Automated Boundary Identification for Machine Learning Classifiers

Citation for the original published paper (version of record):
Dobslaw, F., Feldt, R. (2024). Automated Boundary Identification for Machine Learning Classifiers. 2024 IEEE/ACM INTERNATIONAL WORKSHOP ON SEARCH-BASED AND FUZZ TESTING, SBFT 2024: 1-8. http://dx.doi.org/10.1145/3643659.3643927

Automated Boundary Identification for Machine Learning Classifiers

Felix Dobslaw
Mid Sweden University
Dept. of Quality Mngmt, Communication and Inf. Systems
Östersund, Sweden
felix.dobslaw@miun.se

Robert Feldt
Chalmers University of Technology
Dept. of Computer Science and Engineering
Gothenburg, Sweden
robert.feldt@chalmers.se

ABSTRACT

AI and Machine Learning (ML) models are increasingly used as (critical) components in software systems, even safety-critical ones. This puts new demands on the degree to which we need to test them and requires new and expanded testing methods. Recent boundary-value identification methods have been developed and shown to automatically find *boundary candidates* for traditional, non-ML software: pairs of nearby inputs that result in (highly) differing outputs. These can be shown to developers and testers, who can judge if the boundary is where it is supposed to be.

Here, we explore how this method can identify decision boundaries of ML classification models. The resulting ML Boundary Spanning Algorithm (ML-BSA) is a search-based method extending previous work in two main ways.We empirically evaluate ML-BSA on seven ML datasets and show that it better spans and thus better identifies the entire classification boundary(ies). The diversity objective helps spread out the boundary pairs more broadly and evenly. This, we argue, can help testers and developers better judge where a classification boundary actually is, compare to expectations, and then focus further testing, validation, and even further training and model refinement on parts of the boundary where behaviour is not ideal.

ACM Reference Format:
Felix Dobslaw and Robert Feldt. 2024. Automated Boundary Identification for Machine Learning Classifiers. In *2024 ACM/IEEE International Workshop on Search-Based and Fuzz Testing (SBFT '24), April 14, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3643659.3643927

1 INTRODUCTION

Artificial Intelligence (AI) and Machine Learning (ML) models are increasingly used both during software development but also integrated into software systems themselves [1, 7]. While the use cases and goals of such models vary widely, models that do some form of classification into a small set of finite classes are commonplace. Well-known examples from the literature are loan approvals, insurance coverage, drug prescription, or user verification [18]. In many cases, the form of the ML model doesn't lend itself to human understanding, i.e., the models are opaque, and it is hard to understand and explain why the model predicts one class over another. This raises issues of fairness, justice, and ethics, and there are several known cases where biases in the training data lead to classification models favouring one group over another [17, 18]. Some researchers even argue that opaque ML models (black boxes) should not be used in safety-critical scenarios at all [18].

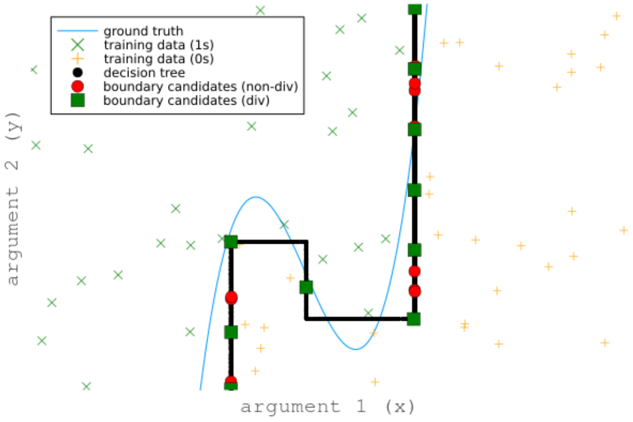

Figure 1: The plot illustrates the decision tree boundaries and two sets of boundary candidates over the simple non-linear synthetic binary classifier (*synth*) that separates all inputs (x and y coordinates) into *valid* and *invalid*. We argue for the need to identify boundaries as well-diversified boundary candidate sets.

Regardless of the ML model form and the level to which it can be inspected, software developers that now also develop and integrate ML models in their systems need methods and testing tools to describe model behaviour. Even if a model can be studied in its static form, it is not clear that all of its dynamic behaviours, when executed on inputs, can be clearly understood. From a software testing viewpoint, static or formal analysis of ML models is not enough to test models only close to where they have been trained. In particular, for ML classification models, testing methods that can identify the decision boundaries between the classes could help developers judge if the current model is good enough or if further refinement and testing are needed.

In the current ML research literature, there has been some focus on the boundaries, mainly when it comes to identifying counterfactual explanations (CFE) and in so-called adversarial example generation [10, 19, 20]. While these methods differ in purpose, they both try to find small changes to existing inputs that change the model output. However, since they focus on specific starting inputs,

SBFT '24, April 14, 2024, Lisbon, Portugal
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0562-5/24/04.
https://doi.org/10.1145/3643659.3643927

they are not focused on identifying decision boundaries that are further away from the training data. From a testing perspective, this is not ideal and would, metaphorically, correspond to only testing a piece of software close to the normal, already-known inputs. We need methods to systematically explore and span the decision boundaries of ML classification models around the training data and in less explored areas of the input space.

Recently, progress has been made to identify boundaries for traditional software automatically, i.e. automating the classical testing technique of boundary value analysis/testing [4, 6, 8, 14]. In this paper, we explore how these techniques can be used to address the ML testing challenges outlined above, focusing on the commonalities, i.e. any ML model is also a piece of software and thus can be tested as such, rather than the differences [5]. We, therefore, extend the concept of boundary squeezing, which has been previously proposed for traditional software testing [14], with new methods for automated boundary testing [4, 6] into a method to span the boundaries of ML classification models more systematically. The key underlying idea is the program derivative [8], which helps circumvent the oracle problem by offering a way to zoom in on or squeeze two points on different sides of a boundary. Two points are a good boundary candidate pair if they are close together in the input space while having as different outputs as possible. However, from a testing perspective, it is not sufficient to find any boundary candidates. We want candidates that reflect different parts of the input space. They can then help point engineers and testers to interesting areas in the input space, preferably with guidance and tooling that removes complexity and limits the selection of escalated boundary candidates or situations to the bare minimum to reduce overwhelm. Thus, these cases should ideally be well separated and spread along the decision boundary. Thus, we try to answer the question:

RQ How can we detect decision boundaries for ML classification models?

As part of the question, we want to understand whether we can obtain a set of well-diversified candidates using recent automated testing method achievements for traditional software. We exemplify the problem with a synthetic binary classifier *synth* and evaluate it on seven datasets. *synth* has two inputs, x and y, as well as a single output for the separation into valid and invalid outputs or classes, resulting in a binary classification problem. The function is $synth(x) = (x+2)(x^2-4)$, and the respective ground truth classifier that defines the boundary is shown in Figure 1 together with a selection of sampled points, implementing:

$$cl_{synth}(x,y) = \begin{cases} 1 & synth(x) < y \\ 0 & \text{else.} \end{cases} \tag{1}$$

The figure further includes a decision tree (black) and two alternative boundary candidate sets (red and green) of differing quality that result from the experiments from this study and which we will discuss in more detail below.

The remainder of the paper is structured as follows. Section 2 overviews related traditional testing literature and ML problems. We propose ML-BSA and the experimental design in section 3, Method. Section 4 presents the results, and section 5 discusses implications and avenues for future work. Section 6 summarizes the main findings and concludes the paper.

2 RELATED WORK

Two areas of Machine Learning related to our work are counterfactual explanations [10, 19] (CFE) and adversarial example generation [20]. They both create new data points similar to an original point but lead to different predictions. In adversarial machine learning, the predictions in these new points are considered erroneous or even dangerous. Techniques have been proposed to "defend" models from being "attacked" by such adversarial methods [2]. On the other hand, in counterfactual explanations, the predictions in the newly generated points are viewed as correct. They are rather used to illustrate to users what needs to change in the data or their situation for the prediction to differ. This can be seen as a type of human-understandable explanation of why the prediction is what it is.

Our work differs from both methods in that we focus on the whole decision boundary and not only aim to find specific examples close to already known inputs. Our approach is agnostic to whether the predictions are actually correct or not. By pinpointing the boundary areas, i.e., where predictions change the most for small changes in inputs, they can be checked and their correctness judged compared to expectations. An important aspect is that knowing where the decision boundary lies might be more important far away from training data points than close to them since the uncertainty of the model can be expected to be higher there.

The CFE technique of Mothilal *et al.* generates a whole set of counterfactual points but also ensures that they are diverse, i.e., change different features of the original inputs and/or change them in different ways [16]. They thus include a measure of diversity when searching for the counterfactual points. Brughmans *et al.* [3] proposed a CFE generation method that can optimize for multiple counterfactual properties, e.g., few inputs changed (short counterfactuals), small overall change (near counterfactuals), or realistic changes (similar to other input differences among the training data). A recent review of the literature [12] showed that many different diversity measures have been used in CFE generation. Like these approaches, our approach also returns a set of diverse points. However, staying close to the original points is unimportant in our approach; our goal is to characterize the decision boundary.

Joshi *et al.* [11] also postulate that examples are useful for understanding classifier decision boundaries. However, they need to train a generative model to ensure that such examples are realistic, which limits applicability, increases computational costs, and assumes that the training data represents all of the actual future inputs.

Our work also connects to methods that focus on the uncertainty of machine learning models. For example, Ma *et al.* [13] introduce model uncertainty metrics for deep neural nets and evaluate how well they can be used to prioritize test cases. They find that a simple metric like the probability of the selected, highest probability, class can indicate uncertainty as well as correlate with mis-classification, i.e. where the decision boundary is misplaced. They show how this can be used to prioritise existing test data and generate adversarial inputs.

In contrast to the existing approaches described above, we build on our work on automating boundary value testing of traditional software [6]. Our approach is not tied to being close to existing or available input points but rather finds pairs of inputs that show where the decision boundary lies. The approach is black-box and model-agnostic and builds on the idea of program derivatives [8], i.e. pairs of inputs that are very close/similar but whose outputs differ as much as possible. We extend this method to testing ML models and try to ensure coverage of the decision boundary by optimizing to find a diverse set of such boundary pairs.

3 METHOD

We here first introduce the search-based algorithm in Section 3.1 to then explain the boundary diversification approach in Section 3.2. Section 3.3 presents the datasets and preprocessing procedures followed by the experimental design in Section 3.4.

3.1 ML Boundary Spanning

The ML Boundary Spanning algorithm (ML-BSA) is described in Algorithm 1. ML-BSA aims to detect a set of boundary candidates that capture a wide range of the actual decision boundary (or boundaries for multi-class problems) of a classification model. It takes in a trained classifier *cl* and the data it was trained on TD consisting of tuples (i, o) with input vector $i \in I$ and output classes $o \in O$ to return the set of boundary candidate pairs, $bc = ((i_1, o_1), (i_2, o_2)) \in BC \subset (I \times O)^2$, for which $o_1 \neq o_2$ and the following holds for an input space specific threshold δ, and specified distance function $d_{in} : d_{in}(i_1, i_2) < \delta$. Initially (line 2) all possible outputs or *classes* are extracted (for binary classifier *synth*, O=$\{0, 1\}$). Until a stop criterion holds, ML-BSA samples new boundary candidates by randomly selecting two distinct outputs, o1 and o2 (line 4, for synth only one possible combination), and then drawing a respective data point from TD each (lines 5 and 6). The data points are then passed along as a candidate pair c (line 7) to inform the search (line 8). During a search, candidate fitness is measured as the program derivative (maximization) between the candidate points[1] as introduced in [8] and slightly adjusted with

$$pd_{cl}(bc_1, bc_2) = \frac{d_{out}(cl(c_1), cl(c_2))}{d_{in}(c_1, c_2) + \delta}, \tag{2}$$

where δ is a distance vector containing a defined minimum/atomic distance for each dimension to avoid division by zero and help signalling convergence to a boundary candidate. d_{in} is the metric measuring the multidimensional distance between the candidate inputs, and d_{out} is the distance metric measuring the distance between the outputs produced by the classifier *cl*. In our ML classification, d_{out} is defined as 0 if the outputs are equal and 1 otherwise. Once a candidate pair obtains input distances $d_{in}(c_{1j}, c_{2j}) < \delta_j$ for all dimensions, the search returns that *boundary candidate* bc. The boundary candidates BC get updated with the new candidate bc (line 9). This can be done in various ways, such as naively adding them or by considering the set diversity as captured, for instance, by d_{in}, as explained in the following section.

Table 1: The datasets with their respective number of dimensions and number of output categories.

Dataset	Inputs	Included Inputs	Outputs
synth	2	all	binary
titanic	11	6	binary
adult	14	7	binary
heart	13	all	binary
iris	4	all	3
wine	13	all	3
car evaluation	6	all	4

3.2 Boundary Diversification

To obtain a wide and evenly spread-out boundary in the ML-BSA `updateBC` step, we applied a diversification strategy based on maximizing the distances of each boundary candidate bc to its two nearest neighbours bc_{nn1}, bc_{nn2} in BC. Before each round, we mark the candidate $\overline{bc_i}$ with the overall shortest distance sum to its two neighbours, i.e. the one candidate contributing the least diversity to BC_i. We then substitute that candidate by the newly retrieved boundary candidate bc in round $i + 1$ and compare the diversity for both boundary sets in BC_i and BC_{i+1} as in

$$div(BC) = \frac{\sum_{bc}^{BC} d_{in}(bc, bc_{nn1}) + d_{in}(bc, bc_{nn2})}{|BC|}, \tag{3}$$

to keep the set with the candidate, leading to a greater div value. The same d_{in} as for the program derivative can be used here. The version we applied in this study has a fixed size for BC, but one could create more adaptive ways of extending or shrinking the boundary depending on the observed characteristics of the model or data.

3.3 Datasets and preprocessing

The seven datasets used in the study can be found with their total number of inputs, the number of inputs used in this study, and the total number of outputs in Table 1. Apart from the example *synth* dataset, we included six openly accessible datasets commonly used in the machine learning literature.

The search-based approach we used is based on traditional heuristic search. Since search-based heuristics, such as evolutionary algorithms for continuous optimization, require floating point input spaces, some preprocessing was needed. Ordinal values were first converted into integers and then into floating-point equivalents. For instance, the category doors for car in the *car evaluation* dataset was translated from $\{2, 4, more\}$ into $\{1.0, 2.0, 3.0\}$. For reasons of simplicity, we excluded categorical inputs from this study, which reduced two datasets from 11 to 6 (**titanic**) and 13 to 7 inputs or attributes (**adult**). Since the study does not measure or compare model quality (e.g. through accuracy, precision, recall), this choice should not impact the validity of the results. After running ML-BSA, the conversions were - after rounding to the nearest integer - applied in the opposite direction to produce boundary candidates with inputs on the correct scale.

[1] for as long as $cl(i_1) \neq cl(i_2)$, i.e. they are on different sides of the boundary.

Algorithm 1 ML Boundary Spanning (ML-BSA) detects boundary candidates of a classification model using the training data and an objective function - in this paper the program derivative. The program derivative is well suited as the objective function as it zooms-in on the boundary.

```
Input: Classifier $cl$, Training Data $TD$
Output: Boundary candidates $BC$
 1: $BC = \emptyset$
 2: $O = unique\_outputs(TD)$
 3: while stop criterion not reached do
 4:     $o_1, o_2 = sample(O, num = 2, repeat = false)$ # $o_1 \neq o_2$
 5:     $i_1 = rand(TD, o_1)$
 6:     $i_2 = rand(TD, o_2)$
 7:     $c = (i_1, i_2)$ # candidate pair
 8:     $bc = opt\_alg(cl, seed = c)$ # boundary squeeze applying program derivative
 9:     $update(BC, bc)$ # react to new boundary candidate $bc$
10: end while
11: return $BC$
```

3.4 Experiments

We empirically evaluated ML-BSA under the following conditions. As an ML model, we trained a regular decision tree of depth 7 for each dataset, including all data points in TD, for applied ML-BSA in two settings (with and without diversity in search) and measured boundary coverage using nearest neighbour distances in the input space. A decision tree is a traditional machine learning model that decides a class based on traversing the tree from a root to a leaf associated with a class. Its depth defines the longest path from the root to a leaf. We ran ML-BSA ten times on each dataset to receive robust mean and variance statistics representing spread and coverage. As d_{in}, we used Euclidean distance, which is suitable since all inputs are floating points.

We applied differential evolution (DE), a popular evolutionary search strategy [9], to implement the boundary squeeze (line 9 in Algorithm 1). An additional stop criterion for unsuccessful squeezes of 3 seconds per iteration was used to ensure that the program does not proceed indefinitely in some edge cases.

DE was instantiated with a candidate pair c=(i_1, i_2), with $cl(i_1)$=$o_1 \neq cl(i_2)$=o_2, which serves as a genome. Adding uniformly distributed random variance to c in support of δ in all dimensions, an initial population of size ten was created, including c. The phenome for c is $((i_1, o_1), (i_2, o_2))$, for which the program derivative fitness from Formula (2) is calculated. The atomic distance δ_i for each input dimension j in TD is set to $\delta_j = 1e\text{-}3 \times |max(TD_j) - min(TD_j)|$ to adapt to different ranges.

ML-BSA was compared in four configurations - the diversity strategy presence in 3.2 for `updateBC` and with two size limits (10 and 20) on the resulting boundary candidate set. Without diversity inclusion, all detected boundary candidates were added to the boundary up to this boundary size limit, while ML-BSA was run for 2.000 iterations using the diversity strategy. All experimental code and result data are openly available[2].

4 RESULTS

For all seven datasets, boundary candidates could successfully be identified using ML-BSA. Runtimes for the searches were in the seconds range per run - classification is almost instant, and searches converged to boundary candidates within milliseconds. Table 2 shows for each dataset the four configurations and the aggregated respective distances to the two nearest neighbours in terms of mean and standard deviation over the input space. We see that for each dataset, the 2-NN distances are larger for ML-BSA with the diversity strategy over the board. This suggests that the strategy covers a broader range of the boundary. At the same time, the variance is smaller when applying the diversity strategy, which suggests that candidates are more evenly distributed over the boundary, better representing the different areas. Figure 2 offers a $[0, 1]$ normalized view on the distributions of the 2-NN distances per dataset for the $|BC| = 20$ runs to confirm these findings visually.

The fact that the number of desired boundary candidates can be controlled for using ML-BSA, both initially but even dynamically, allows for a tailoring of the search to both search space and classifier. Figures 1 and 3 exemplify how ML-BSA covers the decision boundary (or boundaries) for the synthetic example with varying granularity. While we in this study set static boundary set sizes to make the results comparable, dynamic adjustment of the number and distribution of the candidates to capture *interesting* areas where the potential of failure is larger is one development area for ML-BSA.

With access to a boundary candidate set, we can further calculate its distance to the training data and, by that, potentially use it to inform *confidence* or *uncertainty* of each candidate. This information may even be aggregated as a measure of confidence in the entire boundary. Here, for each candidate, two directions can be considered, materialized through the distance of its input/output pairs to the nearest points on its respective side - with potentially great variation for the two directions. Figure 4 exemplifies this for the *synth* dataset, where each dot represents the distance to the closest training data point on the respective side (red for class *invalid*, green for *valid*) and its size showing the distance as normalized overall candidate distances to their nearest neighbour over the boundary on a log-scale. This highlights that, for instance, some candidates have a clearly further distance to one side of the boundary in direct comparison. Still, even certain areas of the boundary seem less *confident* overall. When prepared and presented in an appropriate

[2]

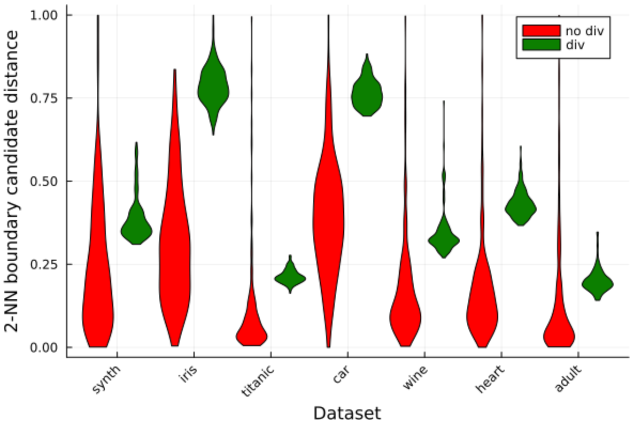

Figure 2: 2-NN distances for the 20 candidate ML-BSA runs comparing optimization with and without diversity. For the latter, points are further apart and more evenly distributed across the boundary, which suggests that it better approximates it.

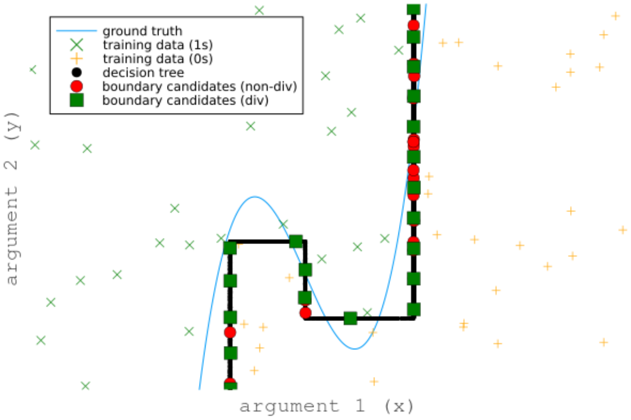

Figure 3: The boundary detection is exemplified for 20 points (as opposed to the 10 points in Figure 1) and presented with and without diversity optimization in search represented by the red and green boundary candidates spread over the boundary. The decision boundary is more evenly covered when diversity is used in the search (green) compared to its approximation through red boundary candidates, for which there are larger gaps that a tester would lack information about when confronted appropriately.

format, this information can be used to steer testers' attention to regions where the boundary lies farthest from the training data.

5 DISCUSSION

Our findings demonstrate that integrating boundary and diversity quantification metrics with search-based optimization enables the identification of varied boundary candidate pairs. This method can delineate the decision boundary of machine learning classifiers, facilitating developers and testers in evaluating the accuracy of the boundary placement and, subsequently, transform selected pairs to tangible test cases.

Table 2: The distances between the two nearest neighbour boundary candidates over ten independent runs. Larger distances indicate a greater space covered, while a smaller variance indicates a more equal distribution among the candidates over the input space.

Dataset	BC	No Div	Div
synth	10	4.31 ± 2.84	**7.56 ± 1.3**
	20	2.54 ± 1.99	**4.13 ± 0.68**
iris	10	2.23 ± 0.79	**5.22 ± 0.22**
	20	1.84 ± 0.78	**3.91 ± 0.24**
titanic	10	98.64 ± 125.92	**186.88 ± 21.0**
	20	54.21 ± 71.16	**112.02 ± 10.29**
car	10	3.61 ± 0.93	**6.08 ± 0.23**
	20	2.95 ± 0.83	**4.76 ± 0.19**
wine	10	229.37 ± 197.14	**360.37 ± 66.29**
	20	113.24 ± 87.3	**215.12 ± 36.06**
heart	10	104.09 ± 56.93	**246.51 ± 17.66**
	20	81.65 ± 50.36	**175.48 ± 14.03**
adult	10	216030 ± 272331	**373263 ± 73218**
	20	118095 ± 152759	**189092 ± 31943**

The use of search-based optimization is crucial as it broadens the applicability of our method. This technique is particularly useful for machine learning models lacking accessible internal derivatives, such as black-box and commercial models, or for models where such derivatives are unfeasible to compute, like decision trees and random forests.

Moreover, our approach is versatile, and can support related use cases. For instance, we've demonstrated how it can be leveraged to pinpoint areas with low boundary confidence, requiring extra, manual attention. Also, at least for low-dimensional spaces, the technique supports visual representation to enhance comprehension of the boundary.

Counterfactual identification and research on adversarial attacks also consider the boundary behavior of ML models but fall short of fully elucidating larger parts of the boundary. They focus on finding particular examples that cross the boundary while ML-BSA (Machine Learning Boundary Spanning Algorithm) provides a methodology and toolset for approximating and spanning larger parts of the decision boundary.

Although visualizing the decision boundary is valuable, it poses challenges in multi-dimensional input spaces or with numerous output classes. Dimensionality reduction techniques may offer a solution, but interpreting their output is often unclear due to the loss of direct mapping to the original features.

As one example, Figure 5 illustrates a 2D dimensionality reduction using UMAP [15][3] of a decision tree model's boundary for the iris dataset, which has 4 dimensions and a ternary output here represented by color. This boundary was exhaustively sampled from the iris training data through over 15,000 iterations of ML-BSA,

[3] Implemented with default UMAP parameters: neighbourhood set at 15 and minimum distance at 0.1, using the Julia library https://github.com/dillondaudert/UMAP.jl.

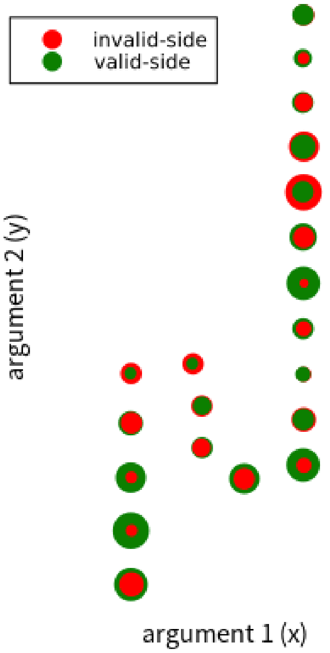

Figure 4: Since boundary candidates contain two input/output pairs with nearby inputs, we can calculate distances to training data points on both sides of the boundary and thereby identify areas of greater *confidence* or *connectedness* between the model and the data. Distances are here normalized over the entire boundary set where larger points signal greater distance to training data, and the colour signals distance into the respective direction. To graphically show both distances to the closest candidates on each side in the same position, the closer one is always plotted on top of the farther. Thus, at times, green is on top of red and vice versa.

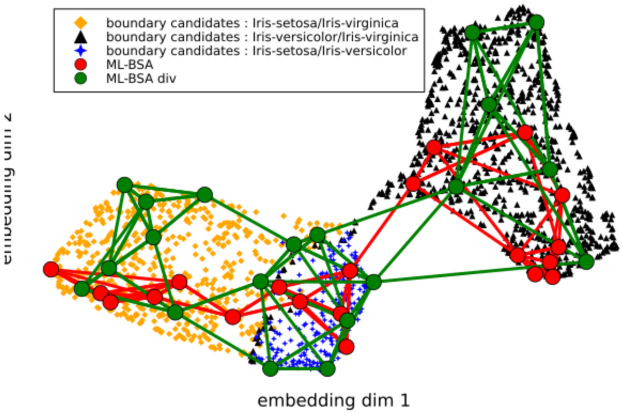

Figure 5: UMAP (dimensionality reduced) visualization of the iris dataset's boundary candidate space. Using exhaustive sampling, three boundaries are differentiated through colour and symbol, seemingly clustered and rather well separated. The larger circles and squares show the ML-BSA derived boundaries. The lines between nearest neighbours show the difference in the area that the solutions span. The plot graphically captures the quantified results, i.e. that diversity-based search has a more equidistant distribution of boundary candidates but is overall inconclusive.

followed by filtering to ensure that boundary candidates were not too closely spaced (d_{in} below 0.3). This process prevents UMAP from overemphasizing areas of high density. The visual representation includes boundary candidates from ML-BSA with (green) and without diversity (red), along with their nearest neighbours, and lines indicating the proximity between nearest neighbours for each boundary candidate.

Upon detailed examination, a greater variation is noticeable among the diversity-optimized boundary candidates. However, the interpretation of this visual representation is not straightforward. For example, the figure seems to show two distinct boundary structures, forming three boundaries where two are adjacent (lower left corner), and the third appears as a line, resembling a boundary itself. Further investigation is required to determine if a wider area covered by a boundary cluster indicates more accurate boundary representation or how the two clusters of boundary candidates correlate with the separation of the three classes.

In contrast, Figure 6 presents a more comprehensible scenario: a decision tree trained on a two-dimensional reduction of the iris dataset. In this figure, the varying quality (broader and more even, for the diversity-based one, in red) of candidate spread is immediately apparent in the input space when comparing the two sets of boundaries. Future research should focus on identifying effective methods for qualitative evaluation of boundary coverage, potentially employing visualization techniques and tools beyond UMAP to facilitate this analysis.

Future research should also evaluate the usefulness of the identified boundary pairs directly with human developers and testers; while a broader and more evenly spread set of boundary pairs has the potential to better inform engineers this needs empirical investigation. Similarly, the type of tabular datasets that we have investigated here might not be typical for the type of machine learning models that are commonly integrated into software systems. Future work should thus investigate also how the boundaries of machine learning models taking more complex inputs, like images and audio, can be addressed with the type of boundary spanning approaches proposed here.

5.1 Limitations

The method is applicable for both traditional and ML software, as it is black-box, not assuming anything from the software under test except for a set of input/output pairs, i.e. labelled training data points in the case of a classifier.

While the execution time of the squeezing (line 8, Algorithm 1) may pose issues for traditional software because of the repeated execution during the search, execution is usually fast for ML, such as deep neural nets, even for complex instances.

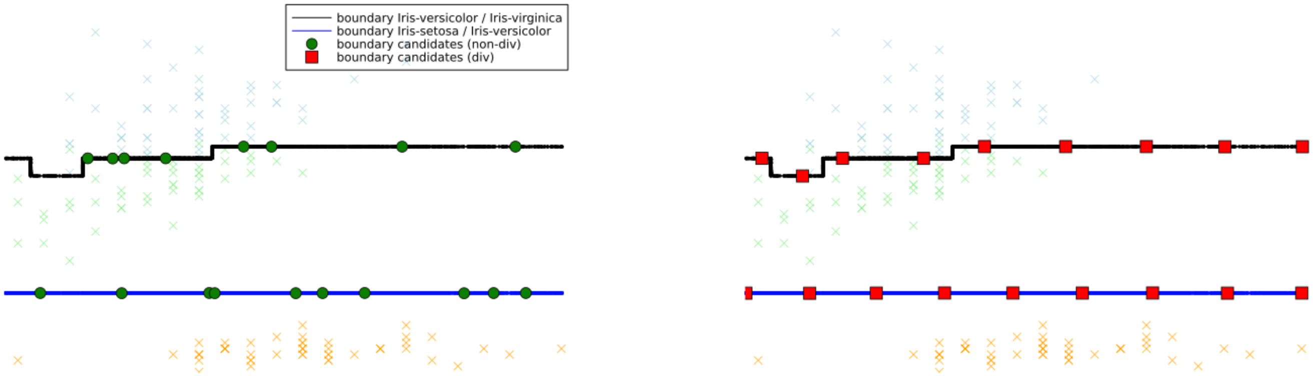

Figure 6: A two-boundary example (PetalWidth, SepalWidth) separating the three species for a decision tree over the iris dataset points, when limiting to two dimensions, is shown in two constellations with and without a diversified boundary. The training data points for actual classes are highlighted in light colors (blue, green, and orange). The boundary candidate set on the left side overall follows a serendipitous distribution. The two decision-tree boundaries are covered evenly in the right version (ML-BSA div), which is a requirement for obtaining representative boundaries. For higher dimensional spaces, the visual analysis is not as clear-cut (see Figure 5).

This study did not include the handling of categorical inputs. However, such support can easily be added, for instance, by employing one-hot encoding.

For multi-class classifications, boundaries tend to be more complex, likely requiring more boundary candidates. How many and whether this could be dynamically identified in response to the different $\binom{n}{2}$ potential boundaries that an n-class problem theoretically has (e.g. between class 1 and 2, as well as class 2 and 3 or 1 and 3 for n=3) has to be empirically evaluated. As identified in previous work for traditional programs, such as Body Mass Index calculation [6], even for ordinal ranges, there is a possibility that the number of actual boundaries found is closer to the theoretical maximum than the intended number of boundaries between ordinal neighbouring classes or categories. As such, a potential use case for ML-BSA would be identifying unintended boundaries.

The idea of a two-edged uncertainty based on boundary candidate distance to either side of the boundary, as discussed in Figure 4, has only been exemplified and not investigated compared to other confidence measures.

6 CONCLUSIONS

In this paper, we automatically identified boundaries for ML classification models, enabled by using the program derivative as the objective function to find boundary candidates. By applying diversity to the search, we successfully covered broader areas with more equidistant coverage of the boundary space. While we successfully quantified this, it was harder to visualize and understand boundaries in higher dimensional spaces - a task for future work.

In contrast to traditional software systems where code responsible for faulty behaviour can usually (easily) be found once revealed, for ML, this is not as straightforward. The creation of the models is not a direct consequence of code but of the learning process that results from a combination of data and code in an opaque way. Helping reveal the boundaries and understanding model weaknesses is thus only a first but necessary step towards fixing faulty behaviour in ML.

REFERENCES

[1] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 291–300.

[2] Panagiotis Bountakas, Apostolis Zarras, Alexios Lekidis, and Christos Xenakis. 2023. Defense strategies for Adversarial Machine Learning: A survey. *Computer Science Review* 49 (2023), 100573.

[3] Dieter Brughmans, Pieter Leyman, and David Martens. 2023. Nice: an algorithm for nearest instance counterfactual explanations. *Data Mining and Knowledge Discovery* (2023), 1–39.

[4] Felix Dobslaw, Francisco Gomes de Oliveira Neto, and Robert Feldt. 2020. Boundary value exploration for software analysis. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 346–353.

[5] Felix Dobslaw and Robert Feldt. 2023. Similarities of Testing Programmed and Learnt Software. In *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 78–81.

[6] Felix Dobslaw, Robert Feldt, and Francisco Gomes de Oliveira Neto. 2023. Automated black-box boundary value detection. *PeerJ Computer Science* 9 (2023), e1625.

[7] Robert Feldt, Francisco G de Oliveira Neto, and Richard Torkar. 2018. Ways of applying artificial intelligence in software engineering. In *Proceedings of the 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*. 35–41.

[8] Robert Feldt and Felix Dobslaw. 2019. Towards automated boundary value testing with program derivatives and search. In *Search-Based Software Engineering: 11th International Symposium, SSBSE 2019, Tallinn, Estonia, August 31–September 1, 2019, Proceedings 11*. Springer, 155–163.

[9] Vitaliy Feoktistov. 2006. *Differential evolution*. Springer.

[10] Riccardo Guidotti. 2022. Counterfactual explanations and how to find them: literature review and benchmarking. *Data Mining and Knowledge Discovery* (2022), 1–55.

[11] Shalmali Joshi, Oluwasanmi Koyejo, Been Kim, and Joydeep Ghosh. 2018. xgems: Generating examplars to explain black-box models. *arXiv preprint arXiv:1806.08867* (2018).

[12] Thibault Laugel, Adulam Jeyasothy, Marie-Jeanne Lesot, Christophe Marsala, and Marcin Detyniecki. 2023. Achieving Diversity in Counterfactual Explanations: a Review and Discussion. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*. 1859–1869.

[13] Wei Ma, Mike Papadakis, Anestis Tsakmalis, Maxime Cordy, and Yves Le Traon. 2021. Test selection for deep learning systems. *ACM Transactions on Software*

Engineering and Methodology (TOSEM) 30, 2 (2021), 1–22.

[14] Bogdan Marculescu and Robert Feldt. 2018. Finding a boundary between valid and invalid regions of the input space. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 169–178.

[15] Leland McInnes, John Healy, and James Melville. 2018. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426* (2018).

[16] Ramaravind K Mothilal, Amit Sharma, and Chenhao Tan. 2020. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the 2020 conference on fairness, accountability, and transparency*. 607–617.

[17] Inioluwa Deborah Raji, Andrew Smart, Rebecca N White, Margaret Mitchell, Timnit Gebru, Ben Hutchinson, Jamila Smith-Loud, Daniel Theron, and Parker Barnes. 2020. Closing the AI accountability gap: Defining an end-to-end framework for internal algorithmic auditing. In *Proceedings of the 2020 conference on fairness, accountability, and transparency*. 33–44.

[18] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence* 1, 5 (2019), 206–215.

[19] Sahil Verma, Varich Boonsanong, Minh Hoang, Keegan E Hines, John P Dickerson, and Chirag Shah. 2020. Counterfactual explanations and algorithmic recourses for machine learning: A review. *arXiv preprint arXiv:2010.10596* (2020).

[20] Rey Reza Wiyatno, Anqi Xu, Ousmane Dia, and Archy De Berker. 2019. Adversarial examples in modern machine learning: A review. *arXiv preprint arXiv:1911.05268* (2019).