Property-Driven Black-Box Testing of Numeric Functions

Arnab Sharma[1], Vitalik Melnikov[2], Eyke Hüllermeier[3], Heike Wehrheim[4]

Abstract:

In this work, we propose a *property-driven* testing mechanism to perform unit testing of functions performing numerical computations. Our approach, similar to the property-based testing technique, allows the tester to specify the requirements to check. Unlike property-based testing, the specification is then used to generate test cases in a targeted manner. Moreover, our approach works as a *black-box* testing tool, i.e. it does not require knowledge about the internals of the function under test. Therefore, besides on *programmed* numeric functions, we also apply our technique to machine-learned regression models. The experimental evaluation on a number of case studies shows the effectiveness of our testing approach.

Keywords: Property-based testing; Regression; Testing machine-learning models

1 Property-Driven Testing

Our testing approach is a form of *learning-based testing* (LBT) [Me], where – given a black-box system under test (SUT) – a *model* of the SUT is *learned*, and thereafter test cases are generated on the model. We in particular employ LBT for learning models of machine-learned functions. Based on this, we develop an approach which in addition to LBT allows for property specification and systematically generates test cases based on the property. Our approach consists of the following steps.

Property specification. We aim to generate test cases based on a requirement specification given by the user. In this, we follow the style used by property-based testing which takes the following form: $Assume \Rightarrow Assert$, where the $Assume$ specifies a pre-condition on the input, and $Assert$ specifies a post-condition on the output of the SUT. These logical conditions involve standard predicate logic using integers or real numbers and basic arithmetic and Boolean operators.

Test data generation. Once we have the property specification and the SUT, the next step is to use the property to generate test cases on the SUT. To this end, we first of all, *learn* a model using standard machine learning techniques. Since we consider numerical functions here, we learn regression models, to be precise either decision trees or neural networks. Next, the learned model as well as the negation of the property are translated into logical formulas. The conjunction of these two formulae is then given to an SMT solver. If the solver finds a (logically) satisfiable model to the formula as a counterexample to the property,

[1] Universität Paderborn, arnab.sharma@upb.de, [2] Universität Paderborn, melnikov@mail.upb.de, [3] LMU München, eyke@ifi.lmu.de, [4] Universität Oldenburg, heike.wehrheim@uol.de

Tab. 1: Results of detected violations (✓ = violation detected, ✗ = no violation detected)

SUT / *Property*	L-OWA	L-Uni	LAF	DeepSet	Total
	MLC/PT	MLC/PT	MLC/PT	MLC/PT	MLC/PT
infimum	✗/✗	✗/✗	✓/✓	✓/✓	2/2
supremum	✗/✗	✗/✗	✓/✓	✓/✓	2/2
monotonicity	✓/✓	✓/✗	✓/✓	✓/✓	**4/3**
Lipschitz	✓/✓	✓/✓	✓/✓	✓/✗	**4/3**
symmetry	✗/✗	✗/✗	✗/✗	✗/✗	0/0
idempotency	✗/✗	✓/✗	✓/✓	✓/✓	**3/2**
conjunction	✓/✓	✗/✗	✓/✓	✓/✓	3/3
disjunction	✓/✓	✓/✓	✓/✓	✓/✗	**4/3**
internality	✗/✗	✓/✗	✓/✓	✓/✓	**3/2**
invariance	✗/✗	✓/✗	✓/✓	✓/✓	**3/2**
additivity	✓/✓	✗/✗	✓/✗	✓/✗	**3/1**
total	5/5	**6/2**	**10/9**	**9/6**	**30/22**

we cross-check it on the SUT. This is necessary since we calculated the logical formula from the model approximating the SUT. If the counterexample is also valid for the SUT itself, we are done with testing and return the counterexample as a violation of the property. Otherwise, we use the counterexample to retrain the model.

We have implemented our approach as part of our existing tool MLcheck and have evaluated it on a number of *predefined* (i.e., implemented) and machine-learned numeric functions [Sh]. Table 1 shows some experimental results for 4 machine-learned SUTs testing for 11 properties (all common to aggregation functions). The table also gives a comparison of our approach (MLC) with property-based testing (PT)[5]. The blue-shaded cells are cases where the property holds for the SUT.

Data Availability

Our artifact can be found at https://github.com/arnabsharma91/MLCHECK-formalise.

Bibliography

[Me] Meinke, Karl: Learning-Based Testing: Recent Progress and Future Prospects. In: Machine Learning for Dynamic Software Analysis: Potentials and Limits - International Dagstuhl Seminar 16172, Germany, 2016.

[Sh] Sharma, Arnab; Melnikov, Vitalik; Hüllermeier, Eyke; Wehrheim, Heike: Property-Driven Testing of Black-Box Functions. In: 10th IEEE/ACM International Conference on Formal Methods in Software Engineering, FormaliSE@ICSE 2022, Pittsburgh, PA, USA, 2022.

[5] https://github.com/HypothesisWorks/hypothesis