2.3.3 Verification of the learned model

Practical guidance – cross-domain

Authors: Rob Ashmore (Dstl), and Dr Radu Calinescu and Dr Colin Paterson (Assuring Autonomy International Programme)

The Model Verification stage of the ML lifecycle is concerned with the provision of auditable evidence that a model will continue to satisfy its requirements when exposed to inputs which are not present in the training data.

Stage Input and Output Artefacts

The key input artefact to this stage is the trained model produced by the Model Learning stage. The key output artefacts are a verified model, and a verification result that provides sufficient information to allow potential users to determine if the model is suitable for the intended application(s).

Activities

1. **Requirement Encoding** - This activity involves transforming requirements into both tests and mathematical properties, where the latter can be verified using formal techniques. Requirements encoding requires a knowledge of the application domain, such that the intent which is implicit in the requirements may be encoded as explicit tests and properties. A knowledge of the technology which underpins the model is also required, such that technology-specific issues (such as overfitting or adversarial vulnerabilities) may be assessed through the creation of appropriate tests and properties.
2. **Test-Based Verification** - This activity involves providing test cases (i.e. specially-formed inputs or sequences of inputs) to the trained model and checking the outputs against predefined expected results. A large part of this activity involves an independent examination of properties considered during the Model Learning stage (guidance on model learning is provided in section 2.3.2), especially the Performant and Robust properties. In addition, this activity also considers test completeness, i.e. whether the set of tests exercised the model and covered its input domain sufficiently. The latter objective is directly related to the Complete property from the Data Management stage (guidance on data management is provided in section 2.3.1).
3. **Formal Verification** - This activity involves the use of mathematical techniques to provide irrefutable evidence that the model satisfies formally-specified properties derived from its requirements. Counterexamples are typically provided for properties that are violated, and can be used to inform further iterations of activities from the Data Management and Model Learning stages.

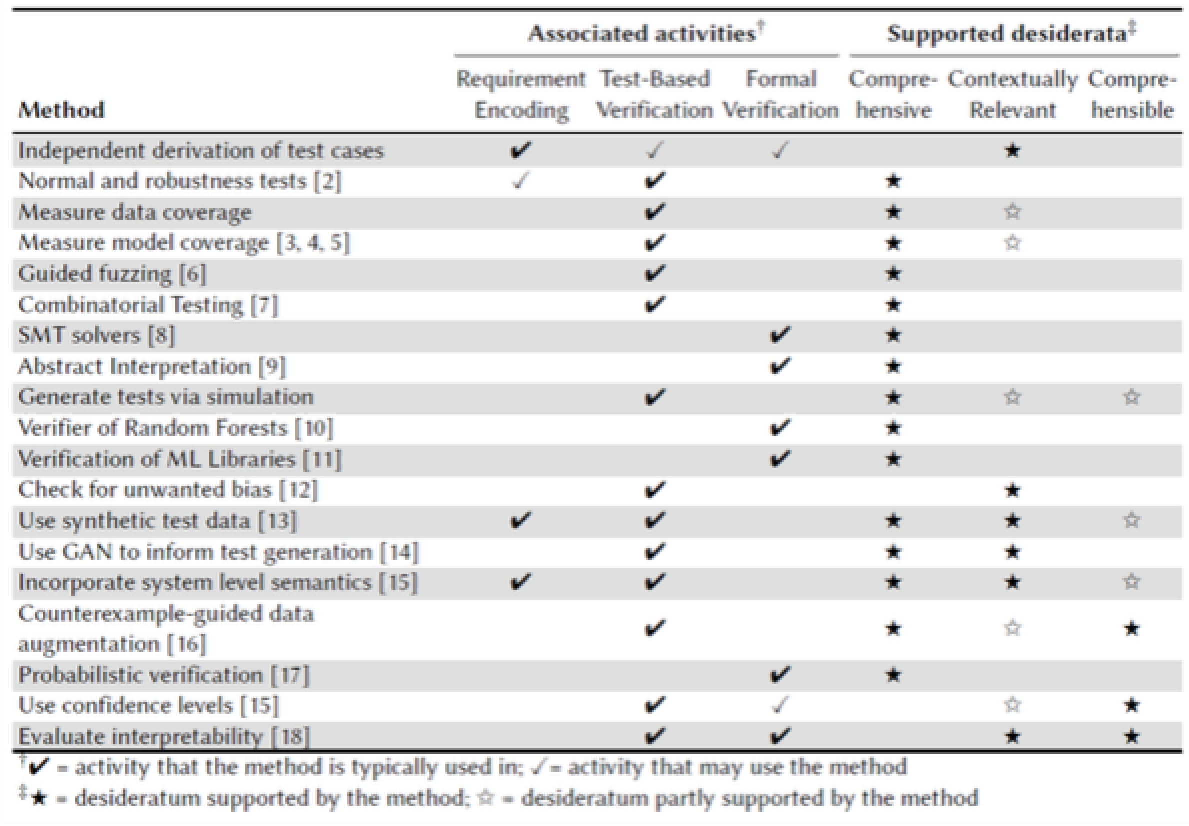
Desired Assurance Properties

In order to be compelling, the verification results (i.e. the evidence) generated by the Model Verification stage should exhibit the following key properties:

1. **Comprehensive** - This property is concerned with the ability of Model Verification to cover:
    a. all the requirements and operating conditions associated with the intended use of the model
    b. all the desired assurance properties from the previous stages of the ML lifecycle (e.g., the completeness of the training data, and the robustness of the model).
2. **Contextually relevant** - This property considers the extent to which test cases and formally verified properties can be mapped to contextually meaningful aspects of the system that will use the model. For example, for a model used in an autonomous car, robustness with respect to image contrast is less meaningful than robustness to variation in weather conditions.
3. **Comprehensible** - This property considers the extent to which verification results can be understood by those using them in activities ranging from data preparation and model development to system development and regulatory approval. A clear link should exist between the aim of the Model Verification and the guarantees it provides. Limitations and assumptions should be clearly identified, and results that show requirement violations should convey sufficient information to allow the underlying cause(s) for the violations to be fixed.

Methods

Table 1 provides a summary of the methods that can be applied during each Model Verification activity in order to achieve the desired assurance properties (desiderata). Further details on the methods listed in Table 1, along with the references to the documents cited in the table are available in [1].



Method	Associated activities†			Supported desiderata‡		
	Requirement Encoding	Test-Based Verification	Formal Verification	Compre-hensive	Contextually Relevant	Compre-hensible
Independent derivation of test cases	✔	✓	✓		★	
Normal and robustness tests [2]	✓	✔		★		
Measure data coverage		✔		★	☆	
Measure model coverage [3, 4, 5]		✔		★	☆	
Guided fuzzing [6]		✔		★		
Combinatorial Testing [7]		✔		★		
SMT solvers [8]			✔	★		
Abstract Interpretation [9]			✔	★		
Generate tests via simulation		✔		★	☆	☆
Verifier of Random Forests [10]			✔	★		
Verification of ML Libraries [11]			✔	★		
Check for unwanted bias [12]		✔			★	
Use synthetic test data [13]	✔	✔		★	★	☆
Use GAN to inform test generation [14]		✔		★	★	
Incorporate system level semantics [15]	✔	✔		★	★	☆
Counterexample-guided data augmentation [16]		✔		★	☆	★
Probabilistic verification [17]			✔	★		
Use confidence levels [15]		✔	✓		☆	★
Evaluate interpretability [18]		✔	✔		★	★

†✔ = activity that the method is typically used in; ✓ = activity that may use the method
‡★ = desideratum supported by the method; ☆ = desideratum partly supported by the method

Table 1 – Assurance methods for the Model Verification

Summary of Approach

1. Take the model to be verified produced by the Model Learning stage (guidance on model learning is provided in section 2.3.2).
2. Apply appropriate methods in order to undertake each model verification activity to demonstrate the machine-learnt model is suitable for its intended use.
    a. Apply appropriate methods for requirement encoding
    b. Apply appropriate methods for verification by adopting either a test-based or formal approach
3. Provide a verification result that provides sufficient information to allow potential users to determine if the model is suitable for its intended application.

References

- [1] Ashmore, R., Calinescu, R. and Paterson, C., 2019. Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges. arXiv preprint arXiv:1905.04223.
- [2] RTCA. 2011. Software Considerations in Airborne Systems and Equipment Certification. Technical Report DO-178C.
- [3] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, et al. 2018. DeepGauge: multi-granularity testing criteria for deep learning systems. In 33rd ACM/IEEE Int. Conf. on Automated Software Engineering. ACM, 120–131.
- [4] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In 26th Symp. on Operating Systems Principles. ACM, 1–18.

- [5] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. 2018. Concolic testing for deep neural networks. In 33rd ACM/IEEE Int. Conf. on Automated Software Engineering. ACM, 109–119.
- [6] Augustus Odena and Ian Goodfellow. 2018. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. (2018). arXiv:1807.10875
- [7] Lei Ma, Felix Juefei-Xu, Minhui Xue, et al. 2019. DeepCT: Tomographic combinatorial testing for deep learning systems. In 26th Int. Conf. on Software Analysis, Evolution and Reengineering. IEEE, 614–618.
- [8] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety verification of deep neural networks. In Int. Conf. on Computer Aided Verification. Springer, 3–29.
- [9] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, et al. 2018. AI2: Safety and robustness certification of neural networks with abstract interpretation. In IEEE Symp. on Security and Privacy (SP). IEEE, 3–18.
- [10] John Törnblom and Simin Nadjm-Tehrani. 2018. Formal verification of random forests in safety-critical applications. In Int. Workshop on Formal Techniques for Safety-Critical Systems. Springer, 55–71.
- [11] Daniel Selsam, Percy Liang, and David L Dill. 2017. Developing bug-free machine learning systems with formal mathematics. In 34th Int. Conf. on Machine Learning-Volume 70. JMLR. org, 3047–3056.
- [12] Rachel KE Bellamy, Kuntal Dey, Michael Hind, Samuel C Hoffman, et al. 2018. AI fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias. (2018). arXiv:1810.01943
- [13] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-networkdriven autonomous cars. In 40th Int. Conf. on software engineering. ACM, 303–314.
- [14] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. Deeproad: GAN-based metamorphic autonomous driving system testing. (2018). arXiv:1802.02295
- [15] Tommaso Dreossi, Somesh Jha, and Sanjit A Seshia. 2018. Semantic adversarial deep learning. (2018). arXiv:1804.07045
- [16] Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Kurt Keutzer, Alberto Sangiovanni-Vincentelli, and Sanjit A Seshia. 2018. Counterexample-guided data augmentation. (2018). arXiv:1805.06962
- [17] Perry Van Wesel and Alwyn E Goodloe. 2017. Challenges in the verification of reinforcement learning algorithms. (2017).
- [18] Finale Doshi-Velez and Been Kim. 2017. Towards a rigorous science of interpretable machine learning. (2017). arXiv:1702.08608