On the Comparison of Learned Classifiers

Soaibuzzaman[0000−0002−8971−5904], Jenny Döring, Srinivasulu Kasi, and Jan Oliver Ringert[0000−0002−3610−3920]

Bauhaus-University Weimar, Germany

Abstract. Machine learning for classification has seen numerous applications to complex, real-world tasks. Learned classifiers have become important artifacts of software systems that, like code, require careful analyses, maintenance, and evolution. Existing work on the formal verification of learned classifiers has mainly focused on the properties of individual classifiers, e.g., safety, fairness, or robustness, but not on analyzing the commonalities and differences of multiple classifiers.
We present MLDiff, a novel approach to comparing learned classifiers based on querying agreements and disagreements between classifications, where one classifier is an alternative or variant of another. We present a prototypical implementation that leverages an encoding to SMT and can discover differences not (yet) seen in available datasets. Our prototype implements MLDiff for any combination of Decision Trees, Linear Support Vector Classification, Logistic Regression Classification, and Neural Networks. We evaluate it on classifiers trained on popular datasets in terms of performance and effectiveness of the analysis to discover disagreements between classifiers.

Keywords: machine learning, comparison, SMT

1 Introduction

Machine learning for classification has seen numerous applications to complex, real-world tasks. Existing work on the formal verification of learned classifiers has mainly focused on the verification of properties, e.g., safety [19,20], fairness [18,42,33,32] or robustness [30,10,2], of individual classifiers but not on the commonalities and differences of multiple classifiers. Learned classifiers become important artifacts of software systems that, like code, require careful analyses, maintenance, and evolution. Typically, alternative or evolved classifiers are compared by individual metrics on known datasets or on variants of the above properties. While these are meaningful and important, they do not provide comparison in terms of disagreements or common classifications.

We present MLDiff, a novel approach to directly comparing learned classifiers where one is an alternative or variant of another. The definifition of MLDiff does not rely on existing datasets or benchmarks and thus covers differences not (yet) seen in available data. It can also show the absence of differences. Our motivation for developing MLDiff is to assist users in comprehending the reasons behind the counter-intuitive properties [39] and disagreements between classifiers. Typical use cases include checking disagreements of classifiers or uncovering safety-critical differences, e.g., generating and browsing instances that witness (mis-)classifications. We discuss example characteristics of MLDiff for querying a combination of classifiers on common features and adding feature constraints in Sect. 4.

We have implemented a prototype of MLDiff (based on SMT [24,21] encodings, see Sect. 5) for combinations of Decision Trees (DT), Linear Support Vector Classification (SVM), Logistic Regression Classification, and Neural Networks. We evaluate MLDiff and our prototype implementation on classifiers trained on popular datasets in Sect. 6. Our implementation is available from [38].

2 Example

Consider the well-known dataset of Iris flowers [14] for classifying flowers by their petal and sepal length and width (four features) into three species. A team of engineers has trained the SVM and the Decision Tree (DT) classifiers shown in Fig. 1 [top] and [left] for evaluation. A traditional comparison reveals an accuracy of 96% for both classifiers and an F1-score of 96% for the DT and 93% for the SVM, making the DT the preferred choice.

However, an MLDiff analysis shows that the DT sometimes classifies instances as Virginica species (used for medical purposes) when the SVM classifies them as the poisonous Versicolor [12]. This disagreement (shown in Fig. 1 [right]) and potentially serious misclassification alarms the engineers. They consult a horticulturist for clarification before deploying either classifier. Note that no such instance is contained in the dataset, i.e., even an exhaustive search through the dataset would not have revealed the potentially dangerous disagreement.

As another example, a team of engineers is working on a facial recognition system for lab access. Maintaining the existing Neural Network (Multi-Layer Perceptron) classifier has become costly as the number of target faces grows (classes: employee IDs). Alternatively, the team investigates a binary Logistic Regression (LogReg) classifier (classes: `access`/`no access`). The existing MLP classifier is trained to classify employees with IDs $0-9$, e.g., the face in Fig. 2 [left] has ID/class 8[1], while the candidate LogReg classifier is trained on more employees (IDs $0-15$) where IDs $0-5$ have access, e.g., the face shown in Fig. 2 [right], and IDs $6-15$ do not.

One question of the engineers is whether the new LogReg classifier would give access to any employee with an ID other than 0-5 when checked against the existing and trusted MLP classifier. MLDiff finds such a case as shown in Fig. 2 [middle] and the team decides to stay with the more fine-grained MLP classifier. Note that MLDiff does not prove any quality properties of the MLP classifier, but rather uncovers potentially interesting differences of the classifiers. Also note that the face in Fig. 2 [middle] does not exist in the dataset and is

[1] The MLP and LogReg classifiers are trained on the Olivetti Faces dataset [23]

$$argMax\left(\begin{pmatrix} 0.21443255 & 0.38513516 & -0.79414658 & -0.44987553 \\ -0.11326783 & -0.59086647 & 0.84205491 & -1.65158089 \\ -0.75651555 & -0.91195898 & 1.08941819 & 1.75014748 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 0.10703851 \\ 0.97002258 \\ -1.01034086 \end{pmatrix}\right)$$

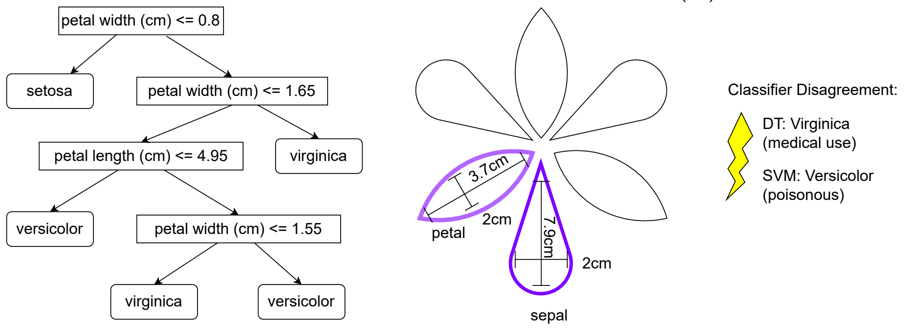

Fig. 1. A learned SVM (Linear Support Vector Classifier) [top], a learned decision tree [left] (both for the Iris dataset), and an analysis result computed by MLDiff predicted as Virginica (medical use) by the DT and as Versicolor (poisonous) by the SVM.

generated by MLDiff as a witness for an analyzed difference (details on the MLP classifier and a PCA-based reduction are provided in Sect. 6.1).

3 Background

We present background on classifiers that assign classes to vectors of feature values (instances). Our presentation applies some pragmatic simplifications, e.g., features are all Real-valued. Common encoding techniques [16] can support other feature domains, e.g., categorial or integer-valued.

We denote classes by $c \in C$, e.g., `versicolor` $\in C$, features by $x \in X$, e.g., `petal length` $\in X$, and valuations of features as instances $d \in D = \mathbb{R}^{|X|}$. For instance $d \in D$ we denote the value of feature $x \in X$ by $d(x) \in \mathbb{R}$. As an example, the instance d from Fig. 1 [right] has $d($`petal length`$) = 3.7$.

We consider classifiers as total functions that map instances to classes as stated in Def. 1.

Definition 1 (Classifier). *A classifier over features X is a total function $cl : \mathbb{R}^{|X|} \to C$ that maps each instance $d \in D = \mathbb{R}^{|X|}$ to a class $c \in C$.*

Variants of this most basic definition may exist, e.g., classifiers as partial functions or classifiers as relations, that could also be supported by MLDiff. It is easy to check that the classifiers from Sect. 5 Defs. 4-7 map to Def. 1.

4 Classifier Comparison

We now introduce the conceptual framework MLDiff and present one approach for its implementation in Sect. 5.1. The goal of MLDiff is to provide deeper in-

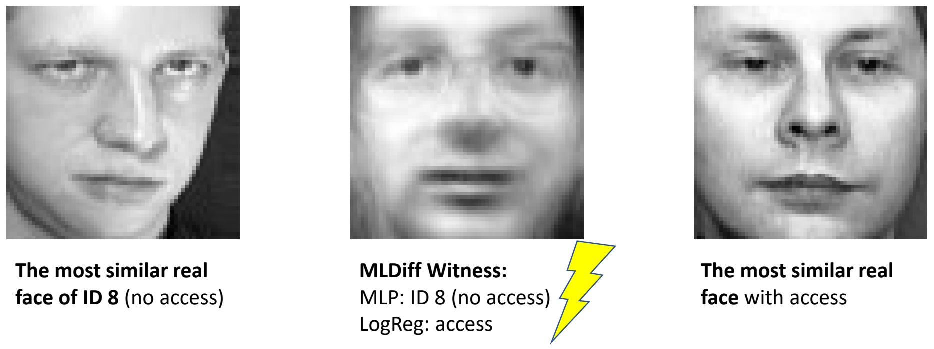
The most similar real face of ID 8 (no access)
MLDiff Witness:
MLP: ID 8 (no access)
LogReg: access
The most similar real face with access

Fig. 2. A conflict of an MLP and a LogReg classifier detected by MLDiff [middle] where the LogReg classifier grants access when the MLP classifier would deny it based on the determined employee ID 8. Closest representatives of the dataset from both classes are shown [left] and [right] from the Olivetti Faces dataset [23] for illustration purposes

sight into the differences between classifiers. Def. 2 presents the MLDiff classifier combination we use for analyses of differences of classifiers.

Definition 2 (MLDiff Classifier Combination). *Given classifier* $cl_1 : \mathbb{R}^{|X_1|} \to C_1$ *and classifier* $cl_2 : \mathbb{R}^{|X_2|} \to C_2$ *we construct the MLDiff classifier combination* $cl_1 \oplus cl_2 : \mathbb{R}^{|X_1 \cup X_2|} \to C_1 \times C_2$ *such that*

$$\forall d \in \mathbb{R}^{|X_1 \cup X_2|} : \quad cl_1 \oplus cl_2(d) = cl_1(d|_{X_1}) \times cl_2(d|_{X_2})$$

where $d|_{X_i}$ *projects* $\mathbb{R}^{|X_1 \cup X_2|}$ *to* $\mathbb{R}^{|X_i|}$ *with corresponding values for features* X_i*.*

Note that if the two classifiers are total functions as in Def. 1 then also their MLDiff combination from Def. 2 is a total function. Cases where features $X_1 = X_2$ are very common when using classifiers on datasets of the same structure. Cases where $X_1 \cap X_2 = \emptyset$, i.e., both classifiers have completely independent input, are meaningful in cases where instances $d_1 \in \mathbb{R}^{|X_1|}$ can be transformed to instances in $d_2 \in \mathbb{R}^{|X_2|}$, e.g., when classifying images with or without PCA pre-transformation (variant of second example in Sect. 2), but would require more complex queries.

Note that common features are shared between classifiers, while classes and classifications are always kept separate. For the two classifiers in Fig. 1 of the Iris dataset we have $X_1 = X_2$ and $C_1 = C_2 = \{\texttt{setosa}, \texttt{virginica}, \texttt{versicolor}\}$. For d as in Fig. 1 [right] we have $cl_1 \oplus cl_2(d) = (\texttt{virginica}, \texttt{versicolor})$, i.e., applied to the same element d, the classifiers disagree. In the second example in Sect. 2 we have $X_1 = X_2$ but $\{0..9\} = C_1 \neq C_2 = \{\texttt{access}, \texttt{no access}\}$. For d as in Fig. 2 [middle] we have $cl_1 \oplus cl_2(d) = (8, \texttt{access})$.

4.1 Use Cases and Queries

We envision various and flexible use cases of MLDiff for the comparison of learned ML models. We believe that generating individual counterexamples is useful in safety- or interpretability-critical contexts where ML models require rapid evaluation and iteration, e.g., in certification, debugging, or auditing scenarios. Analyses may be customized using queries, i.e., assertions over feature variables and predicted classes. Some common example queries are about classifier disagreement encoded by the query $c_1 \neq c_2$ where $(c_1, c_2) = cl_1 \oplus cl_2(d)$ (used in the first example in Sect. 2 and Fig. 1) or common classification results encoded by the query $c_1 = c_2$. An example for a custom query on classifier predictions is $c_1 = 1 \wedge c_2 > 5$ as used in the second example in Sect. 2 and Fig. 2. Queries may also be extended with constraints on feature values (see Sect. 4.2). Given a classifier combination and a query, MLDiff generates a satisfying instance as defined in Def. 3. Note that we do not fix the query language of MLDiff. Different implementations may use different logics, e.g., our prototype in Sect. 5 uses First-Order Logic queries (SMT).

Definition 3 (MLDiff Result). *Given a classifier combination (Def. 2) and a query φ over d, c_1, and c_2, an MLDiff result is a valuation of d, c_1, and c_2 that satisfies φ and $(c_1, c_2) = cl_1 \oplus cl_2(d)$.*

4.2 Feature Constraints

Consider an analysis of the two Iris classifiers from Fig. 1 for agreements. MLDiff could produce a witness instance d with d(`sepal length`) $= -3$. While mathematically correct, one would likely consider this an undesired witness, as the length cannot be negative. We have observed these and similar cases in manual experiments with our prototypical implementation of MLDiff (see Sect. 5).

To prevent these undesired witnesses, we extend MLDiff to take feature constraints into account. Some examples of feature constraints supported by our prototypical implementation of MLDiff are upper and lower bounds, restrictions of feature values to `Int` or `Real` values, and support for categorial features. All these feature constraints easily translate to assertions on a single feature variable in SMT.

5 MLDiff prototype

We present a prototypical implementation of MLDiff (available from [38]). The implementation is designed for transparency and for supporting most features of MLDiff as presented in Sect. 4. Many alternative implementations with different properties in terms of performance or completeness are possible.

```
1 (declare-const x0 Real) ; one constant for each feature
2 ; ...
3 (declare-const xn Real)
4 (declare-const cls1 Int) ; predicted class ID of first classifier
5 (declare-const cls2 Int) ; predicted class ID of second classifier
6 ; assertion for classifier 1 relating x1..xn to cls1
7 ; assertion for classifier 2 relating x1..xn to cls2
8 (assert (not (= cls1 cls2))) ; example query for disagreement of classifiers
```

Listing 1. Generic encoding of the MLDiff classifier combination from Def. 2 into an SMT module with an example query for classifier disagreements in line 8

5.1 Realizing Classifier Comparisons

Our implementation of MLDiff is based on a translation of the classifiers to SMT formulas. A sketch of a generic SMT problem representing the classifier combination from Def. 2 is shown in Lst. 1. Shared `Real`-valued feature variables are declared and used in both classifier translations (Lst. 1, ll. 1-3). The independently determined classification results `cls1` of the first classifier and `cls2` of the second are encoded as `Int` variables (Lst. 1, ll. 4-5) and used to query for particular combinations, e.g., disagreement (Lst. 1, ll. 8). Generated assertions (Lst. 1, l. 6 and l. 7) determine classification results by feature variable valuations in translations specific to each classifier kind as described in Sect. 5.2.

5.2 Translation to SMT

Our notation of the translation combines the structure of each classifier as found similarly in many textbooks [3,16] and pseudo code of generated SMT formulas. We indicate operators and values in SMT formulas by double-underlining them in the algorithms. $\underline{\underline{\texttt{And(}}}$ ... $\underline{\underline{\texttt{)}}}$ represents a conjunction over arguments, $\underline{\underline{\texttt{Or(}}}$ ... $\underline{\underline{\texttt{)}}}$ a disjuntion, and $\underline{\underline{\texttt{Ite(}b\texttt{, }v_1\texttt{, }v_2\texttt{)}}}$ the if-then-else operator. The following Algs. 1-4 can be read as templates where executing control structures, e.g., for-loops, generates SMT API calls to instantiate SMT formulas. For generality, the algorithms use class variable `cls` for either `cls1` or `cls2` in Lst. 1.

Decision Tree Classifier Intuitively, our translation of decision tree classifiers as defined in Def. 4 to SMT, shown in Alg. 1, lists for each class $c \in C$ (l. 1) all conditions of each path from the root to a leaf of c (ll. 2-6) and states that the satisfaction of the conditions determines the classification (l. 8). Applied to the example DT from Fig. 1 Alg. 1 generates the SMT API calls shown in Lst. 2. Note that class `setosa` has a single path (Lst. 2, l. 2) and that class `virginica` has a longest path in the DT (Lst. 2, l. 5) with four decisions.

Definition 4 (Decision Tree). *A decision tree T for features X and classes C is a finite tree with a distinguished root node. Non-leaf nodes $t \in T$ are assigned $D(t) \subseteq D = \mathbb{R}^{|X|}$ and a partition of $D(t)$ to children of t. Leaf nodes of T are assigned a class from C. $D(r) = D$ iff $r \in T$ is the root node.*

Algorithm 1 Translation of a DT from Def. 4 to SMT

```
1: procedure ENCODEDT(decision tree $T$ for classes $C$ and features $X$)
2:   And( for $c \in C$ do          ▷ conjunction over all classes, selecting all paths in $T$
3:     Or( for path $\pi_c \in T$ starting in the root and ending in a leaf with class $c$ do
4:       And( for consecutive nodes $(t, t') \in \pi_c$ do ▷ conjunction over path elements
5:         $x(t)$ <= $v(t)$ if $t'$ left child of $t$
6:         $x(t)$ > $v(t)$ if $t'$ right child of $t$
7:       end for )                            ▷ conjunction of path elements
8:     end for ) == (cls == $c$)        ▷ disjunction over all paths determins class
9:   end for )                                ▷ conjunction over all classes
```

```
And( % listing classes with all their paths through the DT
  Or(And(x3 <= 0.8)) == (cls == 0), % setosa (only one path of length one)
  Or(And(x3 > 0.8, x3 <= 1.65, x2 > 4.95, x3 > 1.55),
     And(x3 > 0.8, x3 <= 1.65, x2 <= 4.95)) == (cls == 1), % versicolor
  Or(And(x3 > 0.8, x3 <= 1.65, x2 > 4.95, x3 <= 1.55),
     And(x3 > 0.8, x3 > 1.65)) == (cls == 2)) % virginica
```

Listing 2. Translation of the DT classifier from Fig. 1 into SMT API calls

Linear Support Vector Classifier A Linear Support Vector Machine (SVM) separates instances for classification by hyperplanes. Its elements are shown in Def. 5. Our translation of SVMs closely follows their structure provided in Def. 5 as shown in Alg. 2 where the dot product of the feature vector and the coefficient matrix W is calculated. The bias vector b is added in line 6 for each class. The dot product only uses linear arithmetic and generates one sum for each class as parameter for the $argMax$ encoding procedure shown in Alg. 3 (called in Alg. 2, l. 2). Applied to the example SVM from Fig. 1 Alg. 2 generates SMT API calls shown in Lst. 3 (coefficients rounded to 5 decimals).

The encoding of $argMax$ iterates over all classes (Alg. 3, l. 2) and exhaustively determines the classification result `cls`. Note the inner nested loop and different comparison operators in l. 4 and l. 5 to ensure that in case of multiple classes with the same scores, the one with the lowest index is determined (see also the validation in Sect. 6.2). Applied to the example SVM from Fig. 1 Alg. 3 generates the SMT API calls shown in Lst. 4 where the scores `s0`-`s2` (parameters of Alg. 3) correspond to those generated in Lst. 3 by Alg. 2, e.g., score `s0` corresponds to the expression in Lst. 3, l. 2.

Definition 5 (Linear Support Vector Classifier). *A Linear Support Vector Classifier for features X and classes C consists of a coefficient matrix $W \in \mathbb{R}^{|C| \times |X|}$ and bias vector $b \in \mathbb{R}^{|C|}$ describing $|C|$ hyperplanes.*

Many implementations, e.g., scikit-learn [28], simplify the representation for binary classifiers. The coefficient matrix $W \in \mathbb{R}^{|C| \times |X|}$ is replaced by a simple vector $w \in \mathbb{R}^{|X|}$ and $argMax$ is simply a comparison for a positive value. Our implementation (not shown in Alg. 3) also handles this special case by encoding this comparison with an if-then-else operator.

Algorithm 2 Translation of a SVM from Def. 5 to SMT

```
1: procedure ENCODESVM(coeffs. $w(c, x)$ for classes $C$ and features $X$)
2:     ENCODEARGMAX (              ▷ call to encode argMax of values for each class
3:       for $c \in C$ do       ▷ calculate dot product by coefficient vectors for each class
4:         Sum( for feature $x \in X$ and corresponding coefficient $w(c, x)$ of $W$ do
5:           x * $w(c, x)$
6:         end for ) + $b(c)$                 ▷ close parenthesis for sum and add bias
7:       end for )                             ▷ end of argMax parameters (sums)
```

```
encodeArgMax( % partial code generated for parameters
  Sum(x0 * 0.21443, x1 * 0.38514, x2 * -0.79415, x3 * -0.44988) + 0.10704,
  Sum(x0 * -0.11327, x1 * -0.59087, x2 * 0.84205, x3 * -1.65158) + 0.97002,
  Sum(x0 * -0.75652, x1 * -0.91196, x2 * 1.08942, x3 * 1.75015) + -1.01034)
```

Listing 3. Excerpt of translation of the SVM classifier from Fig. 1 into SMT API calls (showing code generated from Alg. 2, ll. 3-6) used as parameters for Alg. 3

Algorithm 3 Helper function *argMax* in SMT

```
1: procedure ENCODEARGMAX(scores $s$ for classes $C$)
2:     And( for $c \in C$ do                              ▷ class with highest score
3:       (cls == c) == And( for $c' \in C, c' \neq c$ do        ▷ score greater or equal
4:         $s(c)$ > $s(c')$ if index of $c'$ lower than index of $c$
5:         $s(c)$ >= $s(c')$ if index of $c'$ higher than index of $c$
6:       end for )                  ▷ end conjunction of scores greater than others
7:     end for )                          ▷ end conjunction over all classes
```

```
And((cls = 0) == And(s0 >= s1, s0 >= s2), % setosa
    (cls = 1) == And(s1 > s0, s1 >= s2), % versicolor
    (cls = 2) == And(s2 > s0, s2 > s1)) % virginica
```

Listing 4. Excerpt of code generated from Alg. 3 for the SVM classifier from Fig. 1 into SMT API calls (scores `s0-s2` are expressions shown in Lst. 3)

Logistic Regression Classification Note the structural correspondence between SVMs (Def. 5) and Logistic Regression Classifiers in Def. 6. The class index of an instance $d \in \mathbb{R}^{|X|}$ is $argMax(\frac{1}{1+e^{-(W \cdot d + b)}})$, where the logistic function $\frac{1}{1+e^{-x}}$ is applied to each component of the product $W \cdot d + b$. As the logistic function is monotonic, omitting it before applying $argMax$ in Alg. 3 has no effect on the determined class. The SMT translation of Logistic Regression classifiers from Def. 6 is thus identical to Alg. 2.

Definition 6 (Logistic Regression Classifier). *A Logistic Regression Classifier for features X and classes C consists of a coefficient matrix $W \in \mathbb{R}^{|C| \times |X|}$ and bias vector $b \in \mathbb{R}^{|C|}$ describing $|C|$ hyperplanes.*

Multi Layer Perceptron The most complex model translated by our prototype is the Multilayer Perceptron (MLP), a feedforward artificial neural network with connected neurons in at least three layers (input-, hidden-, and output-layer) with elements shown in Def. 7. The class index of an instance $d \in \mathbb{R}^{|X|}$ is then determined by $argMax(a_k(W_k \cdot (\ ...\ a_0(W_0 \cdot d + b_0)) + b_k))$. A common activation function for hidden layers is the rectified linear unit (ReLU) function $relu(x) = max(0, x)$. Our translation shown in Alg. 4 computes the activation scores of each neuron j in each layer i (ll. 4-8). The procedure applyActivation (Alg. 4, l. 9) is not further detailed here, it encodes activation functions, e.g., the ReLU function on score s as `Ite(s >= 0, s, 0)`. The assignment to *activation* in Alg. 4, l. 9 and the outer loop nest calculations to propagate activation values of the output layer for use in Alg. 4, l. 11.

Definition 7 (Multi-Layer Perceptron). *A Multi-Layer Perceptron Classifier for features X, classes C, and hidden layer sizes $h_i \in \mathbb{N}$ of layer $i = 1..k$ consists of input layer weights $W_0 \in \mathbb{R}^{|X| \times h_1}$, hidden layer weights $W_i \in \mathbb{R}^{h_i \times h_{i+1}}$ for $i < k$, output layer weights $W_k \in \mathbb{R}^{h_k \times |C|}$, corresponding biases $b_{i=0..k}$, and activation functions $a_{i=0..k}$ for each layer type.*

Algorithm 4 Translation of an MLP from Def. 7 to SMT

```
 1: procedure encodeMlp(MLP layers for classes $C$ and features $X$,)
 2:     $activation \leftarrow X$                                ▷ start with feature values/variables
 3:     for layer $i$ in $0..k$ do                               ▷ calculate activation layer by layer
 4:       $scores \leftarrow$ for neuron $j$ in $0..h_{i+1}$ ($i = k$: $0..|C|$) do   ▷ neurons in layer $i+1$
 5:         Sum( for $a \in activation$ and corresponding coefficient $w_i(j, a)$ of $W_i$ do
 6:           $a$ * $w_i(j, a)$
 7:         end for ) + $b_i(j)$                             ▷ add bias of neuron input weights to sum
 8:       end for                                            ▷ end of calculation of scores
 9:       $activation \leftarrow$ applyActivation($scores$)    ▷ input, hidden, or output activ.
10:     end for                                              ▷ propagated all activations to last output layer
11:     encodeArgMax($activation$)                           ▷ see Alg. 3
```

5.3 Limitations of the Prototype

Our current prototype is based on the Z3 SMT-solver [24,43] and as such shares limitations related to more complex arithmetic functions, e.g., it currently does not support non-linear kernels for SVMs nor activation functions for the hidden layers of the MLP other than ReLU and identity.

6 Evaluation

We evaluate our prototypical implementation of MLDiff on combinations of datasets and all classifier kinds from Sect. 5.1, trained as described in Sect. 6.1. Our research questions investigate effectiveness as numbers of disagreements founds and the efficiency of the implementation and its extensions:

– RQ1: How many disagreements does our MLDiff comparison find?
– RQ2: What is the analysis cost for different classifiers?
– RQ3: What is the overhead of adding feature constraints?

All experiments were conducted on a standard desktop computer with an Intel i7-7700K@4.2GHz CPU and 32 GB DDR4 2667 MHz RAM. We used the Z3 SMT-solver [24,43] Python API to run all analyses with a 1-hour timeout.

6.1 Datasets, Training, and Classifiers

We base our evaluation on four datasets with different characteristics sourced from scikit-learn [28]: Iris dataset [14], Digits dataset [1], Breast Cancer dataset [45], and Olivetti Faces dataset [23]. The Iris dataset [14] comprises 150 samples of iris flowers, each characterized by four real-valued, positive features: sepal length, sepal width, petal length, and petal width. The Digits dataset [1] includes 1,797 8x8 pixel images depicting handwritten digits, with features represented by integers ranging from 0 to 16. The total samples are distributed among 10 classes with around 180 samples per class. The Breast Cancer dataset [45] provides diagnostic features computed from digitized images of breast cancer biopsies, with 30 real-valued features for each of the 569 samples, serving as a benchmark for binary classification of tumors as malignant or benign. Lastly, the Olivetti Faces dataset [23] encompasses 400 64x64 pixel grayscale images (features/pixels range from 0 to 1) of faces, capturing 40 different individuals (used in Sect. 2).

We trained Decision Tree (DT), linear Support Vector Machine (SVM), Logistic Regression, and Multi-Layer Perceptron (MLP) classifiers using the scikit-learn [28] library in Python. The DT, SVM, Logistic Regression, and MLP were trained with default parameters. The MLP was executed with a maximum of 100 iterations. A single hidden layer configuration with 10 neurons was allocated for the Digits and Breast Cancer datasets, while a single layer comprising 20 neurons was used for the Iris and Olivetti Faces datasets.

We partitioned each dataset into training and testing subsets using the conventional 80-20 ratio to evaluate the models. Due to the high numbers of features

	Accuracy (%)				F1-score (%)			
	i	d	o	c	i	d	o	c
DT	100.00	86.94	65.00	94.74	1	0.97	0.93	0.99
SVM	100.00	96.39	100.00	93.86	0.98	0.99	1	0.92
LOGREG	100.00	96.94	100.00	95.61	0.98	0.99	1	0.95
MLP	83.33	92.22	75.00	96.49	0.88	0.95	0.74	0.92

Table 1. Accuracy and F1-score of classifiers on datasets (in columns): Iris (i), Digits (d), Olivetti Faces (o), and Breast Cancer (c)

and classes of the Olivetti Faces dataset, our evaluation used 10 classes (0..9) and employed Principal Component Analysis (PCA) to reduce input dimensionality to 24 components. The accuracy and F1-score metrics across all classifiers and datasets are presented in Table 1, showing that most classifiers performed well with accuracy rates exceeding 90% and F1-scores approximating unity. As an exception, the Decision Tree classifier applied to the Olivetti Faces dataset exhibited relatively diminished accuracy levels.

6.2 Validation

All implementations may have bugs and we decided to validate our work against different checks of correctness and completeness.

First, we have validated the correctness of the SMT translation of classifiers by encoding each instance of the dataset into SMT (one at a time) and extracting the predicted class from the SMT model for comparison against the prediction of the original classifier.

Note that after the encoding of the classifier and an instance, a model should always exist as the classifier is a total function. Our initial validation encoded the predicted class and checked for satisfiability, but this check is weaker as it would even be satisfied by a module with no assertions. We also experimented with a simpler encoding of $argMax$, stating that the determined class is that of the maximum score. However, this simpler encoding leads to the SMT solver picking a class if scores are tied (although correct, it fails validation against the scikit-learn [28] variant that picks the lowest class indices as our Alg. 3).

Second, in addition to encoding instances in SMT we tested the extraction of instances, i.e., the analysis results/witnesses. We exhaustively asked the solver to produce all disagreements for unique class combinations of either classifier. Each disagreement was extracted and given to the original scikit-learn [28] classifiers for confirmation in Python.

This second validation discovered interesting behavior of the SMT solver. The solver computed spurious instances for some combinations of classifiers involving SVM, Logistic Regression and MLP Classifiers. Our translation uses `Real`-valued variables with unlimited precision in Z3 [24,43]. The scikit-learn implementation in Python uses `float64` numbers. The SMT solver often computes models close to the decision boundaries that fail to be reproduced in Python due to rounding errors. We call these instances *spurious instances* although the analysis in SMT is correct. We have analyzed various spurious instances manually, e.g., a feature value conversion from `Real` to `float64` introduced a 10^{-16} rounding error that after matrix multiplication in an SVM classifier was significant enough to change the *argMax* result. Issues with accuracy of analyses and rounding have been observed by others in literature [22,25]. We leave further investigation of this issue to future work.

6.3 Results

To answer **RQ1** for different combinations of classifiers, we combined all pairs of classifiers across all four datasets to exhaustively compute disagreements per classification outcomes. We calculated the ratio of found disagreements to possible disagreements, where possible disagreements are equal to the number of classes squared minus the number of classes (where both agree).

Interestingly, MLDiff was able to find **disagreements between all classifiers for all combinations of classes**, i.e., for any combination we could compute an instance that forces the disagreement. We compared this surprisingly high number with disagreements one could find by using only the existing datasets: For half of the datasets 26% (median disagreement ratio of 0.26) or more disagreements can be found only relying on existing elements. However, when excluding the breast cancer dataset, which is a binary classification (and thus has only 2 possible disagreements), the distribution of disagreements is 0.17 for the median, and 0.39 for the third quartile (Q3).

To address **RQ2** we investigate the analysis cost of finding disagreements using MLDiff across all pairs of classifiers (16 pairs) and datasets (64 combinations). Table 2 shows times in seconds for computing all differences (number depends on dataset, but not on classifiers as established in answer to RQ1) between pairs of classifiers on all four datasets: Iris (i), Digits (d), Olivetti Faces (o), and Breast Cancer (c). For Decision Tree (DT), Linear Support Vector Classifier (SVM), and Logistic Regression (LOGREG) classifiers, the analysis cost was quite low, ranging from 0.01 to 24.31 seconds. However, the analysis cost was significantly higher (up to 1382.27 seconds) for the Multilayer Perceptron (MLP), particularly for datasets with high dimensionality and many classes, such as Digits and Olivetti Faces. Conversely, datasets with lower dimensions and fewer classes, like Iris and Breast Cancer, remain fast for these classifiers.

Note that the order of the assertions encoded into the Z3 solver [24,43] has a significant impact on the analysis cost. For instance, the cost of identifying differences between the MLP and SVM on the digits dataset was 242.46 seconds. However, the cost of identifying differences between the SVM and MLP on the same dataset was 1382.27 seconds. Technically, these are identical problems. We confirmed these surprisingly different times by repeating both measurements ten times, with a standard deviation of 2.13 and 8.75, respectively. We believe that this difference is due to internal SMT solver heuristics and instabilities [5] that are known to be sensitive to the problem encoding.

To answer **RQ3** about the overhead of adding feature constraints – as described in Sect. 4.2 – we repeated the experiments for all previous combinations

	DT				SVM				LOGREG				MLP			
	i	d	o	c	i	d	o	c	i	d	o	c	i	d	o	c
DT	0.01	0.01	0.01	0.00	0.02	1.80	1.02	0.03	0.02	2.75	1.32	0.03	6.03	247.90	171.97	0.69
SVM	0.02	2.31	1.00	0.02	0.01	0.06	0.02	0.01	0.02	10.89	5.32	0.03	8.19	**1382.27**	833.39	0.65
LOGREG	0.02	2.74	0.98	0.02	0.02	24.31	7.89	0.03	0.00	0.06	0.02	0.01	7.20	1133.86	838.69	0.56
MLP	7.06	97.54	242.46	0.32	8.03	**254.06**	1009.80	0.44	8.49	324.44	1180.57	0.58	0.01	0.03	0.02	0.01

Table 2. RQ2 analysis time (in seconds) of finding all differences of pairs of classifiers on datasets: Iris (i), Digits (d), Olivetti Faces (o), and Breast Cancer (c)

	DT				SVM				LOGREG				MLP			
	i	d	o	c	i	d	o	c	i	d	o	c	i	d	o	c
DT	0.91	1.31	3.19	1.08	1.04	TO	323.72	1.00	0.96	TO	206.67	**1.00**	1.23	TO	TO	7.09
SVM	0.93	1441.79	188.65	0.96	**0.87**	1.08	2.75	1.02	1.02	TO	TO	1.02	1.54	TO	TO	2.40
LOGREG	1.02	TO	495.57	0.99	1.13	TO	TO	1.04	1.07	1.10	4.52	1.03	1.05	TO	TO	6.32
MLP	1.46	TO	TO	16.03	2.16	TO	TO	2.72	1.08	TO	TO	3.18	0.97	1.18	3.02	1.04

Table 3. RQ3 Overhead of adding feature constraints computed as factors for times from Table 2. TO indicates a timeout after 1h

used in RQ2. Feature constraints are specific to each dataset. We set the feature boundaries to be between 0 and 1 for the Olivetti Faces dataset while assuming that the features for the Iris and Breast Cancer datasets were exclusively positive values. Finally, we restricted the features in the Digits dataset to integer values within the range of 0 to 16.

The outcomes of our experiment are presented in Table 3 as factors of the corresponding times shown in Table 2. The computation timed out mainly on complex models such as MLP and datasets with higher dimensionality and features (no analyses timed out for classifiers of the Iris and Breast Cancer datasets). It is easy to see that adding constraints with more features makes the SMT problem computationally expensive. On the other hand, datasets with fewer features can be handled relatively quickly. Interestingly, there are a few instances where imposing feature constraints reduces the computation time instead of increasing it. In some cases, the computation time remains unaffected despite the imposition of feature constraints.

7 Related Work

Analyses of Classifiers Various works analyze individual machine learning models. Techniques, such as SMT solving [19,20,7,9,32], approximation/abstract interpretation [34,15,22,11], and MILP [40,29,44], are utilized to verify neural networks. MILP reduces the verification problem into a mixed integer linear program. Solving the MILP enables the assessment of safety by identifying lower bounds on global robustness through the use of adversarial examples [13,4,8] or counterexample-guided abstraction refinement (CEGAR) [6] to find property violations.

An approach based on the MILP solver is presented in [8] to establish bounds for feed-forward neural networks with ReLU activations. Local search allows it to identify better input solutions within updated target bounds until no solution exists. In contrast, [13] maximizes hidden neuron activation to create adversarial examples, using the MILP solver to achieve precise results by minimizing and maximizing neuron values layer by layer. Tjeng et al. [40] also proposed a MILP-based method to find the nearest adversarial example based on a given distance metric, using a progressive bound tightening procedure that quickly obtains initial coarse bounds. If these bounds are inadequate, they use a linear programming solver to find tighter lower and upper bounds, similar to [13].

Approximation and abstract interpretation-based methods are generally incomplete [41]. The AI^2 framework [15] was the first to use abstract interpretation, where their method over-approximated neural network computations by interpreting them in an abstract domain. Singh et al. later developed different abstract domains [35,36,37,34] and created an abstract domain library called ERAN[2]. Prabhakar and Afzal [29] proposed an abstraction for feed-forward neural networks with ReLU activations, carefully replacing the neural network weights with intervals to account for the merging. While these methods primarily focus on feed-forward neural networks, Zhang et al. [44] proposed another approach for recurrent neural networks. Other classical machine learning models, such as Support Vector Machines [30] and Decision Trees [10], have also been formally analyzed for their robustness against adversarial perturbations.

All of the above works differ from MLDiff as they analyze single models. However, their use of high-level reasoning tools and techniques to handle complex analyses by abstraction and approximation techniques could likely similarly be applied to MLDiff addressing performance issues like timeouts as observed in RQ3 of Sect. 6.3.

Recent works such as NeuroDiff [27], ReluDiff [26] aim to prove that two similar neural networks behave equivalently or within bounded differences for all inputs, enhancing verification with symbolic approximations and gradient refinements. However, these methods are limited to closely related networks and do not extend to diverse classifiers. MLDiff, instead, addresses the broader challenge of analyzing classifiers to find commonalities and disagreements, regardless of architecture, training, or data.

ML Analysis using SMT The use of SMT-based analyses presents a straightforward approach to verification by transforming an analysis problem into a constraint satisfaction one. This methodology is extensively employed to validate machine learning models [41]. Frameworks such as Reluplex [19] and its successor Marabou [20] utilize SMT-based techniques to verify safety properties of neural networks. Marabou uses a divide-and-conquer approach for parallel execution and maintains symbolic bounds for neurons as linear combinations of inputs. Planet [9] employs an approximation technique to reduce the search space for an SMT solver. This technique leverages interval arithmetic to estimate the bounds for each neuron, encodes the neural network's behavior as linear constraints, and conjoins it with a safety property for verification. A practical and effective

[2] ERAN:https://github.com/eth-sri/eran

bounded model checking-based approach is presented in [31] to verify a controller (Cart Pole System) with a multilayer perceptron. Huang et al. [17] propose a systematic method to prove local robustness in neural networks against adversarial attacks. The approach checks a finite set of points that lead to the same output and propagates constraints between neural network layers. Again, while the works listed above differ from MLDiff in the properties and models analyzed, their advanced encoding and use of specialized SMT solver features may be very beneficial for our algorithms presented in Sect. 5.2.

Sharma and Wehrheim [33] proposed verification-based testing to check the fairness of decision trees. They encode two data instances into SMT (rather than two classifiers, as we do for MLDiff) to derive fairness tests. Sharma et al. [32] propose MLCHECK, property-driven machine learning model testing with a property specification language and systematic test case generation. MLCHECK encodes decision trees and neural networks as SMT formulas, checks for satisfiability, and uses the counterexample as a test input. In another work, Vehicle [7], a compiler for compiling verifiers and theorem provers into a high-level specification for neural networks. Property specification languages such as MLCHECK [32] and Vehicle [7] could be useful extensions of MLDiff to support the formulation of advanced analysis queries.

8 Conclusion

We have presented MLDiff, a novel approach to the comparison of learned classifiers. We have motivated and illustrated the use of MLDiff on real-world datasets and examples. We provide a prototypical implementation that supports Decision Tree, Support Vector Machine, Logistic Regression, and Neural Networks classifiers. An evaluation of MLDiff using the Z3 SMT-solver [24] on popular datasets and classifier combinations shows promising results in terms of effectiveness and performance. Performance of the prototype degrades for more complex models and queries, e.g., when including feature constraints. We leave exploring diverse approaches for realizing efficient MLDiff implementations as future work.

Data Availability

We have made the MLDiff framework and a prototypical implementation available on GitHub as [38].

References

1. Alpaydin, E., Kaynak, C.: Optical recognition of handwritten digits (Jun 1998). https://doi.org/10.24432/C50P49, accessed on 2025-08-22.
2. Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A.V., Criminisi, A.: Measuring neural net robustness with constraints. In: NIPS 2016. pp. 2613–2621 (2016)

3. Bishop, C.M.: Pattern recognition and machine learning, 5th Edition. Information science and statistics, Springer (2007), https://www.worldcat.org/oclc/71008143
4. Bunel, R., Turkaslan, I., Torr, P.H.S., Kohli, P., Mudigonda, P.K.: A unified view of piecewise linear neural network verification. In: NeurIPS 2018. pp. 4795–4804 (2018)
5. Cebeci, C., Bjørner, N., Candea, G., Pit-Claudel, C.: A conjecture regarding smt instability. In: SMT 2025. CEUR Workshop Proceedings, vol. 4008, pp. 136–147. CEUR-WS.org (2025)
6. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: CAV 2000. LNCS, vol. 1855, pp. 154–169. Springer (2000). https://doi.org/10.1007/10722167_15
7. Daggitt, M.L., Kokke, W., Atkey, R., Arnaboldi, L., Komendantskaya, E.: Vehicle: Interfacing neural network verifiers with interactive theorem provers. CoRR abs/2202.05207 (2022), https://arxiv.org/abs/2202.05207
8. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward neural networks. In: NFM 2018. LNCS, vol. 10811, pp. 121–138. Springer (2018). https://doi.org/10.1007/978-3-319-77935-5_9
9. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: ATVA 2017. LNCS, vol. 10482, pp. 269–286. Springer (2017). https://doi.org/10.1007/978-3-319-68167-2_19
10. Einziger, G., Goldstein, M., Sa'ar, Y., Segall, I.: Verifying robustness of gradient boosted models. In: IAAI 2019. pp. 2446–2453. AAAI Press (2019). https://doi.org/10.1609/AAAI.V33I01.33012446
11. Elboher, Y.Y., Gottschlich, J., Katz, G.: An abstraction-based framework for neural network verification. In: CAV 2020. LNCS, vol. 12224, pp. 43–65. Springer (2020). https://doi.org/10.1007/978-3-030-53288-8_3
12. Elias, T.S., Dykeman, P.A.: Edible wild plants: a North American field guide. Sterling Publishing Company, Inc. (1990)
13. Fischetti, M., Jo, J.: Deep neural networks and mixed integer linear optimization. Constraints 23(3), 296–309 (2018). https://doi.org/10.1007/S10601-018-9285-6
14. Fisher, R.A.: Iris (Jun 1988). https://doi.org/10.24432/C56C76, accessed on 2025-08-22.
15. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.T.: AI2: safety and robustness certification of neural networks with abstract interpretation. In: SP 2018. pp. 3–18. IEEE Computer Society (2018). https://doi.org/10.1109/SP.2018.00058
16. Hastie, T., Tibshirani, R., Friedman, J.H.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition. Springer Series in Statistics, Springer (2009). https://doi.org/10.1007/978-0-387-84858-7
17. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: CAV 2017. LNCS, vol. 10426, pp. 3–29. Springer (2017). https://doi.org/10.1007/978-3-319-63387-9_1
18. Ignatiev, A., Cooper, M.C., Siala, M., Hebrard, E., Marques-Silva, J.: Towards formal fairness in machine learning. In: CP 2020. LNCS, vol. 12333, pp. 846–867. Springer (2020). https://doi.org/10.1007/978-3-030-58475-7_49
19. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer (2017). https://doi.org/10.1007/978-3-319-63387-9_5
20. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljic, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.W.: The

marabou framework for verification and analysis of deep neural networks. In: CAV 2019. LNCS, vol. 11561, pp. 443–452. Springer (2019). https://doi.org/10.1007/978-3-030-25540-4_26

21. Konnov, I.: Edmund m. clarke, thomas a. henzinger, helmut veith, and roderick bloem (eds): Handbook of model checking - springer international publishing ag, cham, switzerland, 2018. FAC 31(4), 455–456 (2019). https://doi.org/10.1007/S00165-019-00486-Z
22. Li, J., Liu, J., Yang, P., Chen, L., Huang, X., Zhang, L.: Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In: SAS 2019. LNCS, vol. 11822, pp. 296–319. Springer (2019). https://doi.org/10.1007/978-3-030-32304-2_15
23. Liu, N.: scikit-learn olivetti faces dataset olivettifaces.mat (9 2016). https://doi.org/10.6084/m9.figshare.3829989.v2
24. de Moura, L.M., Bjørner, N.S.: Z3: an efficient SMT solver. In: TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24
25. Neumaier, A., Shcherbina, O.: Safe bounds in linear and mixed-integer linear programming. Math. Program. 99(2), 283–296 (2004). https://doi.org/10.1007/S10107-003-0433-3
26. Paulsen, B., Wang, J., Wang, C.: Reludiff: differential verification of deep neural networks. In: ICSE 2020. pp. 714–726. ACM (2020). https://doi.org/10.1145/3377811.3380337
27. Paulsen, B., Wang, J., Wang, J., Wang, C.: NEURODIFF: scalable differential verification of neural networks using fine-grained approximation. In: ASE 2020. pp. 784–796. IEEE (2020). https://doi.org/10.1145/3324884.3416560
28. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., VanderPlas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in python. JMLR 12, 2825–2830 (2011). https://doi.org/10.5555/1953048.2078195
29. Prabhakar, P., Afzal, Z.R.: Abstraction based output range analysis for neural networks. In: NeurIPS 2019. pp. 15762–15772 (2019)
30. Ranzato, F., Zanella, M.: Robustness verification of support vector machines. In: SAS 2019. LNCS, vol. 11822, pp. 271–295. Springer (2019). https://doi.org/10.1007/978-3-030-32304-2_14
31. Scheibler, K., Winterer, L., Wimmer, R., Becker, B.: Towards verification of artificial neural networks. In: MBMV 2015. pp. 30–40. Sächsische Landesbibliothek (2015)
32. Sharma, A., Demir, C., Ngomo, A.N., Wehrheim, H.: MLCHECK- property-driven testing of machine learning classifiers. In: ICMLA 2021. pp. 738–745. IEEE (2021). https://doi.org/10.1109/ICMLA52953.2021.00123
33. Sharma, A., Wehrheim, H.: Automatic fairness testing of machine learning models. In: ICTSS 2020. LNCS, vol. 12543, pp. 255–271. Springer (2020). https://doi.org/10.1007/978-3-030-64881-7_16
34. Singh, G., Ganvir, R., Püschel, M., Vechev, M.T.: Beyond the single neuron convex barrier for neural network certification. In: NeurIPS 2019. pp. 15072–15083 (2019)
35. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.T.: Fast and effective robustness certification. In: NeurIPS 2018. pp. 10825–10836 (2018)
36. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An abstract domain for certifying neural networks. PACMPL 3(POPL), 41:1–41:30 (2019). https://doi.org/10.1145/3290354

37. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: Boosting robustness certification of neural networks. In: ICLR 2019. OpenReview.net (2019)
38. Soaibuzzaman, Döring, J., Kasi, S., Ringert, J.O.: MLDiff github repository (2025), available from https://github.com/se-buw/MLDiff
39. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R.: Intriguing properties of neural networks. In: ICLR 2014 (2014), http://arxiv.org/abs/1312.6199
40. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: ICLR 2019. OpenReview.net (2019)
41. Urban, C., Miné, A.: A review of formal methods applied to machine learning. CoRR abs/2104.02466 (2021), https://arxiv.org/abs/2104.02466
42. Wexler, J., Pushkarna, M., Bolukbasi, T., Wattenberg, M., Viégas, F.B., Wilson, J.: The what-if tool: Interactive probing of machine learning models. TVCG 26(1), 56–65 (2020). https://doi.org/10.1109/TVCG.2019.2934619
43. Z3Prover repository. https://github.com/Z3Prover/z3, accessed on 2025-08-22.
44. Zhang, H., Shinn, M., Gupta, A., Gurfinkel, A., Le, N., Narodytska, N.: Verification of recurrent neural networks for cognitive tasks via reachability analysis. In: ECAI 2020. FAIA, vol. 325, pp. 1690–1697. IOS Press (2020). https://doi.org/10.3233/FAIA200281
45. Zwitter, M., Soklic, M.: Breast Cancer. UCI Machine Learning Repository (1988). https://doi.org/10.24432/C51P4M, accessed on 2025-08-22.