model [20]. A key limitation

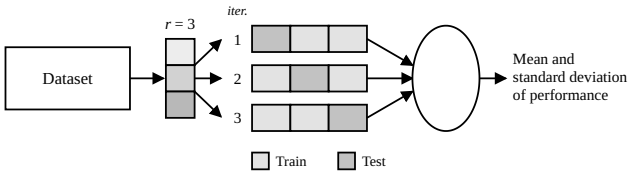


Figure 2: Cross-validation procedure used as a baseline criterion. Adapted from [7].

is the overlap of training subsets, reducing their independence, especially when r is small [7].

2.2 Mutation Testing

Mutation testing (MT) has emerged from the concept of perturbation and evaluates test data by introducing small systematic changes into programs for simulating potential faults. Its theoretical foundations include competent programmer hypothesis and coupling effect [14, 23]. The methodology assesses the possibility of improving software testing with a finite set of program errors [14].

In practice, MT is a defect-based technique in which mutation operators are applied to a program to generate similar versions, known as mutants. The adequacy of a test set is determined by its ability to distinguish those mutants from the original program [14, 23].

MT is applied through generation of mutants from the original program, executing both original program and mutants with the same test set, comparing their outputs, and analyzing the surviving mutants towards improving the test set or identifying equivalent mutants [14]. Figure 3 illustrates the mutation testing process and the way mutants are generated, executed against test cases, and analyzed until the test set has been adequate or the program has been revised.

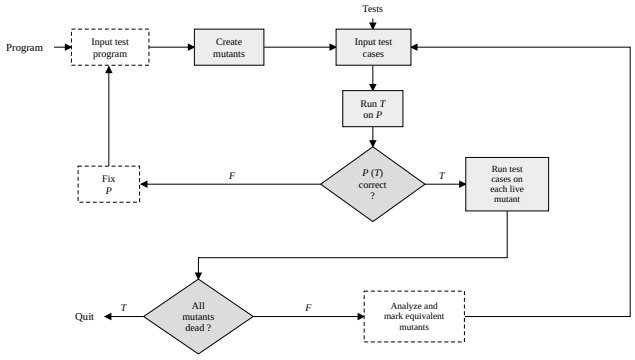


Figure 3: Traditional mutation testing workflow for mutant generation, execution, and analysis. Adapted from [23].

The traditional MT process submits the original program to a mutation system, which generates multiple mutated versions. Test cases executed on the original program verify if it produces the expected output. If the output is incorrect, the program must be fixed before continuing; if the output is correct, the same test cases run on the live mutants. A mutant is killed when its output differs from the original program. Mutants that remain alive are analyzed

towards analyses of whether they are equivalent or whether new test cases are required. The process repeats until all non-equivalent mutants have been killed [23].

Mutation score measures the adequacy of a test set in mutation testing, indicating proportion of non-equivalent mutants killed by the test cases. A mutant is killed if at least one test case produces a different output for it rather than for the original program. A higher mutation score suggests the test set is more effective at distinguishing faulty versions from the original one. Mutants that remain alive must be analyzed regarding their equivalence to the original program or necessity of additional test cases [14].

Equivalent mutants are excluded, since no test case can produce a different behavior between them and the original ones. A test set is mutation-adequate when it kills all non-equivalent mutants, reaching a 1.0 mutation score. Remaining mutants indicate weaknesses in the test set and may guide the creation of additional test cases [14, 23].

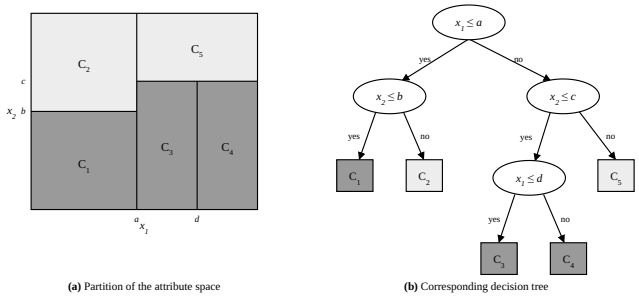
MT is based on two main theoretical assumptions. First, many program faults can be represented by simple syntactic changes that produce possible errors in mutants modeling . Second, the coupling effect assumes test cases that detect simple faults are also likely to detect more complex ones. Therefore, mutation testing focuses on killing simple mutants for indirectly evaluating the fault-detection strength of a test set [14, 23].

Despite its effectiveness, mutation testing faces significant practical difficulties, including high cost, since many mutants may be generated and each must be executed and compared with the original program. Surviving mutants require further analysis regarding whether they reflect weaknesses in the test set or are equivalent to the original program [14, 23]. The analysis often demands human judgment, since program equivalence is generally undecidable. Therefore, mutation testing can become expensive and time-consuming, motivating cost-reduction strategies such as random mutation, restricted mutation, selective mutation, and incremental application of mutation operators [14].

2.3 Decision Trees

Decision trees (DTs) are symbolic, human-interpretable models that organize a dataset into a sequence of feature tests, refining each choice towards a final classification or prediction [7]. Each path from root to leaf forms a conditional conjunction of attribute tests (an IF–THEN rule) that explains why a specific prediction was made [20]. Such transparency is valuable when explainability, compliance, and verification matter.

A DT represents a function that takes a vector of attribute values and returns a single output value called decision [20], reached through a sequence of tests, starting at the root and following the appropriate branch up to a leaf. Each internal node tests one input attribute. Branches are labeled with possible values of that attribute and leaf nodes specify the value to return [20]. Formally, a DT is a directed acyclic graph in which each node is either a split node, with two or more successors, or a leaf node [7]. The latter is labeled with a function and typically considers only the target variable values of examples that reach it, whereas a split node applies a conditional test to attribute values [7]. Figure 4 displays a decision tree and its corresponding partition within the space defined by attributes x_1 and x_2.



(a) Partition of the attribute space **(b)** Corresponding decision tree

Figure 4: Decision-tree structure and corresponding partition of the input feature space. Adapted from [7].

Each node of the tree corresponds to a region in that space. The regions defined by the leaf nodes are mutually exclusive and their union covers the entire space defined by the attributes. A DT can make predictions for any input example [7].

DT learning algorithm selects the attribute with highest importance [20]. For each possible test, the system considers the resulting data subsets and selects the test that maximizes or minimizes a given heuristic function over those subsets [7]. Towards guiding the process, attribute importance is commonly measured by impurity-based criteria (e.g., information gain, reduction expected in entropy after splitting of training examples based on an attribute, or Gini impurity, which measures the probability of misclassifying a randomly chosen sample labeled according to the class distribution in a node). Attributes that produce purer subsets are preferred during tree construction [7, 20].

DTs may overfit the training data, especially when the hypothesis space is large, as in trees with many nodes. The tree can capture noise or accidental regularities instead of patterns useful for future examples. Towards reducing the problem, decision tree pruning removes non-relevant tests and replaces subtrees with leaf nodes when the split has high impurity or little meaningful information. Such trade-off between consistency with the training data and simplicity of the learned hypothesis is essential for improving generalization [20].

2.4 Decision Tree-Based Testing Criteria

Recent studies have investigated DT models for defining test criteria and generating test cases, with promising results [15, 21]. DTs provide an interpretable internal structure in which internal nodes represent branching decisions over features and leaf nodes represent outcomes. Such a structure can guide the selection of test inputs for ML models [15].

The literature reports the following two test adequacy criteria for DT-based classification algorithms that use a tree built from the dataset to generate test cases systematically. Decision tree coverage (DTC) requires a test suite to traverse every path from the root to each leaf in a decision tree model at least once [15]. The criterion adapts principles from control-flow graph coverage, a widely used white-box testing approach for traditional systems. Since each leaf is reachable by a unique path, the number of test requirements for DTC equals the number of root-to-leaf paths in

the tree [15]. Boundary value analysis (BVA) targets the boundary values of decision nodes selecting values to test lower and upper limits of each decision and applies a functional testing approach that emphasizes sensitive regions of decision logic according to the assumption inputs near boundary conditions are particularly effective at revealing faults [15].

A test suite must include at least one test case for each path from the root to every leaf, ensuring a minimum of one test per leaf for satisfying DTC [15]. BVA focuses on decision boundaries at internal nodes and creates test instances that extend beyond the training data by defining boundary values as percentage changes relative to the observed split points in the tree [15]. Consequently, BVA may generate input values absent from the original dataset, since they are positioned just above or below each split threshold.

Decision tree mutation testing (DTMT), another DT-based criterion, was also proposed for test data selection [21]. A test suite for a decision tree is considered DTMT-adequate if it produces different classifications between the original tree and each of its mutant trees. DTMT is based on fault-based testing principles widely used to identify faults in conventional systems. Mutant trees are generated through application of mutation operators to the intermediate decision nodes of the original tree, simulating faults a developer might introduce into the model's decision logic [21].

A limitation of DTMT is the assumption only numerical values are present at decision nodes. An exhaustive search is sequentially conducted over the dataset for assembling the test set, negatively impacting performance on larger datasets [21].

Those criteria constitute the closest decision-tree-based baselines for this study which, in the next section, is positioned within broader research on testing and mutation testing for ML systems.

2.5 Related Work

Testing machine learning (ML) systems poses distinct challenges, since model behavior emerges from data, training procedures, and learned representations, rather than source code. Accordingly, surveys emphasize recurring themes such as test input generation, adequacy criteria, test oracles, and reliability assessment [19, 24].

More recent work has pursued stronger adequacy notions and model-aware guidance for automated software testing. Decision tree (DT) models inferred from black-box observations can steer test-case generation and support model-specific coverage metrics [18]. Mutation testing complements structural coverage by evaluating whether test cases detect injected faults; it can scale to industrial settings, and discrepancies between code coverage and mutation scores may expose weaknesses not captured by either metric alone [10, 17]. Boundary testing is also increasingly automated, with ML techniques identifying behavioral boundaries and improving fault detection relative to manual boundary-value analysis [8]. Together, those lines of work motivate combination of model-guided selection, threshold-focused generation, and mutation-based evaluation.

3 Motivation

This study was motivated by the assumption inputs distinguishing an original decision tree from its mutated variants tend to occur in regions of more sensitive model decision behavior [15, 21]. It hypothesizes sensitivity is closely related to internal split conditions and small perturbations around split thresholds may change the path an input takes and alter the predicted class. If a node splits on $x_f \leq 6.0$ and a mutant changes it to $x_f \leq 4.5$, then inputs with x_f between 4.5 and 6.0 are redirected to a different branch and may change the predicted class. Therefore, inputs near those decision boundaries are relevant candidates for testing.

Since attribute importance guides node splitting and supports decision tree induction [20], focusing on split thresholds provides measurable regions of interest from which test inputs can be selected. For test data selection, that means defining relevant input regions around main attributes. The choice of test cases is based on measurable decision criteria such as split thresholds or feature contributions. Such a hypothesis motivated the definition of two mutation operators for decision tree models, described in the next section.

4 Threshold Mutation Operators

This section introduces two threshold mutation operators for decision trees, namely, local boundary threshold (LBT) and impurity-aware threshold (IAT), which mutate one internal node threshold at a time, support test data selection for supervised machine learning classifiers, and preserve the original decision tree (DT) structure.

Let T be a decision tree trained on a dataset D. Each internal node n defines a split condition $x_f \leq t$, where f is the selected feature and t is the threshold learned by the tree induction algorithm. A threshold mutant T' is obtained by replacing t with an alternative threshold t', keeping the same feature, operator, children, and tree structure. A test instance kills that mutant when the prediction by T' differs from that by T. Let us consider an internal node n in T with split condition $x_f \leq 6.0$. A threshold mutant T' is created by replacing the original threshold with $t' = 4.5$, producing mutated condition $x_f \leq 4.5$. Feature, relational operator, children, and remaining tree structure remain unchanged. For an input instance with $x_f = 5.2$, the original tree follows the left branch, since $5.2 \leq 6.0$, but the mutant follows the right branch, for $5.2 > 4.5$. This branch change may lead the instance down a different path in the tree, potentially causing T and T' to predict different classes. If that occurs, the instance kills the mutant.

This study focuses on CART-style splits on ordered features, thus assuming decision nodes use numerical (or otherwise ordered) feature values. As a result, the proposed operators inherit the same limitation noted for DTMT when features are purely categorical with no imposed order.

DT thresholds define local decision boundaries. Small or plausible threshold changes can expose instances near sensitive regions of the model. Those instances are useful for test data selection, for they reveal cases in which small changes in the learned boundary alter the model's behavior.

4.1 Local Boundary Threshold (LBT)

LBT operator updates a node threshold by moving it to the closest valid candidate threshold, either just before or after the original split. The intuition is the mutant represents a tree that selected a neighboring local boundary instead of the one chosen by the original induction process.

Given an internal node n with threshold t, LBT first identifies candidate $c_j \in C_n$ closest to t and then generates up to two mutants: $t' = c_{j-1}$ and $t' = c_{j+1}$. If those neighboring candidates exist, the first mutant moves the threshold to the previous local boundary and the second to the next local boundary. If the original threshold is already first or last candidate, only one mutant can be generated and, if there are fewer than two distinct local values for the selected feature, no LBT mutant is generated for that node.

The operator is local, for it uses only the candidate thresholds at the node being mutated, and boundary-oriented, since the mutants affect instances closest to the original split. In practice, LBT produces small and interpretable changes, i.e., each mutant moves the decision boundary to the nearest alternative threshold on one side of the split.

Let us consider an internal node n with split condition $x_f \leq 6.0$ and assume the local candidate thresholds for feature f, computed from the instances reaching n, are $C_n = \{3.0, 4.5, 6.0, 8.0, 10.0\}$. Here, the original threshold is 6.0. Towards clarifying the LBT process, it identifies candidate thresholds immediately before and after the original split. The previous local boundary is 4.5 and the next is 8.0. Then, the operator generates two threshold mutants, namely, $x_f \leq 4.5$ and $x_f \leq 8.0$, which keep the same feature, relational operator, children, and tree structure. The only change is the threshold value. Let us consider an input with $x_f = 5.2$. It follows the left branch in the original tree $5.2 \leq 6.0$. With mutant $x_f \leq 4.5$, the same input follows the right branch. If the change causes a different classification, the input kills the mutant.

LBT can be interpreted as a mutation-based version of margin-based selection at the tree boundary. The feature values of instances that kill an LBT mutant are close enough to the original threshold to be redirected by the mutated split. Therefore, it tends to select instances near the decision boundary represented by the tree.

4.2 Impurity-Aware Threshold (IAT)

IAT operator mutates a node threshold by replacing it with alternative thresholds that are competitive under the impurity criterion used in tree induction. It focuses on plausible alternative splits the induction algorithm might have selected.

Given an internal node n, IAT evaluates all candidate thresholds in C_n for feature f used by the original node. For each candidate $c_i \in C_n$, instances in D_n are divided into left and right partitions using the split $x_f \leq c_i$. The weighted impurity of the resulting partitions measures the candidate's quality:

$$I(c_i) = \frac{|D_{\text{left}}|}{|D_n|} \text{impurity}(D_{\text{left}}) + \frac{|D_{\text{right}}|}{|D_n|} \text{impurity}(D_{\text{right}}) \quad (1)$$

The impurity function in Equation 1 uses the same criterion as the original tree (e.g., Gini[1] or entropy[2]) and the candidate closest to the original threshold is treated as the original split and excluded as a mutation target. The function selects the k best remaining candidates with the lowest weighted impurity values.

[1]Gini measures node impurity in classification or regression trees: $i(t) = 1 - \sum_{j=1}^{J} p(j \mid t)^2$, where $p(j \mid t)$ is the class proportion in node t. Lower values indicate purer nodes [5].

[2]Entropy measures the uncertainty of a random variable. For a random variable V with possible values v_k, each occurring with probability $P(v_k)$, entropy is defined as $H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k)$ [20].

Let us consider an internal node n with split condition $x_f \leq 6.0$ and local candidate thresholds $C_n = \{3.0, 4.5, 6.0, 8.0, 10.0\}$. Unlike LBT, IAT does not select candidates based only on proximity to the original threshold. Each candidate threshold is assessed by the impurity criterion from tree induction. Assuming the weighted impurity values are $3.0 \mapsto 0.48$, $4.5 \mapsto 0.30$, $6.0 \mapsto 0.27$, $8.0 \mapsto 0.33$, and $10.0 \mapsto 0.29$, the candidate matching the original split, 6.0, is omitted as a mutation target. If $k = 2$, IAT selects the two remaining thresholds with lowest weighted impurity values, 10.0 and 4.5, and generates the mutants $x_f \leq 10.0$ and $x_f \leq 4.5$.

IAT represents plausible but suboptimal threshold choices. Instead of moving the boundary only to the nearest candidate, it reconstructs the local split search and selects thresholds close to optimal based on impurity. Therefore, an IAT mutant can be seen as a tree that made a different yet reasonable split decision at one node.

4.3 Comparison with Decision Tree Mutation Testing (DTMT)

LBT and IAT operators are alternatives to the mutation operators used in DTMT. DTMT mutates internal decision nodes and selects test instances that produce different predictions in original and mutated trees. Each mutant tree represents a test requirement and a test instance satisfies it by killing the mutant, i.e., original and mutated trees classify the instance differently [21].

DTMT criterion uses ORRN and Cccr mutation operators. The former replaces a decision node's relational operator with another, for example, changing $x_f \leq t$ to $x_f > t$ or $x_f \geq t$. For instance, if a node tests $x_f \leq 6.0$, an ORRN mutant can change it to $x_f > 6.0$. That mutation flips the branch taken by every instance at that node and may redirect large portions of the input space.

Cccr replaces the constant in a decision node with another constant linked to the same feature from a different node. If node n_1 uses $x_f \leq 6.0$ and another node n_2 (possibly in a different subtree) uses the same feature with threshold 10.0, a Cccr mutant of n_1 may replace 6.0 with 10.0, yielding $x_f \leq 10.0$. Therefore, ORRN changes the comparison operator's semantics and Cccr changes the threshold value [21].

Those operators adapt mutation testing to DTs but have limitations. ORRN can cause abrupt changes by replacing a relational operator, potentially inverting or significantly altering the region assigned to a branch. Consequently, some mutants may be too strong, trivial, or equivalent to the original tree in the dataset. Cccr aligns more with threshold-based DT but replaces thresholds with constants from other nodes. Since those constants may not reflect the local data distribution at the mutated node, Cccr can produce syntactically valid threshold mutations less connected to the local split decision [21].

LBT and IAT narrow the mutation space by focusing on threshold changes, preserving feature, relational operator, child nodes, and the remainder of the tree structure—the only change is the threshold of one internal node. The design follows the observation that tree-structured models recursively partition the input space through binary splits in the classification and regression tree (CART) framework. CART uses threshold-based questions of the form $x_m \leq c$ [5] for ordered variables and threshold c defines the

local split boundary for a feature at an internal node. Mutating it directly explores model sensitivity around learned boundaries.

The main difference between DTMT and the proposed operators lies in the way alternative decision boundaries are selected. DTMT's Cccr chooses constants from other tree nodes, whereas LBT and IAT derive candidate thresholds from local instances reaching the mutated node. LBT selects adjacent candidate thresholds immediately before and after the original threshold and IAT selects the best alternatives according to the impurity criterion used by the tree-induction algorithm. Therefore, both produce mutations local to the node and tied to the learning algorithm's split search.

The difference also affects the number of generated mutants. ORRN generates several mutants per decision node, since multiple alternatives can replace each relational operator. Cccr may also generate multiple mutants, depending on number of constants associated with the same feature in other nodes. LBT and IAT impose a smaller and explicit mutation budget, since LBT generates at most two mutants per node and IAT generates at most k mutants per node. Table 1 summarizes the mutation rules considered. R is the set of relational operators, K_f is the set of thresholds used with feature f in other nodes, and C_n is the set of local candidate thresholds at node n. The last column highlights the main distinction between LBT and IAT. Both operators mutate thresholds according to the local split, whereas ORRN changes the comparison semantics and Cccr relies on non-local constants.

Table 1: Summary of DT mutation operators.

Operator	Changes	New value	Relation to local split
ORRN	Operator op	$op' \in R$	Changes branch semantics
Cccr	Threshold t	$t' \in K_f$	Uses non-local tree constants
LBT	Threshold t	c_{j-1} or c_{j+1}	Nearest local boundary
IAT	Threshold t	Best c_i by $I(c_i)$	Best local alternative split

5 Study Design

This section describes the empirical study that evaluated the proposed threshold mutation operators. It investigates whether local boundary threshold (LBT) and impurity-aware threshold (IAT) reduce the cost of decision tree mutation testing while preserving its ability to select challenging test inputs for supervised machine learning classifiers. It also compares LBT and IAT with the original decision tree mutation testing (DTMT) criterion and baseline test-selection strategies.

5.1 Research Questions

Research questions define the focus of a study and guide subsequent data collection and analysis activities [6]. The following research questions guided this study:

- **RQ1.** How much do LBT and IAT reduce the number of generated mutants compared with DTMT?
- **RQ2.** Do IAT and LBT select test sets that are more challenging than standard cross-validation and holdout references?
- **RQ3.** How do IAT and LBT compare with DTMT in equivalent-mutant rate under a lower mutant-generation cost?
- **RQ4.** Do DTMT, LBT, and IAT select the same test instances, or complementary ones?

RQ1 evaluates the cost of the proposed operators, since DTMT generates mutants through relational-operator and constant replacement and may produce many mutants per tree. RQ2 analyzes whether the selected test sets expose more challenging classification scenarios than validation references computed on the same data. RQ3 investigates whether the lower generation cost comes at the price of a higher proportion of equivalent mutants. RQ4 analyzes whether the cheaper operators merely reproduce the DTMT test set or select complementary instances.

5.2 Subject Datasets

The criteria are evaluated with six tabular classification datasets selected to cover supervised classification problems with different numbers of instances, features, classes, and degrees of class imbalance. Only tabular datasets are considered, since the proposed operators rely on CART-style decision tree thresholds over structured attributes. Three out of the six datasets are heavily imbalanced, with a majority class of up to 97.7% of the instances, directly motivating the choice of difficulty metric further described in this section. Those benchmark datasets are publicly sourced from UCI Machine Learning Repository [12].

Preprocessing is applied for each dataset prior to model training. Features are standardized once per dataset and the same standardized data are used by all criteria, classifiers, and references for ensuring comparability. None of the selected datasets contains missing values and a fixed median-imputation rule is defined for completeness. All preprocessing decisions are kept fixed across criteria, classifiers, and seeds. Table 2 summarizes the datasets used.

Table 2: Datasets used in the empirical study.

Dataset	Instances	Features	Classes	Majority (%)
Breast Cancer	569	30	2	62.7
Heart Disease	297	13	5	53.9
Mammography	11183	6	2	97.7
Oil Spill	937	49	2	95.6
Phoneme	5404	5	2	70.7
Sonar	208	60	2	53.4

5.3 Compared Criteria and Baselines

The experimental evaluation compared the proposed mutation operators with established baseline test-selection strategies, as described below.

DTMT. Adopted as the reference mutation-based criterion, it mutates decision tree nodes through relational-operator replacement and constant replacement and selects a test instance whenever that instance kills a mutant [21].

LBT. Mutates a decision node by replacing its threshold with the two adjacent candidate thresholds immediately preceding and following the original split. It therefore generates at most two mutants per internal node.

IAT. Mutates a decision node by replacing its threshold with k alternative candidate thresholds that yield the lowest weighted impurity. k was set to 2 so that the mutation budget of IAT remained comparable to that of LBT.

Holdout 70/30. The baseline follows a standard stratified holdout split using 70% of the data for training and 30% for testing, with the same seed as the corresponding run. It provides a sanity-check

comparison for assessment of whether the proposed criteria select more informative inputs than arbitrary sampling [7, 20].

Cross-validation. Stratified 10-fold cross-validation was used as the standard model-evaluation baseline, reflecting the common practice of estimating model performance from held-out folds rather than mutation-guided selection [7, 20].

5.4 Target Classifiers

The decision tree, used as a surrogate model for test selection, was configured as a scikit-learn CART tree, fitted with seed s, Gini splitting criterion, best splitter, unlimited depth, minimum split size of 2, and minimum leaf size of 1. All other hyperparameters remained at their default values in scikit-learn.

The selected test inputs were used for evaluating independent classifiers. Such a separation reduces the risk of results reflecting only the behavior of the tree used to generate the mutants. The following three supervised classifiers representing different learning mechanisms were considered: k-nearest neighbors (KNN), support vector machine (SVM), and multilayer perceptron (MLP). They represent instance-based, margin-based, and neural-network-based learning, respectively [7, 20].

KNN is instance-based, SVM is margin-based, and MLP is neural-network-based [7, 20]. No additional hyperparameter tuning was applied: KNN (k=3), SVM (RBF kernel), and MLP (one 100-unit hidden layer, ReLU, Adam, and `max_iter=1000`). Mutation-guided test sets difficult for all three suggest the selected inputs capture challenging regions of the input space rather than reflect one classifier family. Transfer is expected when the mutated-tree boundary changes isolate instances in locally ambiguous regions (e.g., near feature-value ties or class overlap) that also reduce margin or local neighborhood purity for other learners. However, the effect is not guaranteed and depends on the stability of the surrogate tree and on dataset geometry.

5.5 Metrics

The criteria were evaluated with the following metrics:

Number of mutants. Measures the cost of each mutation criterion. Fewer mutants indicate lower mutation-generation and execution cost [14, 23].

Observed equivalent-mutant rate. Measures the proportion of generated mutants that are not killed by any available instance in the dataset because they produce the same classifications as the original decision tree on those instances [14, 23].

Macro-F1 degradation. Test-set difficulty is measured as macro-F1 degradation $\Delta = F1_{\text{ref}} - F1_{\text{sel}}$, in percentage points (pp), where $F1_{\text{ref}}$ is the stratified 10-fold cross-validation score of the classifier on the same data and $F1_{\text{sel}}$ is its score on the selected test set. Positive values indicate the selected set is harder than a random sample of same data. This difficulty-as-degradation measure was introduced in this study for summarizing how much the mutation-selected set depresses performance relative to a standard validation reference computed on the same data. Macro-F1, instead of accuracy, was adopted, since three of the six datasets are heavily imbalanced, thus inflating accuracy-based references. The accuracy results led to the same conclusions.

Overlap metrics. Two descriptive set-overlap metrics, namely, (i) coverage, the percentage of DTMT-selected instances that are also selected by LBT (or IAT), and (ii) exclusivity, the percentage of the union of the three selected sets that is selected by exactly one criterion. Those measures are introduced in this study as simple proxies for how much the cheaper operators reproduce DTMT selection (coverage) and how complementary the criteria are as sources of hard instances (exclusivity). They should be interpreted as descriptive overlap rather than established adequacy measures.

5.6 Experimental Procedure

The experimental procedure is executed independently for each dataset and random seed. First, the dataset is preprocessed and a decision tree is trained. The tree is not the final model under test but serves as a white-box selection structure from which mutation-based test requirements are derived.

Mutants are then generated according to each mutation criterion. DTMT generates them using its original operators and LBT and IAT generate threshold mutants. Next, each mutant is executed against the available data. A test instance kills a mutant when original and mutant trees produce different predictions for that instance. For each killed mutant, the first killing instance found under a seeded random permutation of the dataset enters the test set and the selected test set for each criterion is the union of those instances over all killed mutants, following the selection rule of the original DTMT proposal [21]. Although the rule does not explicitly pick the instance closest to the mutated threshold among all killing instances, every killing instance must fall into the redirected region induced by the mutated split (i.e., it crosses the mutated boundary at that node) so that the procedure still targets boundary-sensitive regions. Selecting the closest killing instance is a possible refinement.

The selected instances are then removed from the training data for the target classifiers trained on the remaining instances and evaluated on the selected test set. The procedure avoids training and testing on the same instances. Next, the selected test sets are compared with holdout 70/30 (for which the process is repeated with the same seeds) and 10-fold cross-validation baselines.

Finally, evaluation metrics are computed on the selected test set for each criterion. The number of generated mutants and the equivalent-mutant rate measure cost. Macro-F1 degradation of each target classifier relative to the cross-validation reference measures test-set difficulty and the overlap among DTMT, LBT, and IAT test sets is computed to analyze whether the criteria select the same or complementary instances. The values are then compared across criteria and against the references.

Algorithm 1 summarizes the experimental procedure.

The experiment is repeated with 30 seeds to account for variability in model training, test selection, and random baselines. The analysis is descriptive, i.e., results are summarized over 30 seeds and no formal significance tests or effect sizes are reported.

6 Results

This section reports the results for each research question. All values are means over 30 seeds. Since the selected test sets are small (median of 58 instances, maximum of 1,231), win rates are computed over all 540 paired runs (6 datasets × 30 seeds × 3 classifiers).

Algorithm 1 Experimental procedure

```
for each dataset D and seed s do
    Preprocess D and train decision tree T
    for each criterion C ∈ {DTMT, LBT, IAT} do
        Generate mutants from T using C
        For each killed mutant, select the first killing instance under a seeded permutation
        Train KNN, SVM, and MLP excluding selected instances
        Evaluate classifiers and compute metrics
    end for
    Compare criteria with holdout and cross-validation references
    Compute the overlap among DTMT, LBT, and IAT test sets
end for
```

6.1 RQ1: Mutant-Generation Cost

Table 3 provides mean number of mutants generated by each criterion and percentage savings of the threshold operators relative to DTMT.

Table 3: Mutants generated (mean over 30 seeds) and savings relative to DTMT.

Dataset	DTMT	LBT	IAT	LBT sav. (%)	IAT sav. (%)
Breast Cancer	84	33	42	60.8	50.1
Heart Disease	1,200	104	128	91.3	89.3
Mammography	4,655	252	293	94.6	93.7
Oil Spill	139	51	61	63.5	55.7
Phoneme	55,303	847	956	98.5	98.3
Sonar	72	35	41	51.8	42.8
Pooled mean	10,242	220	253	97.9	97.5
Per-dataset mean				76.7	71.6

LBT and IAT generate fewer mutants than DTMT across all six datasets, with per-dataset savings ranging from 51.8% to 98.5% for LBT and from 42.8% to 98.3% for IAT. Their unweighted mean savings are 76.7% and 71.7%, respectively. Pooled means (220 and 253 mutants versus 10,242 for DTMT, a 97.9% and 97.5% reduction) are dominated by the two datasets that produce the largest trees, namely, Phoneme and Mammography, which should be read as aggregate totals rather than typical per-dataset values. The gap grows with tree size, since Cccr is quadratic in number of nodes sharing a feature, whereas LBT and IAT generate at most two mutants per node.

6.2 RQ2: Test-Set Difficulty

Table 4 provides macro-F1 of the cross-validation reference and the degradation Δ (pp) per criterion. Higher Δ means a harder selected test set. Table 5 shows the win rate, i.e., fraction of paired runs in which the selected set yielded a lower macro-F1 than the reference.

The overall degradation ranges from 10.6 pp (IAT) to 18.8 pp (LBT), and the selected set is harder than the reference in 91–98% of the 540 paired runs.

Although the test sets are selected from a decision-tree surrogate, the difficulty transfers to non-tree classifiers (KNN, SVM, and MLP) since threshold mutations tend to redirect instances near local class overlap regions in the feature space, where multiple decision boundaries are plausible. Instances close to such ambiguous regions are often harder regardless of the specific classifier family. Such transfer is not guaranteed and depends on how well the surrogate tree approximates the structure of the dataset.

Table 4: Macro-F1 of the cross-validation reference and degradation Δ (pp) per criterion (mean over 30 seeds and 3 classifiers).

Dataset	Reference F1	Δ DTMT	Δ LBT	Δ IAT
Breast Cancer	97.0	14.1	20.7	11.1
Heart Disease	28.6	4.1	6.2	2.7
Mammography	82.0	21.3	33.0	18.6
Oil Spill	71.6	18.4	17.0	6.1
Phoneme	83.6	17.2	26.1	17.6
Sonar	84.9	15.3	10.2	7.5
Mean	74.6	15.1	18.8	10.6

Table 5: Win rate: paired runs in which the selected set yielded a lower macro-F1 than the cross-validation reference.

Criterion	Runs harder than reference	Win rate (%)
DTMT	519 / 540	96.1
LBT	531 / 540	98.3
IAT	491 / 540	90.9

The three criteria select test sets consistently harder than a random sample of same data, i.e., degradation ranges from 6.2 to 33.0 pp for LBT and from 2.7 to 18.6 pp for IAT per dataset, and the selected set is harder than the reference in 91–98% of the 540 paired runs. Heart Disease dataset has the lowest reference macro-F1 (28.6), suggesting a harder multi-class problem under the fixed classifier configurations and dataset size. That lower baseline also limits headroom for additional degradation compared with the other datasets. Unlike the pooled mutant counts of RQ1, the overall row of Table 4 weights each dataset equally (30 seeds × 3 classifiers per dataset); therefore, it is not dominated by larger datasets.

6.3 RQ3: Equivalent Mutants

A mutant is considered equivalent if no instance in the dataset distinguishes it from the original tree. It consumes generation and execution time but contributes no test instances. This is an upper-bound approximation, since mutants not killed by the available data are treated as equivalent. Table 6 provides the equivalence rate per dataset and Table 7 summarizes generation efficiency as the expected number of mutants generated for obtaining a useful (killable) one.

Table 6: Equivalent-mutant rate (%) per dataset (mean over 30 seeds).

Dataset	DTMT	LBT	IAT
Breast Cancer	26.0	23.1	20.2
Heart Disease	18.6	6.3	3.4
Mammography	8.7	16.7	19.1
Oil Spill	26.5	18.2	19.5
Phoneme	4.8	16.4	14.3
Sonar	30.3	12.7	9.7
Eq. rate	19.1	15.6	14.4

Overall, LBT and IAT waste a smaller fraction (15.6% and 14.4% versus 19.1% for DTMT, with each dataset weighted equally in the mean) of their mutants than DTMT (19.1%). Their absolute waste is two orders of magnitude smaller, since they generate far fewer mutants (RQ1). Advantages vary. DTMT has the lowest equivalence

Table 7: Generation efficiency (mean over datasets and seeds). Gen. per useful = $1/(1 - \text{equivalence rate})$.

Criterion	Eq. rate (%)	Gen. per useful	Mutants	Equivalent
DTMT	19.1	1.24	10,242	557
LBT	15.6	1.18	220	35
IAT	14.4	1.17	253	37

rate on the two largest trees (Mammography and Phoneme); however, threshold-based operators dominate the other four datasets. The absolute columns of Table 7 are dominated by the two large-tree datasets so that the pooled absolute waste of DTMT corresponds to a smaller fraction than the unweighted mean rate in Table 6.

6.4 RQ4: Overlap Among Selected Test Sets

RQ4 asks whether the cheap criteria simply rederive the DTMT test set or select genuinely different instances. Tables 8 and 9 show, respectively, DTMT set coverage and composition of the union of the three selected sets and Figure 5 displays the aggregated overlap.

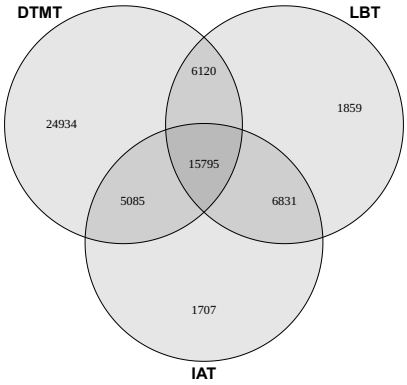


Figure 5: Aggregated overlap of the test sets selected by DTMT, LBT, and IAT. Source: Elaborated by authors.

Table 8: Coverage of DTMT set: % of DTMT-selected instances also selected by each threshold operator (mean over 30 seeds).

Dataset	LBT (%)	IAT (%)
Breast Cancer	31	28
Heart Disease	52	54
Mammography	36	32
Oil Spill	31	33
Phoneme	44	42
Sonar	27	27
Mean	37	36

Table 9: Composition of DTMT ∪ LBT ∪ IAT (%): percent of instances in the union that fall into each region.

Dataset	DTMT only	LBT only	IAT only	Shared (≥2)
Breast Cancer	28	7	12	53
Heart Disease	33	3	1	64
Mammography	45	2	4	49
Oil Spill	29	9	9	53
Phoneme	41	2	2	55
Sonar	19	13	14	54
Mean	32	6	7	55

LBT and IAT recover on average only 37% and 36% of the DTMT set, i.e., roughly two thirds of the instances selected by the expensive baseline are not found by cheap operators. Conversely, 32% of the union are exclusive to DTMT, whereas the two threshold operators contribute 6% and 7% of exclusive instances each. The criteria are complementary sources of hard instances rather than substitutes. Combining the two cheap operators yields a set whose difficulty lies between them (Table 10), i.e., LBT-exclusive instances harden the IAT set, although the union does not surpass LBT alone.

Table 10: Macro-F1 (×100) evaluated on the selected subset (LBT, IAT, or LBT ∪ IAT), averaged over seeds and classifiers. Lower means harder.

Dataset	LBT	IAT	LBT ∪ IAT
Breast Cancer	76.4	86.0	79.1
Heart Disease	22.4	25.9	24.1
Mammography	49.0	63.3	52.2
Oil Spill	54.7	65.6	57.4
Phoneme	57.5	66.0	60.1
Sonar	74.7	77.4	75.8
Mean	55.8	64.0	58.1

6.5 Summary of Findings

RQ1. LBT and IAT generate fewer mutants than DTMT on every dataset, with per-dataset savings of 51.8–98.5% (LBT) and 42.8–98.3% (IAT) that grow with tree size. Pooled totals fall by 97.9% and 97.5%. **RQ2.** Every criterion degrades macro-F1 relative to a random sample of same data (overall Δ of 15.1 pp for DTMT, 18.8 pp for LBT, and 10.6 pp for IAT) in 91–98% of the runs. **RQ3.** LBT and IAT waste fewer mutants than DTMT (15.6% and 14.4% versus 19.1%) with absolute waste two orders of magnitude smaller, whereas DTMT retains the lowest rate on the two largest trees. **RQ4.** Cheap operators are not a shortcut to the DTMT set, since they recover only 36–37% of it and 32% of the union are exclusive to DTMT — the criteria select largely different, complementary hard instances.

7 Threats to Validity

Standard categories of internal, construct, conclusion, and external validity are used for the discussion of threats to interpretation of results and reproducibility considerations are summarized.

Internal validity. The operators act on a surrogate classification and regression tree (CART) learned with fixed induction choices (seed, depth, split criterion). Threshold candidates, mutants, and selected instances depend on the learned tree and different tree hyperparameters may change the selected sets.

Construct validity. Test-set difficulty is operationalized via accuracy and macro-F1 of KNN, SVM, and MLP models, which are indirect proxies for fault detection in deployed ML systems. Degradation and overlap measures depend on the reference procedure chosen and summarize set relationships rather than practical complementarity.

Conclusion validity. Results are descriptive (30 seeds) with no significance testing, confidence intervals, or paired-effect sizes. Aggregation choices matter, since pooled mutant totals are dominated by datasets with largest trees so that per-dataset summaries should guide interpretation.

External validity. The study is limited to six tabular classification datasets, CART-style ordered-feature splits, and three classifiers; therefore, conclusions may not transfer to other data

types, tasks, models, or purely categorical splits without an ordering/encoding.

Reproducibility. Reproduction requires access to the datasets, preprocessing pipeline, model configurations, random seeds, and mutation scripts. They should be documented and a replication package should be released when possible.

8 Conclusion

This manuscript introduced two threshold mutation operators, namely, local boundary threshold (LBT) and impurity-aware threshold (IAT) towards improving mutation-based test data selection for decision-tree classifiers. Unlike previous decision tree mutation testing (DTMT) operators, they focus on local threshold changes while preserving the original tree structure and semantics.

The threshold operators reduce mutant-generation cost on every dataset, with savings between 51.8% and 98.5% (LBT) and between 42.8% and 98.3% (IAT) and waste a smaller average fraction of mutants as equivalent (15.6% and 14.4% versus 19.1% for DTMT). The selected test sets remain consistently challenging, degrading macro-F1 by 6.2–33.0 pp (LBT) and 2.7–18.6 pp (IAT) relative to a cross-validation reference, in 91–98% of the paired runs.

Despite not fully replacing DTMT, LBT and IAT reproduce much of DTMT's test-selection behavior at a substantially lower cost and with fewer equivalent mutants. In general, the findings suggest local threshold mutation is a promising strategy for reducing the cost of mutant generation while selecting challenging test data for decision-tree-based machine learning testing.

9 Future Work

Future work will assess impurity-aware threshold (IAT)'s k, alternative surrogate-tree configurations, threshold-proximal instance selection, and extensions to ensemble methods.

Artifact Availability

The artifact is available at https://github.com/waisman-braga/threshold-mutation-testing-operators.

Acknowledgements

The authors acknowledge CNPq for support, under process n. 306719/2025-8 and 406941/2025-4, and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), through Programa de Excelência Acadêmica (PROEX).

References

[1] Samuel Ackerman, Guy Barash, Eitan Farchi, Orna Raz, and Onn Shehory. 2024. *Theory and Practice of Quality Assurance for Machine Learning Systems: An Experiment-Driven Approach.* Springer Nature Switzerland, Cham. doi:10.1007/978-3-031-70008-8

[2] Gabriel Araujo, Marcos Kalinowski, Markus Endler, and Fabio Calefato. 2024. Professional Insights into Benefits and Limitations of Implementing MLOps Principles. In *International Conference on Enterprise Information Systems, ICEIS - Proceedings*, Vol. 2. Science and Technology Publications, Lda, 305–312. doi:10.5220/0012741100003690

[3] José Augusto Baranauskas and Maria Carolina Monard. 2000. *Reviewing Some Machine Learning Concepts and Methods.* Technical Report 102. Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, Brazil. https://repositorio.usp.br/directbitstream/714e0ea9-4a3a-4c57-b5fb-e07e8dc0f852/BIBLIOTECA_113_RT_102.pdf Technical report, ICMC-USP.

[4] Christopher Bishop. 2011. Embracing Uncertainty: Applied Machine Learning Comes of Age. In *Machine Learning and Knowledge Discovery in Databases*, Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 4–4.

[5] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. 1984. *Classification and Regression Trees.* Chapman & Hall/CRC, Boca Raton London New York Washington, D.C. doi:10.1201/9781315139470

[6] Katia Romero Felizardo et al. 2017. *Revisão sistemática da literatura em engenharia de software: teoria e prática.* Elsevier.

[7] Katti Faceli, Ana Carolina Lorena, João Gama, Tiago Agostinho de Almeida, and André Carlos Ponce de Leon Ferreira de Carvalho. 2025. *Inteligência Artificial: Uma Abordagem de Aprendizado de Máquina* (3 ed.). LTC, Rio de Janeiro. 376 pages.

[8] Xiujing Guo, Hiroyuki Okamura, and Tadashi Dohi. 2025. Improving Test Suite Generation Quality Through Machine Learning-Driven Boundary Value Analysis. *Array* 27 (2025), 100496. doi:10.1016/j.array.2025.100496

[9] Khan Mohammad Habibullah, Juan Garcia Diaz, Gregory Gay, and Jennifer Horkoff. 2024. Scoping of Non-Functional Requirements for Machine Learning Systems. In *Proc. Int. Conf. Requir. Eng.* IEEE Computer Society, 496–497. doi:10.1109/RE59067.2024.00061

[10] Kush Jain, Goutamkumar Tulajappa Kalburgi, Claire Le Goues, and Alex Groce. 2023. Mind the Gap: The Difference Between Coverage and Mutation Score Can Guide Testing Efforts. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE).* IEEE, 102–113. doi:10.1109/ISSRE59848.2023.00036

[11] Teru Kamogashira. 2022. Machine Learning in Diagnosis Support with Posturography Data. *Equilibrium Research* 81, 4 (2022), 212–221. doi:10.3757/jser.81.212

[12] Markelle Kelly, Rachel Longjohn, and Kolby Notkin. 2025. UCI Machine Learning Repository. doi:10.24432/C5K33Z Available at https://uci.edu.

[13] Yuta Kobayashi, Yu-Chung Peng, Evan Yu, Brian Bush, Youn-Hoa Jung, Zachary Murphy, Lee Goeddel, Glenn Whitman, Archana Venkataraman, and Charles H. Brown. 2023. Prediction of Lactate Concentrations after Cardiac Surgery Using Machine Learning and Deep Learning Approaches. *Frontiers in Medicine* 10 (Sept. 2023). doi:10.3389/fmed.2023.1165912

[14] José Carlos Maldonado. 2017. *Introdução ao teste de software 2a ed.* Elsevier.

[15] Beatriz N. C. Silveira, Vinicius H. S. Durelli, Sebastião H. N. Santos, Rafael S. Durelli, Marcio E. Delamaro, and Simone R. S. Souza. 2025. Advancing Test Data Selection by Leveraging Decision Tree Structures: An Investigation into Decision Tree Coverage and Mutation Analysis. *Journal of Software Engineering Research and Development* 13, 1 (March 2025). doi:10.5753/jserd.2025.4084

[16] Peter Norvig. 2022. *Inteligência Artificial* (4 ed.). Grupo Gen, Rio de Janeiro, RJ.

[17] Goran Petrović, Marko Ivanković, Gordon Fraser, and René Just. 2022. Practical Mutation Testing at Scale: A View from Google. *IEEE Transactions on Software Engineering* 48, 10 (2022), 3900–3912. doi:10.1109/TSE.2021.3107634

[18] Swantje Plambeck and Görschwin Fey. 2023. Data-Driven Test Generation for Black-Box Systems From Learned Decision Tree Models. In *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS).* IEEE, 27–32. doi:10.1109/DDECS57882.2023.10139633

[19] Vincenzo Riccio, Gunel Jahangirova, Andrea Stocco, Nargiz Humbatova, Michael Weiss, and Paolo Tonella. 2020. Testing Machine Learning Based Systems: A Systematic Mapping. *Empirical Software Engineering* 25, 6 (2020), 5193–5254. doi:10.1007/s10664-020-09881-0

[20] Stuart J. Russell and Peter Norvig. 2022. *Artificial Intelligence: A Modern Approach* (fourth edition, global edition ed.). Pearson, Boston.

[21] Beatriz Silveira, Vinicius Durelli, Sebastião Santos, Rafael Durelli, Marcio Delamaro, and Simone Souza. 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* (Campo Grande/MS). SBC, Porto Alegre, RS, Brasil, 38–46. https://sol.sbc.org.br/index.php/sast/article/view/28213

[22] C C Taylor, etc., D J Spiegelhalter, and D Michie (Eds.). 1994. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood Ltd, Publisher, Harlow, England.

[23] W. Eric Wong (Ed.). 2001. *Mutation Testing for the New Century.* Springer US, Boston, MA. doi:10.1007/978-1-4757-5939-6

[24] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. 2020. Machine Learning Testing: Survey, Landscapes and Horizons. *IEEE Transactions on Software Engineering* (2020).