Empirical Software Engineering (2023) 28:34
https://doi.org/10.1007/s10664-022-10273-9

Differential testing for machine learning: an analysis for classification algorithms beyond deep learning

Steffen Herbold[1] · **Steffen Tunkel**[2]

Accepted: 6 December 2022 / Published online: 24 January 2023

Abstract
Differential testing is a useful approach that uses different implementations of the same algorithms and compares the results for software testing. In recent years, this approach was successfully used for test campaigns of deep learning frameworks. There is little knowledge about the application of differential testing beyond deep learning. Within this article, we want to close this gap for classification algorithms. We conduct a case study using Scikit-learn, Weka, Spark MLlib, and Caret in which we identify the potential of differential testing by considering which algorithms are available in multiple frameworks, the feasibility by identifying pairs of algorithms that should exhibit the same behavior, and the effectiveness by executing tests for the identified pairs and analyzing the deviations. While we found a large potential for popular algorithms, the feasibility seems limited because, often, it is not possible to determine configurations that are the same in other frameworks. The execution of the feasible tests revealed that there is a large number of deviations for the scores and classes. Only a lenient approach based on statistical significance of classes does not lead to a huge amount of test failures. The potential of differential testing beyond deep learning seems limited for research into the quality of machine learning libraries. Practitioners may still use the approach if they have deep knowledge about implementations, especially if a coarse oracle that only considers significant differences of classes is sufficient.

Keywords Machine learning · Software testing · Differential testing

Communicated by: Shaukat Ali

✉ Steffen Herbold
steffen.herbold@uni-passau.de

Steffen Tunkel
steffen.tunkel@stud.uni-goettingen.de

[1] Faculty of Computer Science and Mathematics, University of Passau, Passau Germany

[2] Institute of Computer Science, University of Goettingen, Goettingen, Germany

1 Introduction

With differential testing, multiple implementations of the same algorithm[1] are executed with the same data. Inconsistencies between the results are indicators of bugs. In recent years, this idea was adopted to define test oracles for deep learning (e.g., Pham HV et al. 2019;Wang Z et al. 2020; Guo Q et al. 2020; Asyrofi MH et al. 2020) . This works well, because there are multiple versatile frameworks, such as PyTorch (Paszke A et al. 2019) and TensorFlow (Abadi et al. 2016) that allow the definition of exactly the same neural network structures and training procedures. Beyond deep learning, this idea was also successfully applied to linear regression (McCullough BD et al. 2019), to cases where the optimal solutions were known as oracles beforehand. Due to the importance of linear regression as a basic technique, there are many powerful implementations of this, which enabled the definition of differential testing. Moreover, the known optimal solution served as an additional test oracle. However, even though differential testing was successful for deep learning and linear regression, it is not obvious that this should also be the case for other machine learning algorithms. For example, while the general concept of random forests (Breiman L 2001) is well-defined,[2] other aspects depend on the implementation, e.g., which decision tree algorithm is used and how the sub-sampling can be configured. Whether we can expect random forest implementations to behave the same depends on how developers navigate these options, e.g., if they implement the same variants in a hard-coded way or if they expose configuration options through hyperparameters. Due to this, it is unclear if and how different implementations of the same algorithms can be directly compared to each other. With deep learning, this problem does not exist: network structures, training procedures, loss functions, and optimization algorithms are all configured by the user of the framework through the API. Moreover, while some deviations are always expected with randomized algorithms like the training algorithms for (deep) neural networks, other algorithms are deterministic and should lead to exactly the same results, which was also not yet considered.

Thus, while we know that differential testing can be useful to define pseudo oracles (Davis MD and Weyuker EJ 1981) for the quality assurance of algorithms, we lack similar knowledge for other types of machine learning tasks beyond deep learning and linear regression. Within our work, we close this gap for classification algorithms and investigate the following research question.

Research Question What is the *potential*, *feasibility*, and *effectiveness* of differential testing for classification algorithms beyond deep learning, i.e., for techniques like decision trees, random forests, or k-nearest neighbor classification?

We investigate our research question within an exploratory case study of four machine learning frameworks: Scikit-learn (Pedregosa et al. 2011), Weka (Frank et al. 2016), Spark MLlib (Meng et al. 2016), and Caret (Kuhn M 2018). We use a methodology with three phases within our study. First, we identify for which algorithms we find multiple implementations. Through this, we evaluate the *potential*, i.e., for how many algorithms differential testing could possibly work, due to the availability of multiple implementations in large frameworks. Second, we compare the algorithms in detail to understand if it is possible

[1]Following ISO/IEC/IEEE 24765:2017, an implementation is the translation of a design into software components (ISO/IEC/IEEE 2017). In our case, the design is the (often abstract or mathematical) definition of an algorithm, the software components are the executable code for an algorithm in a machine learning framework that operationalizes an algorithm.

[2]bootstrap sampling of instances, subsets of features for decision

to configure them in the same way using their public interfaces through the definition of appropriate hyperparameters. Through this, we evaluate the *feasibility*, i.e., if we can actually exploit the potential because implementations are sufficiently similar to serve as pseudo oracles and identify inconsistencies. Third, we implement differential testing for the algorithms for which we are able to find identical configurations. Through this, we evaluate the *effectiveness* of differential testing, i.e., the capability to find inconsistencies with the goal to reveal potential bugs.

Through this study, we contribute the following to the state-of-the-art.

- We found that while there is a large potential for differential testing, due to different implementations of the same algorithms, it is difficult to harness the potential and find feasible combinations of algorithms that should behave the same. This is due to the diversity in hyperparameters between the implementations.
- We observe many deviations between implementations that should yield similar results. The number of deviations indicates that there is a large number of noise in the results, which makes it impossible for us to pick up a reliable signal to identify possible bugs. Experts that want to test specific algorithms could possibly use a lenient test oracle that only considers if the classifications are significantly different, but it is unclear if such an oracle is sufficiently powerful to detect bugs.

The remainder of this article is structured as follows. We define our terminology and notations in Section 2, followed by a discussion of the related work in Section 3. Then, we present our case study in Section 4, including our subjects, methodology, and results for each of the three phases. We discuss the implications of our results in Section 5, followed by the threats to validity in Section 6, before we conclude in Section 7.

2 Terminology and Notations

Within this article, we focus on *binary classification*. Formally, we have instances $x_i = (x_{i,1}, ..., x_{i,m}) \in \mathcal{F} \subseteq \mathbb{R}^m$ with labels $y_i \in \{0, 1\}$ for $i = 1, ..., n$. We say that $\mathcal{F}$ is the feature space. The binary set $\{0, 1\}$ represents the classes and we note that the classes are considered as categories and can be replaced by any other binary set, e.g., $\{false, true\}$ or {-1, +1}. A classification model tries to find a function $f : \mathcal{F} \rightarrow \{0, 1\}$ such that $f(x_i) \approx y_i$. Often, this is done by estimating scores for both classes $c \in \{0, 1\}$ as $f^c_{score} : \mathcal{F} \rightarrow \mathbb{R}$ such that $f(x_i) = \arg\max_{c \in \{0,1\}} f^c_{score}(x_i)$. We note that the scores are often probabilities of classes and the class assignment can also be optimized using different thresholds for scores. However, neither the question if the scores represent probabilities nor the optimization of thresholds is relevant to work and not further discussed.

Based on the above definitions, a classification algorithm A is an algorithm that takes as input training instances $(X^{train}, Y^{train}) = (x^{train}_i, y^{train}_i), i = 1, ..., n^{train}$ and outputs functions f, $f_{score} = A(X^{train}, Y^{train})$. We further define $(X^{test}, Y^{test}) = (x^{test}_i, y^{test}), i = 1, ..., n^{test}$ as test data. When we discuss the comparison of two algorithms, we refer to them as A^1 and A^2 with the respective functions f^1, f^1_{score}, f^2, and f^2_{score} as result of the training. Moreover, we use the notation $\mathbb{1}_{condition}$ for the indicator function, which is one when the condition is fulfilled and zero otherwise.

We note that we use the terms false negative and false positive not with respect to the classification of the algorithms, as is common in the literature about machine learning. Instead, we use these terms in their common meaning in the software testing literature: a false

positive is a software test that fails, even though there is no wrong behavior and, vice versa, a false negative is a software test that misses wrong behavior it should detect.

3 Related Work

We restrict our discussion of related work to other studies that utilize multiple implementations of the same algorithms for the definition of software tests. There are several recent surveys on software engineering and software testing for machine learning, to which we refer readers for a general overview (Zhang JM et al. 2020; Braiek HB and Khomh F 2020; Giray G 2021; Martínez-Fernández et al. 2021). This includes other approaches for addressing the oracle problem for machine learning, like metamorphic testing (i.e., systematic changes to input data such that the expected changes in output are known,(e.g., Murphy et al. 2008; Xie X et al. 2011; Ding J et al. 2017), using crashes as oracle (e.g., Herbold S and Haar T, 2022), defining numeric properties that can be checked (e.g., Karpathy A, 2018) or trying to map the performance of predictions to requirements to derive tests (e.g., Barash G et al., 2019). Further, there is also a different type of differential testing approach that is used for the testing of the robustness of models (e.g.,Pei K et al. 2019; Guo J et al. 2021), i.e., similar deep learning models or inputs were used as pseudo oracles. However, such tests of specific models are not within the scope of our work, which considers the testing of the underlying libraries that implement machine learning algorithms and not the testing of learned models.

One method to solve the test oracle problem of machine learning (e.g.,Murphy C et al. 2007; Groce A et al. 2014; Marijan and Gotlieb 2020) is to implement a differential testing between multiple implementations of the same algorithm (Murphy C et al. 2007) and use the different implementations as pseudo-oracles (Davis MD and Weyuker EJ 1981). While this idea is not new for machine learning and was already shown to work for the MartiRank algorithm (Gross P et al. 2006) by Murphy C et al. (2007), there was only little follow up work in this direction until recently.

Most notably, this idea gained traction for the testing of deep learning frameworks. (Pham HV et al. 2019) developed the tool CRADLE, which used, e.g., TensorFlow (Abadi et al. 2016) and Theano (Theano Development Team 2016) as backends. The comparison of results of outputs between different implementations was effective at finding errors, even though only few different deep learning models were used. Following this study, there were many papers that build upon this work, e.g., LEMON by Wang Z et al. (2020) that proposes a mutation approach to cover more different network architectures during the testing, AUDEE by Guo Q et al. (2020) that uses a genetic algorithm to generate effective tests, and CrossASR by Asyrofi MH et al. (2020) which applies a similar concept for the testing of automated speech recognition models. Beyond deep learning, the idea was also used by McCullough BD et al. (2019) for the comparison of different implementations of linear regression algorithms and was shown to be effective. However, we note that while McCullough BD et al. (2019) compared different implementations to each other, they use analytic solutions as ground truth, i.e., they did not really use the implementations themselves as pseudo oracles.

In comparison to prior work, our focus is not on the differential testing of a single algorithm (Murphy C et al. 2007; McCullough BD et al. 2019) or deep learning. Instead, we consider this question broadly for classification algorithms. With the exception of multilayer perceptrons, we exclude deep learning, which can also be used for classification, as

these are already studied.[3] We further note that neither Murphy C et al. (2007) nor McCullough BD et al. (2019) studied classification algorithms, but rather a ranking algorithm and a regression algorithm. To the best of our knowledge, this is, therefore, the first work that considers differential testing for non-deep learning classification algorithms.

4 Case Study

Within this section, we discuss the main contribution of our work: a case study about four machine learning libraries that explores the usefulness of differential testing for classification algorithms. In the following, we first discuss the subjects of our study, followed by a description of our methodology, including the variables we measure and the research methods we use. Then, we proceed with the presentation of the results for each phase of our case study. We provide our implementation of the tests and additional details through our replication kit online (Tunkel S and Herbold S 2022).

4.1 Subjects

We used purposive sampling (Patton 2014) for our case study based on four criteria. First, we wanted to use subjects implemented in different programming languages. One important use case of differential testing is to validate the behavior of new implementations of algorithms, which includes the implementation in a new programming language. Second, all subjects should be implemented by, to the best of our knowledge, independent project teams. Overlap between teams could have an adverse effect on our study of differential testing, because our results could be affected by the same developers working on multiple implementations. There is a non-negligible likelihood that such a developer would make similar design and implementation choices, which could lead to an overestimation of similarities between independent implementations. Third, subjects should cover multiple algorithms. If our subjects are too small, this would negatively affect our research as the likelihood of finding the same algorithms multiple times would decrease. Fourth, the libraries should not focus on deep learning. Since our goal is to provide a perspective beyond deep learning, we specifically exclude libraries whose main use case is deep learning, even if the methods provided by these libraries could also be used to implement, e.g., a linear logistic regression model. Based on these criteria, we selected four such libraries, as shown in Table 1.

4.2 Methodology

Our case study was conducted in three phases, as depicted in Fig. 1. Each phase provides insights into a different aspect of the usefulness of differential testing. Moreover, each phase builds on the results of the previous phase.

In the first phase, we evaluate the basic assumption of differential testing: we have multiple implementations of the same algorithm. To evaluate this, we analyze each framework and extract the classification algorithms. We then compare these lists to each other to determine overlaps. The number of overlaps determines the potential for the differential testing, because this means that multiple implementations are available. For all frameworks, we

[3] Multilayer perceptrons are commonly found in general purpose machine learning libraries but do not provide the same flexibility as deep learning frameworks.

Table 1 Overview of the machine learning libraries selected for our study

Framework	Version	Language	Description
Scikit-learn	1.0.1	Python	Scikit-learn (Pedregosa et al. 2011) is a popular machine learning library for Python and one of the reasons for the growing popularity of python as a language for data analysis.
Weka	3.8.5	Java	Weka is a popular machine learning library for Java which had the first official release in 1999. The library is very mature and has been used by researchers and practitioners for decades and is, e.g., part of the Pentaho business intelligence software.[4]
Spark MLLib	3.0.0	Scala	Spark MLLib (Meng et al. 2016) is the machine learning component of the rapidly growing big data framework Apache Spark (Zaharia M et al. 2010) that is developed with Scala.
Caret	6.0-90	R	Caret (Kuhn M 2018) provides common interfaces for many learning algorithms implemented in R. Through this, Caret provides a harmonized Application Programing Interface (API) that allows using many learning algorithms that are implemented with different APIs as part of different R packages. Thus, Caret greatly reduces the difficulty of trying out different algorithms to solve a problem.

[4] https://www.hitachivantara.com/en-us/products/data-management-analytics/pentaho-platform.html

exclude pure meta-learners from our analysis. meta-learners are wrappers around other classifiers (e.g., generic boosting (Freund Y and Schapire RE 1997) or bagging (Breiman L 1996)), i.e., they propagate the learning to other classifiers called base classifiers. Thus, meta-learners require special consideration: we must first establish that there are other equal algorithms, because otherwise it would be unclear if differences observed when comparing meta-learners are due to deviations between the meta-learners, or due to differences between the base classifiers.

In the second phase, we evaluate if we actually manage to implement differential tests. This is non-trivial due to the hyperparameters of algorithms. Hyperparameters allow the users of a machine learning algorithm to configure the training and/or the resulting model, e.g., by restricting sizes, enforcing regularization, or even by selecting which variant of an algorithm is used. While the same is true for deep learning, there is a big difference: with deep learning, every user of a library has to configure the hyperparameters for each neural network and training. On the one hand, this increases the burden on the developers, as this is not pre-configured. On the other hand, there are no restrictions and thus, defining an equivalent set of hyperparameters is not problematic. This is different with other algorithms. For example, there are many different decision tree variants, such as CART (Brieman et al. 1984), ID3 (Quinlan 1986), and C4.5 (Quinlan 1993). A random forest could be based on each of these variants, in which case the random forests could not be directly compared. However, even the same variant of a decision tree is not always the same, as concrete implementations may not implement the original specification or they may have added more features, e.g., for avoiding overfitting through limiting tree depths or requiring a certain number of instances to make decisions. Thus, within the second phase, we compare different implementations based on their API documentations and determine for which overlaps we manage to find matching configurations. For Caret, we also included the API documentation of the R packages that are providing the implementation that is wrapped by Caret in our analysis. Furthermore, our analysis also includes the reasons for cases where we fail to match the hyperparameters.

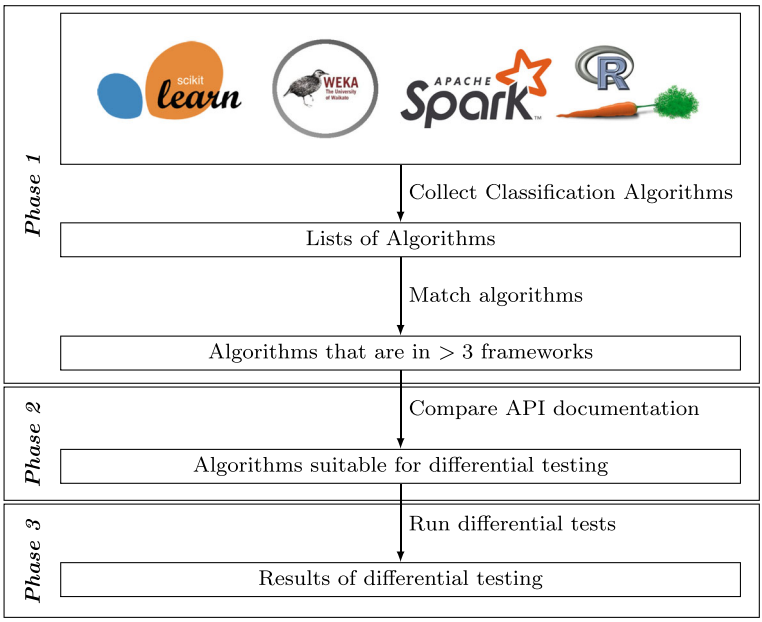

Fig. 1 Methodology overview

In the third phase, we implement a simple differential testing approach. We use four data sets as input.

- UNIFORM: 200 randomly generated instances, with ten features that are uniform in the interval [0, 1] with two classes, such that 50% of the data are in each class. The separation between the classes is based on a 10-dimensional rectangle, i.e., a clear separation between the classes is possible.
- RANDOM: 200 randomly generated instances, where the features are uniform within the interval [0, 1] with two classes that are randomly assigned. Thus, no reasonable classification is possible.
- BC: The *breast cancer* data from the UCI repository has 569 instances with 30 numeric features for cancer prediction. The features are normalized to the interval [0, 1].
- WINE: The *wine* data from the UCI repository has 178 instances with 11 numeric features for wine quality prediction. The features are normalized to the interval [0, 1].

The first two data sets are reused from our prior work (Herbold S and Haar T 2022) for the smoke testing of algorithms. With UNIFORM, we can test if different implementations perform the same, for data that is relatively easy to learn based on informative features. With RANDOM, we test if the implementations also yield the same results, in case the data is not informative. This scenario can be considered "more difficult", as the resulting model is more or less random. Still, the "randomness" should depend on the algorithm and, ideally, different implementations still yield the same results. The other two data sets are popular data sets that are often used as examples for using machine learning models. All data sets are randomly split into 50% training data and 50% test data.

The data we use have two aspects in common: they have two classes and the features are within well-bounded and numerically not problematic feature spaces. Consequently, we are

implementing a relatively simple differential testing scenario, i.e., we do not intentionally stress the algorithms.[5] Our rationale for this is that we want to test the general feasibility of differential testing to find errors due to deviations between implementations. If our data would target corner cases, our results could overestimate the effectiveness because differences might be larger for extreme data or be polluted by algorithms crashing failures due to numeric issues. Moreover, classification with more classes either uses exactly the same algorithms (e.g., decision trees and neural networks) or a combination of multiple binary models (e.g., one-vs-follower approach with support vector machines). However, we may also underestimate the difference between algorithms for multiple classes.

We use four criteria to evaluate the outcome of the differential tests for two algorithms A^1 and A^2. The first two criteria measure if the results are equal, i.e., if two different implementations yield the same results. To evaluate this, we consider the number of differences between the classes predicted by the two algorithms defined as

$$\Delta = \sum_{i=1}^{n} \mathbb{1}_{f^1(x_i) \neq f^2(x_i)} \tag{1}$$

as the first criterion and the number of deviations between the scores defined as

$$\Delta_{score} = \sum_{i=1}^{n} \mathbb{1}_{f^1_{score}(x_i) \neq f^2_{score}(x_i)} \tag{2}$$

as the second criterion. Please note that we consider scores to be equal, if the difference is less than 0.001. Smaller differences are almost always irrelevant and may easily be due to differences in how floating point operations are implemented. If we were to enforce exact equality of scores, this could inflate the false positive test results.

Additionally, we often cannot expect that two results of different implementations are exactly the same, e.g., because an aspect of the algorithm is randomized. However, the results should not differ too much and we should still get roughly the same result in terms of overall performance. We evaluate this by looking at the significance of the differences between the classes and scores and derive the two remaining criteria. The third criterion considers the differences in the scores. Since the scores are continuous values and we cannot make general assumptions regarding their distribution, we use the Kolmogorov-Smirnoff test. Thus, we check if the scores of the two models have the same distribution. The fourth criterion applies this concept to the classes. Similarly, we use the Chi-Squared test to compare the classes to check if the differences in classifications are significant. We reject the null hypotheses of the tests (no difference) if we observe a p-value of less than 0.05. We do not conduct any correction of repeated tests, as this correction would be dependent on the number of tests that are executed, i.e., become a parameter of the test campaign. This is hard to implement in practice. However, this means that we expect about 5% of the significant differences we observe to be false positives, because the p-value follows a uniform distribution if the null hypothesis is true. If and how this affects the differential testing will be discussed based on our results.

While it may seem counterintuitive that we use four criteria to evaluate the effectiveness of software tests, these criteria allow us not only to answer the general question if differential tests are effective, but also how lenient the comparison of the results must be in order to not

[5]This is also the reason why we only use the test cases UNIFORM and RANDOM from our prior work (Herbold S and Haar T 2022). Our analysis of crashes showed that such data does not lead to crashes, which would make the analysis of the effectiveness of the differential testing more difficult.

yield false positives. This is due to the nature of machine learning. Figure 2 shows two cases, where correct implementations of the same algorithm can yield different optimal outcomes. In both examples, the absolute differences we observe (the first two criteria) are not suitable to determine that a test failed, i.e., one result is wrong. However, the differences between the outcomes should not be statistically significant. Ideally, we hope to see the limits of both absolute comparisons (criteria one and two) and statistical comparison (criteria three and four) for the differential testing as criteria for passing or failing differential tests.

Additionally, we apply all four criteria to both the training and the test data. Both training and test data have advantages and disadvantages. The training data has the advantage that it is readily available. However, software tests that only rely on training data may miss cases in which the implementations do not correctly generalize beyond the training data. This is the advantage of test data, as the computed functions for scoring and classification are evaluated on an independent data set. However, this may also be prone to more false positives, because equally good results on the training data may lead to different results on the test data (see Fig. 2). Since we use the same amount of training and test data, the likelihood of differences between implementations on the training and test data is not impacted by the amount of data available.

4.3 Phase 1: Overlap of Algorithms

Table 2 shows the overlapping algorithms of the four machine learning frameworks we found using the API docs. We grouped the algorithms by their underlying paradigm, e.g., *Naive Bayes* or *Decision Tree*. The first row demonstrates the reason for this. While Scikit-learn provides unique classes for different variants of Naive Bayes, Spark MLlib has only a single class. However, the implementation in Spark MLlib can be configured to use different variants, e.g., Gaussian Naive Bayes or Multinomial Naive Bayes. Our data shows that there is a significant overlap:

- Naive Bayes, Decision Trees, Random Forest, Support Vector Machines, Multilayer Perceptrons, and Logistic Regression can be found in all frameworks;
- a trivial classifier, k-Nearest Neighbor, and Gradient Boosting in at least three of the frameworks; and

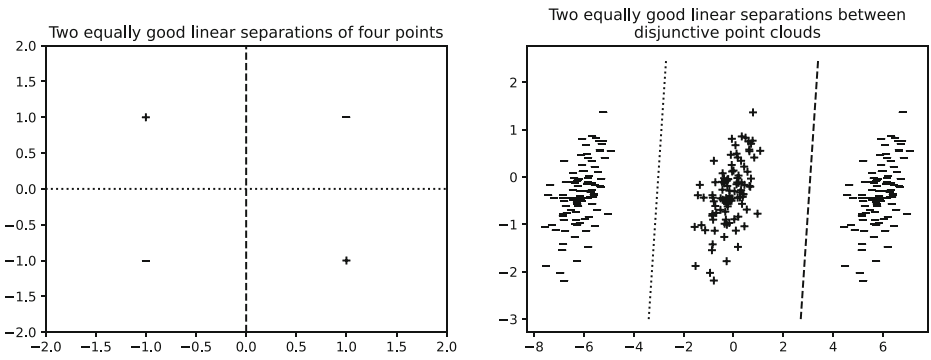

Fig. 2 Two examples of data sets, where the dotted and dashed lines represent possible linear separations of the data that could, e.g., be computed by a logistic regression. Both models are equally good in terms of accuracy

Input: A^1, A^2, (X^{train}, Y^{train}), (X^{test}, Y^{test})
Output: Number of differences Δ^{train}, Δ^{test}, Δ^{train}_{score}, Δ^{test}_{score} and p-values p^{train}_{KS}, p^{test}_{KS}, $p^{train}_{\chi^2}$, $p^{test}_{\chi^2}$

// Run training
$f^1, f^1_{score} \leftarrow A^1(X^{train}, Y^{train})$
$f^2, f^1_{score} \leftarrow A^2(X^{train}, Y^{train})$

// Criterion 1: Absolute differences between classes
$\Delta^{train} \leftarrow \sum_{i=1}^{n^{train}} \mathbb{1}_{f^1(x_i^{train}) \neq f^2(x_i^{train})}$
$\Delta^{test} \leftarrow \sum_{i=1}^{n^{test}} \mathbb{1}_{f^1(x_i^{test}) \neq f^2(x_i^{test})}$

// Criterion 2: Absolute differences between scores
$\Delta^{train}_{score} \leftarrow \sum_{i=1}^{n^{train}} \mathbb{1}_{f^1_{score}(x_i^{train}) \neq f^2_{score}(x_i^{train})}$
$\Delta^{test}_{score} \leftarrow \sum_{i=1}^{n^{test}} \mathbb{1}_{f^1_{score}(x_i^{test}) \neq f^2_{score}(x_i^{test})}$

// Criterion 3: Significance of difference between scores
$p^{train}_{KS} \leftarrow KS.test(f^1_{score}(X^{train}), f^2_{score}(X^{train}))$
$p^{test}_{KS} \leftarrow KS.test(f^1_{score}(X^{test}), f^2_{score}(X^{test}))$

// Criterion 4: Significance of difference between classifications
$p^{train}_{\chi^2} \leftarrow \chi^2.test(f^1(X^{train}), f^2(X^{train}))$
$p^{test}_{\chi^2} \leftarrow \chi^2.test(f^1(X^{test}), f^2(X^{test}))$

Algorithm 1 Differential testing algorithm

- a Perceptron, Stochastic Gradient Descent, Gaussian Process, LDA, QDA, Nearest Centroid, Extra Trees, and Logistic Model Trees are present in two frameworks.
- Additionally, there are 47 algorithms that are only implemented by a single framework (see Table 5 in the appendix).

While we are not aware of any scientific data regarding this, the eight algorithms that can be found in all frameworks are all well-known and frequently mentioned in lists of techniques that should be learned.[6] There seems to be a link between the potential for differential testing and the popularity of approaches. Intuitively, this makes sense, because developers of machine learning frameworks likely implement popular methods first. For example, Scikit-learn has a popularity/maturity based inclusion criterion based on the number of citations.[7] Another interesting aspect is that there are sometimes multiple implementations for the same algorithm within a single framework. For Scikit-learn, Weka, and Spark MLlib, these are different variants of an algorithm.[8] For Caret, this is not the case. The reason for this is that Caret is a meta-framework, i.e., a framework that wraps the implementations of

[6]Some examples from Kaggle.com: https://www.kaggle.com/general/253858 https://www.kaggle.com/getting-started/96750 https://www.kaggle.com/getting-started/268971
[7]https://scikit-learn.org/stable/faq.html#what-are-the-inclusion-criteria-for-new-algorithms
[8]For every rule, there is an exception, in this case the RidgeClassifier from Scikit-learn, which is a special case of the LogisticRegression.

Table 2 Overlapping algorithms between frameworks based on a scan of algorithm names in the API docs

Scikit-learn	Weka	Spark MLlib	Caret
BernoulliNB, ComplementNB, MultinomialNB, GaussianNB, CategoricalNB	NaiveBayesMultinomial, NaiveBayes	NaiveBayes	awnb, naive_bayes, nb, manb
DecisionTreeClassifier	SimpleCart, J48, DecisionStump, HoeffdingTree	DecisionTreeClassifier	C5.0Tree, C5.0Rules, rpartCost, C5.0Cost, rpartScore, rpart, rpart1SE, rpart2, C5.0, ctree, ctree2, bartMachine
RandomForestClassifier	RandomForest	RandomForestClassifier	wsrf, RRF, RRFglobal, rfRules, ordinalRF, ranger, Rborist, rf, ORFlog, ORFpls, ORFridge, ORFsvm, cforest
LinearSVC, NuSVC, SVC	SMO	LinearSVC	svmSpectrumString, svmRadial, svmRadialCost, svmRadialSigma, svmPoly, svmLinear, svmLinear2, svmExpoString, svmRadialWeights, svmBoundrangeString, svmLinearWeights, lssvmRadial, lssvmPoly, lssvmLinear, svmLinear3, svmLinearWeights2
MLPClassifier	MultilayerPerceptron	MultilayerPerceptron-Classifier	mlpKerasDecayCost, mlpKerasDecay, mlpKerasDropoutCost, mlpKerasDropout, mlpSGD, mlpML, mlpWeightDecayML, mlp, mlpWeightDecay, monmlp
MLPClassifier	MultilayerPerceptron	MultilayerPerceptron-Classifier	mlpKerasDecayCost, mlpKerasDecay, mlpKerasDropoutCost, mlpKerasDropout, mlpSGD, mlpML, mlpWeightDecayML, mlp, mlpWeightDecay, monmlp
LogisticRegression, RidgeClassifier	Logistic, SimpleLogistic	LogisticRegression	regLogistic, plr, polr
DummyClassifier	ZeroR		null

Table 2 (continued)

Scikit-learn	Weka	Spark MLlib	Caret
KNeighborsClassifier	IBk		snn, kknn, knn
GradientBoostingClassifier		GBTClassifier	gbm_h2o, gbm
Perceptron	VotedPerceptron		
SGDClassifier	SGD		
GaussianProcessClassifier			gausssprRadial, gausssprPoly, gausssprLinear
LinearDiscriminantAnalysis			slda, sparseLDA, sda, rrlda, Linda, rlda, rda, PenalizedLDA, pda, pda2, Mlda, loclda, stepLDA, lda, lda2, RFlda, dda
NearestCentroid			pam
QuadraticDistriminantAnalysis			qda, QdaCov, stepQDA
ExtraTreeClassifier			extraTrees
	LMT		LMT

For Scikit-learn, Weka, and Spark MLlib we report the names of the classes, in which the algorithms are implemented. For Caret, we report the name of the method within Caret

different R packages to provide a common API. Due to this, there are sometimes implementations from different R packages for the same algorithm, e.g., in the first row from the packages `naive_bayes` and `nb`. Consequently, there is even a possibility for differential testing within the Caret package, without even using other frameworks.

> There is a large potential for differential testing beyond deep learning. This potential is strongest for well-known and popular algorithms, while newer or specialized algorithms are not good candidates for differential testing.

4.4 Phase 2: Feasible Subset

Table 3 reports the feasible combinations of algorithms that we believe should be the same, based on the API documentation. Overall, we identified three variants of Naive Bayes (GNB, KDENB, MNB), two variants of the random forest (RF1 with fixed depth and RF2 without fixed depth), three variants of Support Vector Machines (LSVM with a linear kernel, PSVM with a polynomial kernel, and RBFSVM with a RBF kernel), the Multilayer Perceptron (MLP), the trivial classifier (DUMMY), the k-Nearest Neighbor (KNN) algorithm, and three variants of Logistic Regression (LR without regularization, RIDGE with ridge regularization and LASSO with lasso regularization).

This means we found feasible combinations for all candidates with implementations in at least three of the four frameworks, with the exception of the gradient boosting trees. The reason we failed here is that the Spark MLlib API does not specify which kind of decision tree is implemented. Even beyond this, while we identified many different decision trees when analyzing the potential, we could not find any feasible combination of implementations. For CART this almost worked, but was not possible because Caret did not allow passing the appropriate parameters to the underlying library. Another issue was the options that were configurable through hyperparameters (incl. defaults) and hard-coded values. In all cases, we needed to carefully consider the hyperparameters to set them to appropriate values to behave the same as with the other frameworks. There was no case, where the out-of-the-box default parameters between two implementations were equal for all implementations. As part of this analysis, we tried to identify hyperparameters, that can be set with all implementations and that should have the same effect. We tried to maximize the number of such equal hyperparameters, but often only found few, if any. In case the hyperparameters were not available in all frameworks, we tried to determine which value was used by the implementation from the API. Often, this was impossible because such implicit assumptions on the algorithm configuration that are not exposed to users through the API as configurable hyperparameters are also not documented. Overall, this means that the feasible subsets do not cover the full capabilities of the frameworks, i.e., it is not possible for any of the algorithms to test the full set of hyperparameters within our case study.

A notable aspect of finding feasible combinations is that a good knowledge about the algorithms is required. For example, the different implementations of the RIDGE group use different variants to define the strength of the regularization. While the Scikit-learn implementation LogisticRegression uses the inverse regularization strength $\frac{1}{2 \cdot \alpha}$, all other implementations, including the Scikit-learn implementation RidgeClassifier directly use α as regularization strength. Without understanding the underlying mathematical principles, deriving suitable tests is not possible. However, we do not believe that this is, in practice, a restriction on the feasibility of such tests: developers who actively work on such algorithms,

Table 3 Feasible combinations with at least three implementations

Group	Scikit-learn	Weka	Spark MLlib	Caret
GNB	GaussianNB	NaiveBayes	NaiveBayes	naive_bayes, nb
KDENB		NaiveBayes		naive_bayes, nb
MNB	MultinomialNB	NaiveBayesMultinomial	NaiveBayes	
RF1	RandomForestClassifier	RandomForest		ranger, rborist
RF2	RandomForestClassifier	RandomForest		ranger, rborist, rf
LSVM	SVC	SMO	LinearSVC	svmLinear, svmLinear2, svmLinear3
PSVM	SVC	SMO		svmPoly
RBFSVM	SVC	SMO		svmRadial
MLP	MLPClassifier	MultilayerPerceptron	MultilayerPerceptronClassifier	mlp*, mlpSGD*
DUMMY	DummyClassifier	ZeroR		null
KNN	KNeighborsClassifier	IBk		knn
LR	LogisticRegression	Logistic	LogisticRegression	regLogistic, plr, polr
RIDGE	LogisticRegression, RidgeClassifier	Logistic	LogisticRegression	regLogistic, plr
LASSO	LogisticRegression		LogisticRegression	regLogistic

Parameters are ommited and can be found in Table 6 in the Appendix. The mlp classifiers of Caret could not be executed because they require an R package which is not on CRAN anymore.

either through quality assurance or development, should have the required background on the techniques anyways.

> The potential for differential testing can often be operationalized in form of feasible combinations that should yield the same behaviour. However, such tests can only cover a subset of the implemented functionality, because of differences between the configuration options that are exposed as hyperparameters.

4.5 Phase 3: Test Execution

We executed the tests for all pairs of algorithms we identified in the second phase, i.e., 87 pairs of algorithms with the parameters defined in Table 6 in the appendix on the four data sets with both training and test data for evaluation, i.e., we have 87 pairs $\cdot$ 4 data sets $\cdot$ 2 training/test data $=$ 696 comparisons. We observed differences between the classifications for 457 pairs (65.6%), 67 (9.6%) differences were significant. Since some algorithms cannot compute scores,[9] we only have 432 pairs with scores. We observed differences between the scores for 313 pairs (72.4%) and 141 (32.6%) were significant. For better insights, we looked at the results for each group of algorithms in detail. Table 4 summarizes the results for each group. We refer to the algorithms by their framework. In case there are multiple implementations per framework, we also provide the name of the implementations.

Overall, these results paint a complex picture regarding the effectiveness. For example, we found no patterns regarding which frameworks often disagree with each other, except for a tendency that the deviations between different implementations wrapped by Caret are mostly not significant (Fig. 3). We also found no pattern regarding which data sets are responsible for the deviations (Fig. 4). Moreover, there is no group of implementations in which both classes and scores are equal. We observe so many absolute differences that we cannot really conclude anything regarding concrete bugs. Nevertheless, we found a couple of repeating aspects.

- Perhaps unsurprisingly, non-deterministic algorithms like random forests or MLPs can only be compared using statistical significance. This seems to work well for the classifications using the χ^2 test. However, our results indicate that the scores should not be compared, as the differences are very often significant.
- Perhaps surprisingly, we often observe different classifications with deterministic algorithms, even to the point where we cannot state which algorithms are likely correct. For example, the small differences for GNB make it impossible to determine, without a detailed triage of all involved algorithms, to infer if any of the implementations contain a bug causing the small deviations, or if this is natural due to implementation choices. Most of the time, these differences are not statistically significant, which further raises the question if investigating these issues would be worth the effort.
- Scores often depend on the implementation. Even for a "simple" algorithm like LR, we observe significant differences. Thus, we cannot recommend to use differential testing for scores, as there is a very large risk of false positives.

[9] All algorithms in the groups LSVM, PSVM, RBFSVM, the algorithm ranger from the RF1 and RF2 groups, and the algorithm RidgeRegression from the RIDGE group.

Table 4 Overview of the results of the execution of the differential tests

Group	Summary of test results
GNB	Classes are equal or almost equal, with small differences for Weka. Scores are almost always different, except for the Caret-naive_bayes and Caret-nb, and the Spark MLlib and Scikit-learn implementations, which are equal. No differences are significant.
KDENB	Small but insignificant differences in classes and scores between the Caret-naive_bayes and Caret-nb. Large differences to Weka, that are significant for both classes and scores on about half of the data sets.
MNB	Classes and scores are equal, with one exception: the Scikit-learn implementation sometimes classifies one instance differently. These differences are in cases where the score is almost exactly 0.5 and for one framework slightly smaller and for the other framework slightly larger than 0.5, e.g., for Scikit-learn 0.499 and for Weka 0.501.
RF1	Results are never equal, with larger differences on test than on training data. The differences between classes are not significant, except in two cases on the RANDOM data, where the classes are significantly different. The differences between the scores are almost always significant.
RF2	Results are never equal, with larger differences on test than on training data. The differences of the classes are not significant, except in two cases on the RANDOM data. The scores between Weka and Caret-rboist is only significant once on the WINE data. The scores of Scikit-learn are significantly different most of the time.
LSVM	Results are never equal, with mostly small and insignificant differences. The exception is Spark MLlib, which has a large difference to the other implementations that are significant on the RANDOM and WINE data.
PSVM	Scikit-learn and Weka are equal. Caret yields different results, but these differences are only significant on the UNIFORM data.
RBFSVM	Scikit-learn and Weka are equal. Caret yields different results and these differences are almost always significant.
MLP	On the RANDOM and UNIFORM data, the classes are almost equal between all implementations with no significant difference. On the WINE and BC data there are large and significant differences, where Weka disagrees with the other frameworks. The differences between scores are always large and significant.
DUMMY	The classes are always equal, the scores depend on the implementation of the trivial model: Caret and Weka have the same approach, Scikit-learn always disagrees.
KNN	The classifications are equal or almost equal between all implementations. The scores of Scikit-learn and Caret are also equal or almost equal, with the exception of the WINE data. Here, Caret is equal to Weka instead. On the other data sets, Weka has a large and significant difference from the other implementations.
LR	The classes are equal or almost equal. The scores are almost always equal, except on the BC data, where we observe significant differences between all implementations.
RIDGE	The classes and scores of Weka, Scikit-learn-LogisticRegression, and Caret-plr are equal. The other implementations have small and insignificant differences for the classes and large and significant deviations for the scores.
LASSO	The classifications of Caret and Scikit-learn are equal or almost equal. Spark MLlib has large differences, but they are only significant on the RANDOM data. The scores between all implementations are different. The differences between Caret and Scikit-learn are mostly not significant, Spark MLlib is always significantly different.

We also note that our choice to use a threshold for the p-value without correction is not without consequences. For the Kolmogorov-Smirnoff tests between the scores, we have over 32% results in which we observe significant differences, i.e., far more than the expected 5% of false positives. Since there are too many differences between scores anyways, the

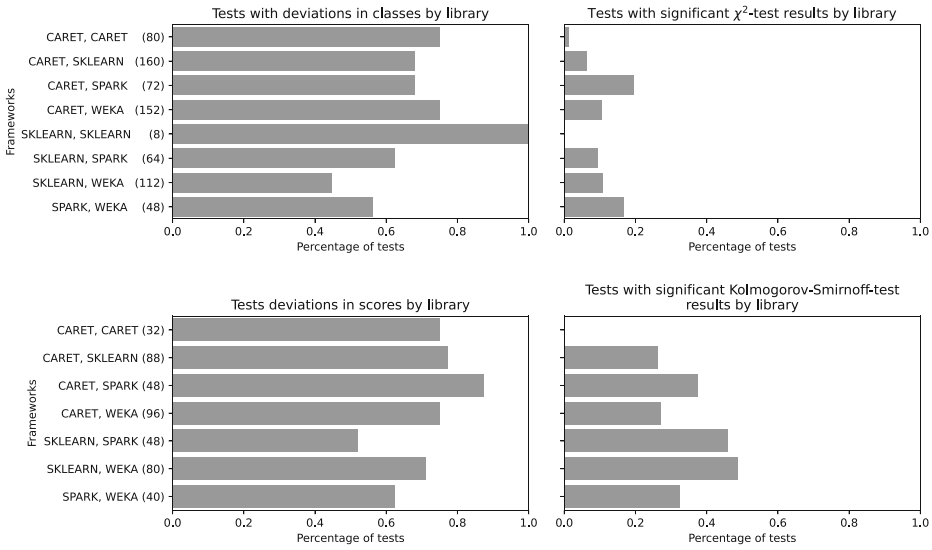

Fig. 3 Differences observed for pairs of libraries. The labels show the pair of libraries and the number of tests executed. For example, "CARET, SKLEARN (160)" means that we ran 160 tests where one classifier was from either framework

impact of possible false positives is limited. For the Chi-Squared, we have only about 10% significant difference. Consequently, the randomness of the p-value distribution, if the null hypothesis is true, would explain about half of the significant differences we observe. This adds another wrinkle to our already hard to interpret results. However, we would argue that for deterministic algorithms, only very small differences are acceptable, and anything large enough to be picked up as a significant signal by the Chi-Squared test actually points towards a significant difference between deterministic implementations. For randomized algorithms, a potential solution would be to use repeated tests, e.g., 100 different UNIFORM samples. One could then observe if the number of significant deviations is less than the expected false positive rate of 5%, which would indicate acceptable differences, or if it is large, which would indicate a significant difference.

Due to these issues, we believe that differential testing may be used by developers who already know the details of their implementation, to compare their work with other implementations. However, whether this is more effective and/or efficient than directly only debugging a single implementation is not clear without further study. Using differential testing as pseudo-oracle (Davis MD and Weyuker EJ 1981) to automatically drive test campaigns between tools seems to have only a limited potential.

> The results of the tests indicate that effective and efficient usage of differential testing between implementations is mostly not possible on an absolute level, i.e., expecting equal results. Tests for significant differences of classifications seem to be the only feasible option, but should only be used with proper care regarding false positives and require that testers are already experts for the implementations under test.

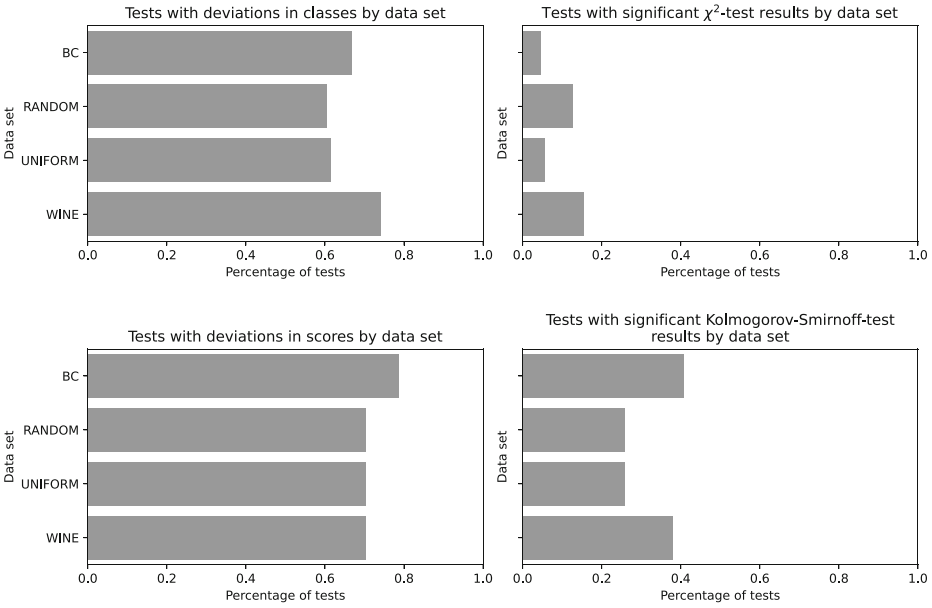

Fig. 4 Differences observed with the data sets we used

5 Discussion

Overall, our results show that while we have a large potential, it is very difficult to find feasible configurations of the algorithms. When we then execute these tests, we observe that there are many differences, to the point where we cannot distinguish between possible problems and normal deviations. We saw that especially the scores depend a lot on the implementation. Even if we just consider whether the binary classifications are equal, we observed too many differences. There is no reason to believe that this problem should not exist for more classes as well. While we expected that there would be some differences between the frameworks, the magnitude of these differences surprised us. We note that the data sets we use are neither very large, nor known to be numerically challenging.

Consequently, we believe that differential testing is just not possible between frameworks, unless a very lenient test oracle is used that avoids the large number of false positives, e.g., using the Chi-Squared tests. However, this comes at the cost of a larger chance of false negatives as well, because unexpected small differences between classes and any kind of difference between scores cannot be detected. If such a lenient oracle would be sufficient to detect bugs is, therefore, unclear. The cases where this would also be possible are rather the exception than the rule. We can only speculate regarding the reasons for this. However, we believe that there are four main drivers for the differences we observe:

– The definition of the algorithms in the literature may not be sufficiently detailed to guarantee that everything is implemented in exactly the same way. While we believe that this could sometimes be a reason, this cannot explain all differences we observe. For example, LSVM has a clear mathematical definition which is well-known and should lead to the same optimal result, except in few corner cases. In comparison, no two implementations yield exactly the same results.

- It is possible that the API documentations are incomplete and there are hidden values of hyperparameter that are hard-coded in the implementations but not visible through the documentation. However, in this case, we should see clearer patterns in the deviations, e.g., one algorithm deviating from the others on all data sets. While we sometimes observe this, there are relatively few cases where only one implementation disagrees with all others. Usually, many implementations disagree with each other within one group.
- Optimizations for the numerical stability, run-time, or memory efficiency could lead to subtle changes in the algorithms, which could explain the deviations between the implementations. The pattern of deviations should be the same as before, i.e., one algorithm disagrees with the others, which we often do not observe very often.
- The different programming languages, including the libraries available, could also have an impact. While, e.g., the different pairs for GNB indicate that this may be due to such an effect, we also observe many cases that indicate that this is not an issue. For example, the different R implementations that Caret offers often disagree with each other as well, even though they are, presumably, based on the same native R features. Hence, while this could be a factor, we believe that this only plays a minor role.

Unfortunately, we could only determine how often the above factors are the reason for differences through an extremely time-consuming manual analysis of the source code of the machine learning libraries, possibly even including libraries on which they are based, e.g., to understand the numerics involved. Such a large scale comparative audit is not possible with our resources, because we would basically need to achieve similar skills as the core developers for each of the four frameworks.

Regardless of our lack of evidence, we believe that all of the above reasons provide a reasonable explanation for the results of our experiments. We believe that each of the reasons is a driver for some of the cases in which we observe differences, especially those where we do not observe significant differences. Unfortunately, this also means that the differential testing is not effective as automated oracle, because we would need to invest a huge amount of effort to conduct an in-depth analysis of differences between most pairs of algorithms, which is not feasible. Consequently, this shows the limited capability of differential testing to provide an exact "specification" through a second implementation that provides a reliable oracle for deciding the correctness of software tests.

We also believe that our results are not contradicting that differential testing works well with deep learning frameworks, e.g., PyTorch (Paszke A et al. 2019) and TensorFlow (Abadi et al. 2015): exact equality is not expected, due to the randomized nature of the training, equality of scores is also not expected because the decision boundaries of neural networks can also easily slightly change with different initialization. Thus, there is a built-in uncertainty in the training of neural networks which means that the differential testing is only concerned with checking if the classification stays roughly the same, which we also found to work. While these differences might be made slightly worse by the implementation differences or similar as we describe above, they do not hinder the differential testing, because exact equality is not the expectation, anyways.

This is also in line with the fairly randomized algorithms we have within our data set, i.e., the random forest and the MLP. Thus, one can argue that the differential testing works beyond deep learning, if the algorithms are randomized and the only expectation is that the differences between classifications are not significant. However, subtle bugs that only affect a small amount of data can likely not be detected this way. For the algorithms in our data that are deterministic and often implement a formula or optimization approach, we

should be able to observe a stronger type of equality, i.e., exactly the same results or even scores, which is not the case due to the reasons listed above, which in the end mean that the specification of the algorithms is imprecise. It follows that differential testing requires either a relaxed view on the expected equality or stronger specifications of the behavior of machine learning algorithms. Consequently, we answer our research question as follows:

> While there is a large potential and it is possible to derive feasible tests, exhaustive testing of all parameter combinations or algorithms is not possible. More importantly, due to many false positive differences, the differential testing does not lead to a suitable automated test oracle if exact or almost exact equality is expected. For randomized algorithms, where some differences are expected anyways, differential testing can still be effective, but only if bugs cause large differences.

6 Threats to Validity

We report the threats to the validity of our work following the classification by Cook TD et al. (1979) suggested for software engineering by (Wohlin et al. 2012). Additionally, we discuss the reliability as suggested by Runeson and Höst (2009).

6.1 Construct Validity

There are crucial design choices in each phase of our case study that could affect our results. We describe these threats below, including why we believe that the alternatives would not lead to a better construct.

The construct of the first phase of our study is based on a coarse grouping of algorithms by the general approach, e.g., grouping decision trees together. While we have a good reason for this, i.e., that some algorithms have different variants in one implementation, while others have these variants over multiple implementations, this construct may lead to an overestimation of the potential. An indicator that this happens to some degree is the decision tree, where we have a huge group for the potential, but no feasible variants. The alternative design choice would have been a more fine-grained grouping (i.e., by concrete algorithm variant) and then possibly repeating the same implementation multiple times. However, we believe that this option has one major drawback, i.e., it would lead to merging aspects of the second phase of our study with the first phase. The more fine-grained grouping is only possible by studying the hyperparameters through the API documentation, which is precisely what we do in the second phase, where we not only achieve a more fine-grained grouping that accounts for algorithm variants, but at the same time check if the variants are sufficiently similar to provide equal sets of hyper parameters.

The construct of the second phase of our study is based on studying the API documentation of algorithms to find equal configurations. While we believe that this is the only practically feasible choice, there is a certain risk involved with relying on the documentation. If the documentation does not document all parameters or fails to describe all relevant aspects, we may miss feasible candidates. The alternative would be to directly check the source code to understand all options (incl. hard-coded choices). However, this would essentially be the same as an in-depth code review of the implementations under test, which we

believe is not realistic, neither for us as researchers as part of a study, nor for practitioners as part of a test campaign.

The construct of the third phase of our study is based on four different comparisons that consider equality of scores and classes, as well as significance of differences. While we believe that these are reasonable criteria to determine if a differential test fails, other options might also work, e.g., considering the mean absolute deviation of scores. However, such an approach has the drawback that a threshold is required that defines how much deviation is acceptable. We believe that our approach based on statistical tests achieves a similar result, but with a stronger mathematical foundation.

We could also have used a different approach for the statistical analysis. For example, instead of using the χ^2 test to compare statistical differences between implementations, we could also have executed each implementation multiple times with the same data, but different training and test splits. Then, we could have statistically compared the accuracy on these splits, e.g., with Wilcoxon's signed rank test (Wilcoxon F 1945), if we use the same set of splits for each implementation or with the Mann-Whitney U test (Mann HB and Whitney DR 1947) if we would use different splits. Possibly, such an approach could be more tolerant to noise due to differences in how randomized components are implemented in different frameworks. However, such an approach would have the disadvantage that our experiment would be sensitive to the number of different train-test splits we use, as this constitutes the sample size for the comparison of the accuracy. This could be counterbalanced by also including effect sizes (e.g., Cliff's delta (Cliff N 1993)) or by using a Bayesian signed rank test with a Region of Practical Equivalence (ROPE) (Benavoli A et al. 2016). But again, we would need to determine relevant effect sizes for this, which has a similar risk. Consequently, we believe that our choice of statistical methods is suitable for our study, but cannot rule out that other statistical approaches for differential testing may lead to different results.

Another potential issue with our construct is that implementations may also randomly lead to unstable results, even with the same data and hyperparameters (including a fixed random seed). To account for this, we asserted that the results remain the same with ten randomly generated data sets for all algorithms. We did not observe any deviations between the classes and the scores.

6.2 Internal Validity

We do not believe that there are threats to our conclusions regarding the potential, feasibility, or effectiveness, because these conclusions are directly derived from our quantitative evidence. However, our discussion regarding potential reasons is not derived from this evidence, but rather from our knowledge about software development and machine learning. Consequently, we may overestimate or underestimate the issues we discuss as possible reasons in Section 5 and also miss other reasons. However, we clearly state this limitation to be transparent to readers of this article. The only feasible alternative would be not to discuss the reasons for our results at all, which we believe would be a larger problem than possible wrong speculative reasoning, if this is clearly marked as such.

6.3 External Validity

While we consider four machine learning libraries in our study, it is unclear if the results would hold for other libraries as well or if a larger effectiveness with more feasible combinations would be possible. However, we do not believe that this is likely, because we already

considered large and mature libraries. Additionally, caret is a meta-framework that means we actually included many different R libraries in our study.

We also cannot be certain how our results generalize to other types of machine learning, e.g., clustering or regression. However, we note that these algorithms are similar in their structure to classification algorithms. For example, we see no reason why there should be notable differences in the testability between a random forest regression and a random forest classifier or between a k-Nearest Neighbor classifier and a k-Means clustering. Thus, we believe that our results provide a strong indication for the expectations beyond classification, i.e., that exactly the same results usually cannot be expected, but a more coarse comparison could be possible.

6.4 Reliability

A significant amount of our work was the manual analysis of the APIs of four machine learning frameworks we studied carried out together by the authors. We cannot rule out the possibility that we missed algorithms (phase 1) or feasible combinations (phase 2). We believe that the threat for the first phase is negligible, because all the API documentation of all frameworks specifically contains categories for supervised and/or classification algorithms. For the second phase, we read all API docs and all parameters as pairs to minimize the likelihood of missing aspects. Moreover, the results of the second phase were validated by implementing the feasible combinations later on, which required us to double check all results. Thus, while we may have missed something, we do not think that it is likely that we missed so many algorithms or feasible combinations that this would have an effect on our conclusions.

7 Conclusion

Within this paper, we evaluate the potential, feasibility, and effectiveness of differential testing for classification algorithms beyond deep learning. While we found that there is a huge potential, especially for popular algorithms, we found that it was already difficult to identify feasible tests such that the API documentation indicated that the behavior of two implementations should be the same. When we then executed the feasible tests, we found so many differences between the results, that we could not find a signal in the noise, i.e., identify true positive differences that indicate actual bugs in an implementation among all the false positive tests results with differences due to other reasons. However, our results indicate that for experts it may still be possible to use a relatively lenient approach based on significant differences between classification results to determine if there are bugs within a software, but it is unclear if such an oracle would be sufficiently powerful to detect bugs. However, the number of false positives seems to be too large to be useful for researchers as automated pseudo-oracle to evaluate the effectiveness of testing approach for such algorithms.

Appendix: Additional Results

Within this appendix, we report additional results for all phases of the case study. Table 5 shows a complete list of algorithms from all frameworks and how they overlap. Table 6 shows how the hyperparameter must be configured for all feasible combinations we

Table 5 Overlapping algorithms between frameworks based on a scan of algorithm names in the API docs

Scikit-learn	Weka	Spark MLlib	Caret
BernoulliNB, ComplementNB, MultinomialNB, GaussianNB, CategoricalNB	NaiveBayesMultinomial, NaiveBayes	NaiveBayes	awnb, naive_bayes, nb, manb
DecisionTreeClassifier	SimpleCart, J48, DecisionStump, HoeffdingTree	DecisionTreeClassifier	C5.0Tree, C5.0Rules, rpartCost, C5.0Cost, rpartScore, rpart, rpart1SE, rpart2, C5.0, ctree, ctree2, bartMachine
DecisionTreeClassifier	SimpleCart, J48, DecisionStump, HoeffdingTree	DecisionTreeClassifier	C5.0Tree, C5.0Rules, rpartCost, C5.0Cost, rpartScore, rpart, rpart1SE, rpart2, C5.0, ctree, ctree2, bartMachine
RandomForestClassifier	RandomForest	RandomForestClassifier	wsrf, RRF, RRFglobal, rfRules, ordinalRF, ranger, Rborist, rf, ORFlog, ORFpls, ORFridge, ORFsvm, cforest
DecisionTreeClassifier	SimpleCart, J48, DecisionStump, HoeffdingTree	DecisionTreeClassifier	C5.0Tree, C5.0Rules, rpartCost, C5.0Cost, rpartScore, rpart, rpart1SE, rpart2, C5.0, ctree, ctree2, bartMachine
RandomForestClassifier	RandomForest	RandomForestClassifier	wsrf, RRF, RRFglobal, rfRules, ordinalRF, ranger, Rborist, rf, ORFlog, ORFpls, ORFridge, ORFsvm, cforest
LinearSVC, NuSVC, SVC	SMO	LinearSVC	svmSpectrumString, svmRadial, svmRadialCost, svmRadialSigma, svmPoly, svmLinear, svmLinear2, svmExpoString, svmRadialWeights, svmBoundrangeString, svmLinearWeights, lssvmRadial, lssvmPoly, lssvmLinear, svmLinear3, svmLinearWeights2
MLPClassifier	MultilayerPerceptron	MultilayerPerceptronClassifier	mlpKerasDecayCost, mlpKerasDecay, mlpKerasDropoutCost, mlpKerasDropout, mlpSGD, mlpML, mlpWeightDecayML, mlp, mlpWeightDecay, monmlp

Table 5 (continued)

Scikit-learn	Weka	Spark MLlib	Caret
LogisticRegression, RidgeClassifier	Logistic, SimpleLogistic	LogisticRegression	regLogistic, plr, polr
DummyClassifier	ZeroR		null
KNeighborsClassifier	IBk		snn, kknn, knn
GradientBoostingClassifier		GBTClassifier	gbm_h2o, gbm
Perceptron	VotedPerceptron		
SGDClassifier	SGD		
GaussianProcessClassifier			gaussprRadial, gaussprPoly, gaussprLinear
LinearDiscriminantAnalysis			slda, sparseLDA, sda, rrlda, Linda, rlda, rda, PenalizedLDA, pda, pda2, Mlda, loclda, stepLDA, lda, lda2, RFlda, dda
NearestCentroid			pam
QuadraticDistriminantAnalysis			qda, QdaCov, stepQDA
ExtraTreeClassifier			extraTrees
	LMT		LMT
PassiveAgressiveClassifier			
RadiusNeighborsClassifier			
	RandomTree		
	REPTree		
	DecisionTable		
	Ripper		
	OneR		
	PART		
	KStar		

Table 5 (continued)

Scikit-learn	Weka	Spark MLlib	Caret
	BayesNet		nbDiscrete
			spls, kernelpls, pls, simpls, widekernelpls
			gpls, plsRglm
			dnn, pcaNNet, mxnet, mxnetAdam, nnet, avNNet, multinom
			hda, binda, fda, hdda, hdrda, mda, smda, rmda, amdai
			sdwd, dwdLinear, dwdRadial, dwdPoly
			xgbDART, xgbLinear, xgbTree
			elm
			deepboost
			glm, glmStepAIC, randomGLM, bayesglm
			gam, bam, gamSpline, gamLoess
			glmnet, glmnet_h2o
			FRBCS.W, SLAVE, FH.GBML, FRBCS.CHI
			rotationForest, rotationForestCp
			rbf, rbfDDA
			earth, gcvEarth
			CSimca, RSimca
			tan, tanSearch, awtan
			vbmpRadial
			ownn
			PRIM

Table 5 (continued)

Scikit-learn	Weka	Spark MLlib	Caret
			nodeHarvest
			evtree
			nbSearch
			xyf
			rocc
			rFerns
			ordinalNet
			partDSA
			msaenet
			lvq
			protoclass
			vglmCumulative
			vglmContRatio
			vglmAdjCat
			chaid
			logreg

Table 6 Feasible combinations with at least three implementations

Group	Scikit-learn	Weka	Spark MLlib	Caret
GNB	GaussianNB	NaiveBayes	NaiveBayes modelType=gaussian	naive_bayes usekernel=FALSE adjust=1 laplace=0
				nb usekernel=FALSE adjust=1 laplace=0
KDENB		*NaiveBayes* -K		*naive_bayes* usekernel=TRUE adjust=1 laplace=0
				nb usekernel=TRUE adjust=1 laplace=0
MNB	*MultinomialNB*	*NaiveBayesMultinomial*	*NaiveBayes* modelType=multinomial	
RF1	*RandomForestClassifier* n_estimators=x_1 max_features=x_2 max_depth=x_3	*RandomForest* -I x_1 -K x_2 -depth x_3		*ranger* num.trees=x_1 mtry=x_2 max.depth=x_3 splitrule=gini min.node.size=1
				rborist nTree=x_1 predFixed=x_2 nLevel=x_3 minNode=1
RF2	*RandomForestClassifier* n_estimators=x_1 max_features=x_2	*RandomForest* -I x_1 -K x_2		*ranger* num.trees=x_1 mtry=x_2 splitrule=gini min.node.size=1
				rborist nTree=x_1 predFixed=x_2 minNode=1
				rf ntree=x_1

Table 6 (continued)

Group	Scikit-learn	Weka	Spark MLlib	Caret
				mtry=x_2
LSVM	*SVC*	*SMO*	*LinearSVC*	*svmLinear*
	C=x_1	C x_1	regParam=x_1	tau=x_1
	kernel=linear	-k PolyKernel*		
				tol=0.001
				svmLinear2
				cost=x_1
				svmLinear3
				cost=x_1
				epsilon=0.001
				Loss=L2
PSVM	*SVC*	*SMO*		*svmPoly*
	C=x_1	C x_1		C=x_1
	degree=x_2	-E x_2		degree=x_2
	kernel=poly	-k PolyKernel*		
	gamma=1			scale=1
				tol=0.001
RBFSVM	*SVC*	*SMO*		*svmRadial*
	C=x_1	C x_1		C=x_1
	gamma=x_2	-G x_2		sigma=x_2
	kernel=rbf	-k RBFKernel*		
				tol=0.001
MLP	*MLPClassifier*	*MultilayerPerceptron*	*MultilayerPerceptronClassifier*	*mlp**
	hidden_layer_sizes=x_1	-h x_1	layers=x_1	size=x_1
	learning_rate_init=x_2	-L x_2	stepSize=x_2	learnFuncParams=c (x_1,0)
	max_iter=x_3	-N x_3	maxIter=x_3	maxit=x_3
	momentum=0.0	-M 0		
	solver=sgd		solver=gd	
	alpha=0.0			
	activation=logistic			
		-I		
				*mlpSGD**
				size=x_1
				learn_rate=x_2
				max_epochs=x_3
				momentum=0
				l2reg=0
				gamma=0
				lambda=0
				repeats=1

Table 6 (continued)

Group	Scikit-learn	Weka	Spark MLlib	Caret
DUMMY	*DummyClassifier* strategy=most_frequent	*ZeroR*		*null*
KNN	*KNeighborsClassifier* n_neighbors=x_1	*IBk* -K x_1		*knn* k=x_1
LR	*LogisticRegression*	*Logistic* -R 0	*LogisticRegression* regParam=0	*regLogistic* cost=0
	max_iter=10000	-M 10000 -S	maxIter=10000	
	penalty=none			loss=L2_primal epsilon=0.0001
				plr lambda=0 cp=bic
				polr method=logistic
RIDGE	*LogisticRegression* C=$1/(2 \cdot x_1)$ max_iter=10000	*Logistic* -R x_1 -M 10000 -S	*LogisticRegression* regParam=x_1 maxIter=10000	*regLogistic* cost=x_1
	penalty=l2			loss=L2_primal epsilon=0.0001
	RidgeClassifier alpha=x_1 max_iter=10000			*plr* lambda=x_1 cp=bic
LASSO	*LogisticRegression* C=$1/x_1$ max_iter=10000 penalty=l1		*LogisticRegression* regParam=x_1 maxIter=10000 elasticNetParam=1	*regLogistic* cost=x_1 loss=L1 epsilon=0.0001

Algorithm names are italic followed by the required parameters. Parameters in the same row have the same meaning. If parameters are not specified for an algorithm, this means that the API documentation indicates that the default is the same as what is achieved by using the parameters for the other algorithms. The parameter values x_i indicate that the parameter is configurable with the same meaning in all implementations.

identified. Table 7 provides additional details for the results of the execution of the differential tests. Figure 3 shows how often we observed differences between implementations grouped by the framework they were implemented in. Figure 4 shows how often we observed differences grouped by the data set we used to execute the tests. Figures 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17 and 18 show for each algorithm we tested the values of Δ and Δ_{score}, normalized by the number of instances in the data set n. To better show the

Table 7 Feasible combinations with at least three implementations

Group	Test results
GNB	The classes of all implementations are equal, except Weka, which yields different classes on up to 2% of the instances on the RANDOM, WINE, and BC data. These differences are not significant. The scores of the two pair naive_bayes and nb, as well as the pair Scikit-learn Spark MLlib are equal. For all other pairs, we often observe large numbers of deviations between the scores on up to 89% of the instances, both on the test and training data. However, these differences are also not significant, i.e., we observe the same distributions when all instances are considered, but the scores of the individual instances depend on the implementation.
KDENB	We observe differences between all implementations. The differences between the R packages naive_bayes and nb are small, with respect to Δ, i.e, we observe only few differences in classifications with at most 1%. However, we observe big differences in scores, on both training and test data. However, none of these differences are significant. The differences to the Weka implementation are large: we never observed the exact same classification with Δ showing differences on between 3% and 27% of the data. These differences are significant in about half of the time, i.e., on the WINE data and the test date of UNIFORM. We also observe large differences in the scores, including cases where every single score is different. Similar to the classifications, the differences in scores are significant one the WINE data, the test data of UNIFORM, and, in addition, the test data of RANDOM, but only between the nb package and WEKA.
MNB	We did not observe any differences between the Weka and Spark MLlib implementation. We also did not observe any differences in the scores between all three implementations. However, the Scikit-learn implementation classifies one instance differently in ten of 21 comparisons. These differences are in cases where the score is almost exactly 0.5 and for one framework slightly smaller and for the other framework slightly larger than 0.5, e.g., for Scikit-learn 0.499 and for Weka 0.501. In this case, we do not consider the scores different, but the classes are different regardless.
RF1	The results of the random forests with a fixed depth of five are never exactly the same. Between 0.7% and 40% of classifications are different. We note that the differences are larger on the test data than on the training data. The differences between the two R packages and Scikit-learn are similar to each other. The deviations between these three implementations and Weka are about twice as large. However, these differences between the classifications are mostly not significant, with two exceptions on the RANDOM data, where ranger deviates significantly from Scikit-learn and weka. For the scores, we observe that the Weka and Rborist have deviations on between 28% and 82% of the instances. However, these large differences are only significant on the training data of WINE. The scores of Weka imlementation and Rborist have deviations from Scikit-learn on between 52% and 100% of the instances. These differences are almost always significant. We note that package ranger does not support scores.
RF2	The results of the random forest without a fixed depth are also almost never exactly the same, with two exceptions, once for Scikit-learn and the R package ranger, and once for Scikit-learn and the R package rf. Otherwise, there are differences for up to 38% of the instances. Same as for the random forest with fixed depth, the differences are larger on the test data. However, in most cases, these differences are not significant, except on the WINE data, where ranger is significantly different from all other implementations. For the scores differences on between 23% and 87% of the instances. However, none of these differences are significant, i.e., while the scores are different, their distributions are not.
LSVM	None of the linear SVMs implementations lead to equal results. The svmlinear and svmlinear2 from Caret are almost equal, with only a single instance that is classified differently on the WINE data. The Scikit-learn and Weka implementations are also almost equal with two single instance misclassified on the test data of UNIFORM and WINE. The differences between most other pairs of implementations are similar, with between 1% and 11% of the instances deviating from each other. The clear outlier is the Spark MLlib implementation, which yields a different class on 38% to 44% of the instances on the RANDOM data, and also differences on 12% to 16% of the WINE instances. These outliers of Spark MLlib are the only significant deviations of the linear SVM.

Table 7 (continued)

Group	Test results
PSVM	The Scikit-learn and Weka implementations of the polynomial SVM are equal. The Caret implementation yields different classifications for 17% to 27% of the instances. However, this difference is only significant on the UNIFORM data.
RBFSVM	The Scikit-learn and Weka implementations of the RBF SVM are equal. The care implementation yields different classifications for 6% to 52% of the instances. This difference is always significant, except training data of BC.
MLP	All three MLPs lead to the same classifications on the RANDOM data. The Scikit-learn and Weka implementations are also also equal on the UNIFORM data. The Scikit-learn and Spark MLlib implementation classify one instance differently on the UNIFORM data. However, they are equal on the BC and WINE data. The difference of Scikit-learn and Spark MLlib to Weka is on the BC and WINE data is significant, with differences on between 34% and 51% of the instances. The scores of all three implementations are significantly different from each other, with deviations on between 53% and 100% of the instances.
DUMMY	The trivial classifiers always have the same classes. The scores depend on the implementation of the trivial model: Caret and Weka have the same approach and always yield the same results. Scikit-learn uses a different approach and disagrees with the two other scores. These disagreements are significant.
KNN	The nearest neighbor algorithms have the same classes, except on the WINE data, where about 1% of the instances are classified differently between all combinations of frameworks. These differences are not significant. The scores of Scikit-learn and Caret are also equal, except on the WINE data, where 5% of the instances have different scores. This difference is significant. On WINE, the scores of Caret are equal to those of Weka. On the other data sets, Weka has significantly different scores from Caret and Scikit-learn, with 91% to 100% of instances receiving different scores.
LR	The logistic regressions have the same classes, except on the test data of BC, where about 3% of the instances are classified differently. These differences are not significant. Similarly, the scores are always equal for all implementations on the UNIFORM, RANDOM, and WINE data, as well as for Scikit-learn, Weka, and Spark MLlib on the BC training data. We observe differences on between 3% and 9% of the instances on the remaining tests on the BC data. The differences between the scores are significant.
RIDGE	The Weka, the LogisticRegression from Scikit-learn, and the Caret model plr yield identical for both classifications and scores results. The differences between the classes predicted by the other pairs of implementations are between 0.3% and 4%, i.e., relatively small and not statistically significant. The scores are inconsistent and have large and significant deviations of on between 65% and 99% of the instances. The RidgeClassifier of Scikit-learn does not compute scores.
LASSO	The Caret and Scikit-learn implementation yield almost the same classes, with up to three instances classified differently. These differences are not significant. The differences to Spark MLlib are large and between 4% and 43% of instances are classified differently. However, these differenes are only significant on the RANDOM data. The scores are different for between 67% and 98% of the instances for Caret and Scikit-learn. However, the difference is only significant on the training data of BC. The scores of Spark MLlib are different on at least 99% and 100% of the instance, i.e., almost always. These differences are significant.

Algorithm names are italic followed by the required parameters. Parameters in the same row have the same meaning. If parameters are not specified for an algorithm, this means that the API documentation indicates that the default is the same as what is achieved by using the parameters for the other algorithms. The parameter values x_i indicate that the parameter is configurable with the same meaning in all implementations

range of differences we observed, the Figures also contain violin plots that show the distribution of the deviations. Figures 19 and 20 show the relationship of the accuracy of the predictions and the values of Δ and Δ_{score} to provide further insights into both the performance that we observed during our experiments, as well as the (lack of) a relationship of this with our data and algorithms.

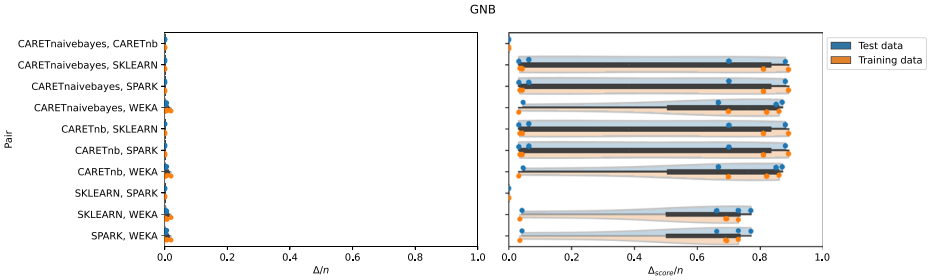

Fig. 5 Deviations observed with GNB

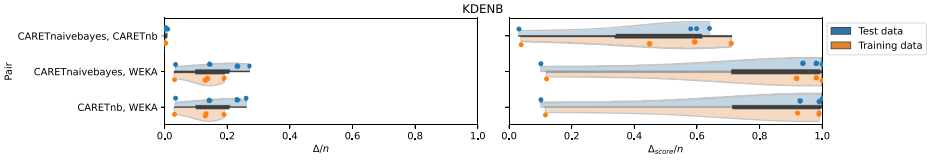

Fig. 6 Deviations observed with KDENB

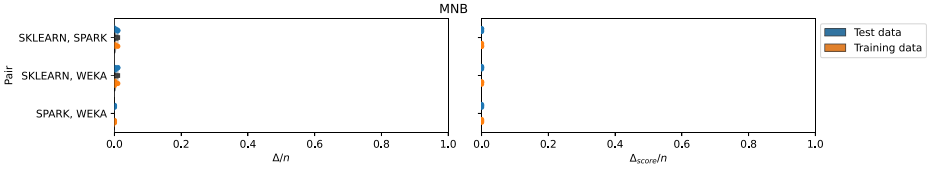

Fig. 7 Deviations observed with MNB

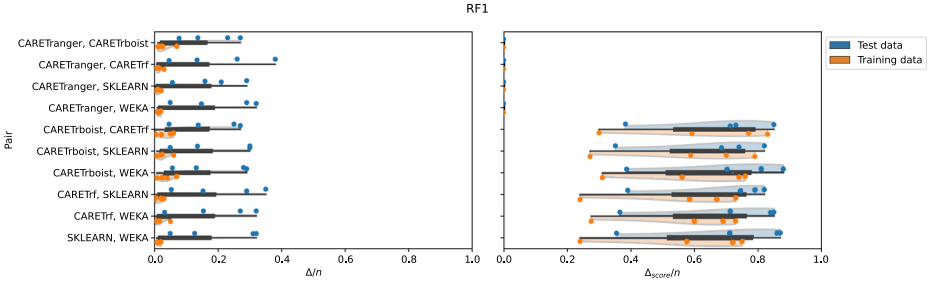

Fig. 8 Deviations observed with RF1

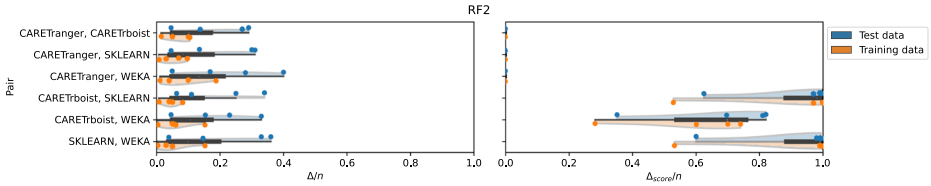

Fig. 9 Deviations observed with RF2

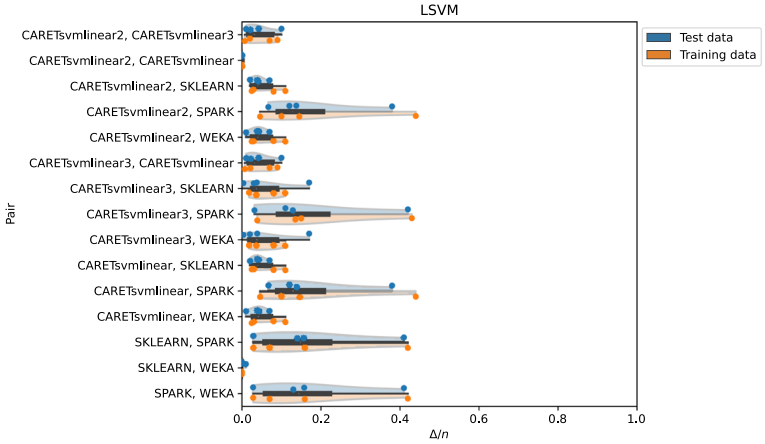

Fig. 10 Deviations observed with LSVM

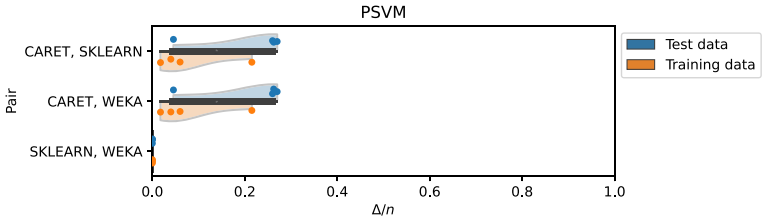

Fig. 11 Deviations observed with PSVM

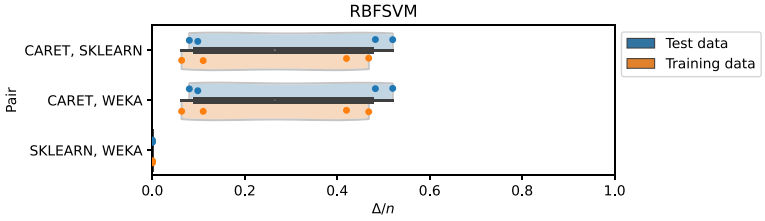

Fig. 12 Deviations observed with RBFSVM

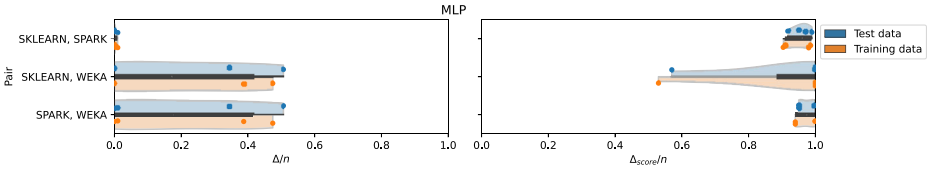

Fig. 13 Deviations observed with MLP

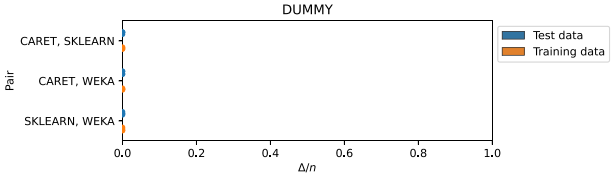

Fig. 14 Deviations observed with DUMMY

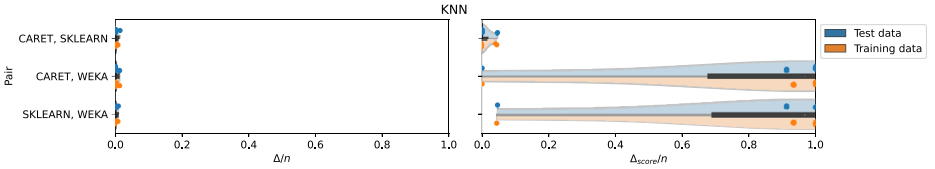

Fig. 15 Deviations observed with KNN

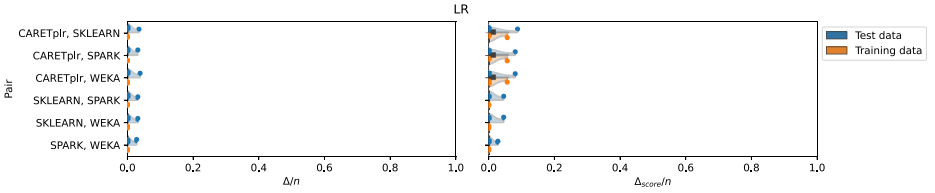

Fig. 16 Deviations observed with LR

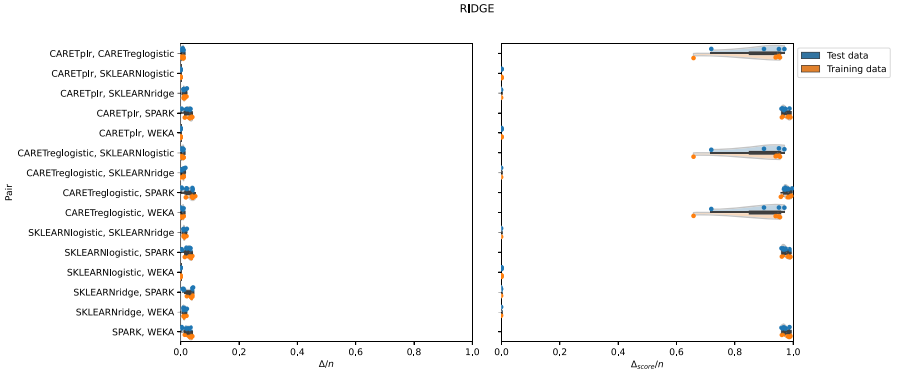

Fig. 17 Deviations observed with RIDGE

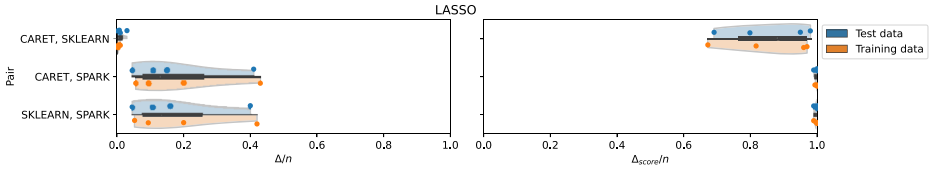

Fig. 18 Deviations observed with LASSO

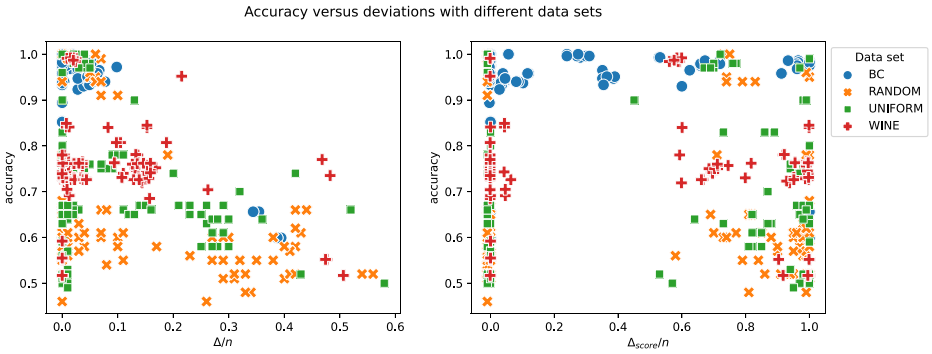

Fig. 19 Relationship of accuracy and the deviations observed by data set

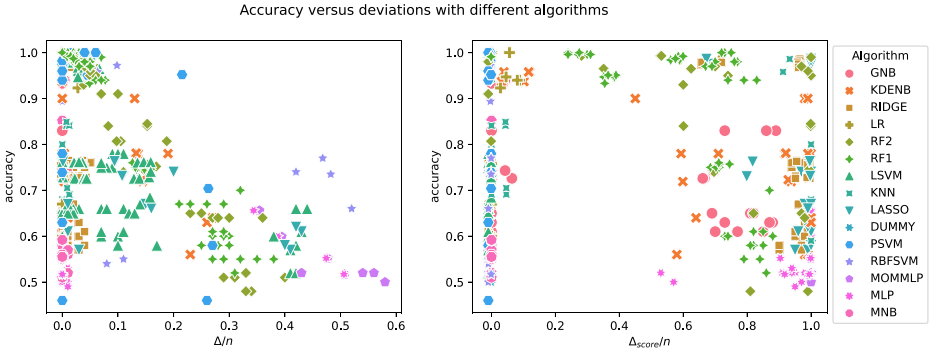

Fig. 20 Relationship of accuracy and the deviations observed by algorithm

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability Statement The datasets generated during and/or analysed during the current study are available in the Zenodo repository, https://doi.org/10.5281/zenodo.7341091.

References

Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2015) TensorFlow : large-scale machine learning on heterogeneous systems https://www.tensorflow.org/softwareavailablefromtensorflow.org

Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, Kudlur M, Levenberg J, Monga R, Moore S, Murray DG, Steiner B, Tucker P, Vasudevan V, Warden P, Wicke M, Yu Y, Zheng X (2016) Tensorflow: a system for large-scale machine learning. In: Proceedings of the 12th USENIX conference on operating systems design and implementation. OSDI'16, USENIX Association, Berkeley, pp 265–283. http://dl.acm.org/citation.cfm?id=3026877.3026899

Asyrofi MH, Thung F, Lo D, Jiang L (2020) Crossasr: efficient differential testing of automatic speech recognition via text-to-speech

Barash G, Farchi E, Jayaraman I, Raz O, Tzoref-Brill R, Zalmanovici M (2019) Bridging the gap between ml solutions and their business requirements using feature interactions. In: Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering. Association for Computing Machinery, New York, ESEC/FSE 2019, pp 1048–1058 . https://doi.org/10.1145/3338906.3340442

Benavoli A, Corani G, Demsar J, Zaffalon M (2016) Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis. ArXiv:1606.04316

Braiek HB, Khomh F (2020) On testing machine learning programs. J Syst Softw 164:110542. https://doi.org/10.1016/j.jss.2020.110542

Breiman L (1996) Bagging predictors. Mach Learn 24:123–140.

Breiman L (2001) Random forests. Mach Learn 45(1):5–32. https://doi.org/10.1023/a:1010933404324

Brieman L, Friedman J, Olshen R, Stone C (1984) Classification and regression trees. Wadsworth, Belmont

Cliff N (1993) Dominance statistics: ordinal analyses to answer ordinal questions. Psychol Bull 114(3):494–509. https://doi.org/10.1037/0033-2909.114.3.494

Cook TD, Campbell DT, Day A (1979) Quasi-experimentation: design & analysis issues for field settings, vol 351. Houghton Mifflin, Boston

Davis MD, Weyuker EJ (1981) Pseudo-oracles for non-testable programs. In: Proceedings of the ACM '81 conference, ACM, New York, NY, USA, ACM '81, pp 254–257, https://doi.org/10.1145/800175.809889

Ding J, Kang X, Hu XH (2017) Validating a deep learning framework by metamorphic testing. In: 2017 IEEE/ACM 2nd international workshop on metamorphic testing (MET), pp 28–34, 10.1109/MET.2017.2

Frank E, Hall MA, Witten IH (2016) The WEKA workbench online appendix for Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann

Freund Y, Schapire RE (1997) A decision-theoretic generalization of on-line learning and an application to boosting. J Comput Syst Sci 55(1):119 – 139. https://doi.org/10.1006/jcss.1997.1504

Giray G (2021) A software engineering perspective on engineering machine learning systems: state of the art and challenges. J Syst Softw 180:111031. https://doi.org/10.1016/j.jss.2021.111031

Groce A, Kulesza T, Zhang C, Shamasunder S, Burnett M, Wong WK, Stumpf S, Das S, Shinsel A, Bice F, McIntosh K (2014) You are the only possible oracle: effective test selection for end users of interactive machine learning systems. IEEE Trans Softw Eng 40(3):307–323. https://doi.org/10.1109/TSE.2013.59

Gross P, Boulanger A, Arias M, Waltz D, Long PM, Lawson C, Anderson R, Koenig M, Mastrocinque M, Fairechio W, Johnson JA, Lee S, Doherty F, Kressner A (2006) Predicting electricity distribution feeder failures using machine learning susceptibility analysis. In: Proceedings of the 18th conference on innovative applications of artificial intelligence, vol 2. AAAI Press, IAAI'06, pp 1705–1711 http://dl.acm.org/citation.cfm?id=1597122.1597127

Guo Q, Xie X, Li Y, Zhang X, Liu Y, Li X, Shen C (2020) Audee: automated testing for deep learning frameworks. In: 2020 35th IEEE/ACM international conference on automated software engineering (ASE), pp 486–498

Guo J, Zhao Y, Song H, Jiang Y (2021) Coverage guided differential adversarial testing of deep learning systems. IEEE Trans Netw Sci Eng 8(2):933–942. https://doi.org/10.1109/TNSE.2020.2997359

Herbold S, Haar T (2022) Smoke testing for machine learning: simple tests to discover severe bugs. Empir Softw Eng 27(2). https://doi.org/10.1007/s10664-021-10073-7

ISO/IEC/IEEE (2017) Iso/iec/ieee international standard - systems and software engineering–vocabulary. ISO/IEC/IEEE, 24765:2017(E) pp 1–541. https://doi.org/10.1109/IEEESTD.2017.8016712

Karpathy A (2018) Cs231n: convolutional neural networks for visual recognition. https://cs231n.github.io/neural-networks-3/

Kuhn M (2018) Caret: classification and regression training. https://CRAN.R-project.org/package=caret, rpackageversion6.0-80

Mann HB, Whitney DR (1947) On a test of whether one of two random variables is stochastically larger than the other. Ann Math Stat 18(1):50 – 60. https://doi.org/10.1214/aoms/1177730491

Marijan D, Gotlieb A (2020) Software testing for machine learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol 34. pp 13576-13582

Martínez-Fernández S, Bogner J, Franch X, Oriol M, Siebert J, Trendowicz A, Vollmer AM, Wagner S (2021) Software engineering for ai-based systems: a survey. ACM Trans Softw Eng Methodol

McCullough BD, Mokfi T, Almaeenejad M (2019) On the accuracy of linear regression routines in some data mining packages. WIREs Data Min Knowl Disc 9(3):e1279. https://doi.org/10.1002/widm.1279

Meng X, Bradley J, Yavuz B, Sparks E, Venkataraman S, Liu D, Freeman J, Tsai D, Amde M, Owen S et al (2016) Mllib: machine learning in apache spark. J Mach Learn Res 17(1):1235–1241

Murphy C, Kaiser GE, Arias M (2007) An approach to software testing of machine learning applications. In: SEKE, vol 167

Murphy C, Kaiser GE, Hu L, Wu L (2008) Properties of machine learning applications for use in metamorphic testing. In: SEKE, vol 8. pp 867-872

Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S (2019) Pytorch: an imperative style, high-performance deep learning library. In: Wallach H, Larochelle H, Beygelzimer A, d'Alché-Buc F, Fox E, Garnett R (eds) advances in neural information processing systems 32, Curran Associates, Inc, pp 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Patton MQ (2014) Qualitative research & evaluation methods: integrating theory and practice. Sage Publications

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 (2011) Scikit-learn: machine learning in Python. J Mach Learn Res 12:2825–2830

Pei K, Cao Y, Yang J, Jana S (2019) Deepxplore: automated whitebox testing of deep learning systems. Commun ACM 62(11):137–145. https://doi.org/10.1145/3361566

Pham HV, Lutellier T, Qi W, Tan L (2019) Cradle: cross-backend validation to detect and localize bugs in deep learning libraries. In: 2019 IEEE/ACM 41st international conference on software engineering (ICSE), pp 1027–1038. https://doi.org/10.1109/ICSE.2019.00107

Quinlan JR (1986) Induction of decision trees. Mach Learn 1(1):81–106

Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc, San Francisco

Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng 14(2):131–164

Theano Development Team (2016) Theano: a Python framework for fast computation of mathematical expressions. arXiv:1605.02688

Tunkel S, Herbold S (2022) Replication Kit for: differential testing for machine learning: an analysis for classification algorithms beyond deep learning. https://doi.org/10.5281/zenodo.7341092

Wang Z, Yan M, Chen J, Liu S, Zhang D (2020) Deep learning library testing via effective model generation. In: Proceedings of the 28th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering, Association for Computing Machinery, New York, NY, USA, ESEC/FSE 2020, pp 788–799. https://doi.org/10.1145/3368089.3409761

Wilcoxon F (1945) Individual comparisons by ranking methods. Biometrics Bulletin 1(6):80–83. http://www.jstor.org/stable/3001968
Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslen A (2012) Experimentation in Software Engineering. Springer Publishing Company, Incorporated
Xie X, Ho JW, Murphy C, Kaiser G, Xu B, Chen TY (2011) Testing and validating machine learning classifiers by metamorphic testing. J Syst Softw 84(4):544–558,. https://doi.org/10.1016/j.jss.2010.11.920
Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX conference on hot topics in cloud computing (HotCloud)
Zhang JM, Harman M, Ma L, Liu Y (2020) Machine learning testing: survey , landscapes and horizons. IEEE Trans Softw Eng, pp 1–1

Steffen Herbold is professor and Chair of AI Engineering at the University of Passau, Germany. His research is focused on the responsible and fair solution of problems with machine learning, the quality assurance of applications using machine learning, and their operation.

Steffen Tunkel is currently working as a software engineer at Evotec. He received his M.Sc. degree in computer science from the University of Goettingen and his B.Eng. degree in electrical engineering from the HAWK Hildesheim/Holzminden/Göttingen.