Thinking Beyond Distributions in Testing Machine Learned Models

Anonymous Author(s)
Affiliation
Address
email

Abstract

Testing practices within the machine learning (ML) community has centered around assessing a learned model's predictive performance measured against a test dataset, often drawn from the same distribution as the training dataset. While recent work on robustness and fairness testing within the ML community has pointed to the importance of testing against distributional shifts, these efforts also focus on estimating the likelihood of the model making an error against a reference dataset/distribution. We argue that this view of testing actively discourages researchers and developers from looking into other sources of robustness failures, for instance corner cases which may have severe undesirable impacts. We draw parallels with decades of work within software engineering testing focused on assessing a software system against various stress conditions, including corner cases, as opposed to solely focusing on average-case behaviour. Finally, we put forth a set of recommendations to broaden the view of machine learning testing to a rigorous practice.

1 Introduction

In Machine Learning (ML), testing often refers to the evaluation of a trained model on an unseen held-out test dataset, often expected to follow the same distribution as the training data. While such an evaluation shows the researcher (and the consumer of the research) how well the training algorithm captures phenomena of interest from the training data, and generalizes over to unseen data from the same distribution, it does not provide any guarantees around model behavior when they are deployed in practice [5]. In other words, such testing is meant to empirically evaluate certain sources of data, training algorithms, feature(s), or representational choices, rather than to estimate the utility and harms of the system when deployed on real world data [5, 31].

Prior work addresses this shortcoming in testing by evaluating the performance of models across domains [15, 32], introducing domain adaptation techniques [8], measuring performance under distribution shift [26], or collecting datasets that reflect target domain distributions or characteristics [7, 16, 14]. However, these efforts tie *testing an ML model* to testing against a dataset with a distribution that matches a target domain [3]. We argue that such distribution-driven testing implicitly makes assumptions about the costs of failures which often underestimate the importance and severity of failures in the tail of the distribution. However, it is often the "black swan" events in the tail of the distribution that lead to nonlinearities in behavior that result in unsafe outcomes (e.g., the meltdown security vulnerability in hardware [20, 17]). Even an ML system with 99% accuracy may have severe vulnerabilities that are masked within the 1% error rate. Furthermore, traditional ML testing does not account for the different severities of harms different errors cause in deployment, or whether they affect different subgroups at different rates [2].

In this paper, we provide a broad account of the various such methodological shortcomings in data distribution driven machine learning testing. We then draw on testing practices from other mature

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

engineering disciplines and provide a set of comprehensive recommendations towards rigorous ML testing, along the conditions to test for, the processes that guide this testing process, and the artefacts that can be a step towards ML system reliability.

2 Methodological Shortcomings in ML Model Testing

Historically, testing of AI systems was not exclusively statistical in nature [18]. For instance, [12] lays out how one might construct test sets for evaluating specific aspects of a Natural Language Processing (NLP) system by using both a randomly sampled dataset to evaluate the conceptual competence of the system, and a dataset of instances containing different linguistic variations of the same sentence conveying the same meaning, as a way to evaluate the linguistic competence of the NLP system. The switch from such a broad view of what evaluation means, to a singular focus on statistical evaluations based on held-out test sets is arguably a side effect of the overwhelming success of statistical NLP in the 90s. From a system reliability perspective, the goal of model testing should ideally include considerations such as a) robustness to distribution shifts, b) distributions of errors, and c) severities of those errors. These factors are often overlooked, or relegated to optional, auxiliary testing efforts in much of ML research and development. In this section, we outline some of the core shortcomings that results from the common practices within machine learning testing.

Treating all examples as equal: The dominant paradigm of measuring model accuracy on an I.I.D. dataset drawn from the same distribution as the training data mitigates against the pitfalls of over-fitting during learning. However it also makes two critical assumptions which reduce the diagnostic power of the testing. Firstly, in weighting all data points in the test set equally, the measurement is most sensitive to failures in model performance on the head of the distribution—and conversely least sensitive to failures in model performance on the tail. For example, if an image understanding model is evaluated on a dataset in which humans appear in only a small fraction of images then those images will be given little weight in the evaluation. Secondly, the comparison of predictions (i.e., model outputs) with expectations ignores any available context. Notably, it ignores features of the model inputs. For example, the evaluation of an image analysis model for detecting stop signs would ignore the important factor of whether there are also pedestrians or cyclists in the image.

Treating all failures as equal: Suppose a computer vision system misidentifies a person as an animal. This is a significantly different type of error than, say, misidentifying a cat for a dog. The dominant ML testing paradigm fails to account for the fact that not all errors are qualitatively equal. This simplification reduces the labour and other efficiency costs involved in doing contextualized evaluations of classes of errors. We similarly see these economic imperatives at play in the focus of much of the work on (un)fairness in NLP on gender bias specifically—compared to other axes of biases—because gender-labeled datasets are relatively easier to assemble [2]. Similarly, most of the testing efforts outlined above treat models in isolation, considering just the inputs and outputs of the model, whereas, it is important to consider how the model integrates into the environment where it will be introduced, accounting for delayed impacts and feedback loops [21].

Overlooking corner cases: As outlined above, the traditional ML testing often focuses on average-case behaviour of the model. Each data point in a test set can be viewed as an individual test, and whether or not the data point, or test, passes or fails makes up part of the accuracy, which is then reported as a metric for system performance. However, this approach often does not look into which data points are failing the test, or whether there is a pattern there. Often times the errors correspond to the edge cases that are a minority, while the performance metric focuses on the head of the distribution. Recent work pointing to fairness and robustness failures has prompted work on teasing apart the evaluation set, and associating a subset of that set with a behavior one wants to test for. For instance, disaggregated fairness evaluation is essentially about performing the test on subsets of the data that have a certain feature held constant. Similarly, robustness testing often involves examining the tail distribution for certain edge cases–things like natural adversarial examples [15], or typos for NLP [28]. In other words, defining metrics for fairness and robustness can in essence be thought of as adding specific attention to sections of the tail distribution.

Lack of an integrated process: It is important to note that recent research has begun inquiries into ML testing for the edge cases. For instance, fairness testing or disaggregated testing [2] separately assesses model performance against data pertaining to different subgroups of people; counterfactual testing [19] such as perturbation sensitivity analysis [25] assesses model behaviour in response to

small controlled changes in the input, similar to unit testing. Approaches such as the fairness gym [6] or system dynamics based simulations [21] can be thought of as instances of integration testing. However, these efforts largely continue to be disconnected efforts with those specific robustness goals, rather than being integrated into a unified or standard process of ML development, even for critical production systems.

Lack of artifacts for transparency: One of the consequences of not having integrated processes around comprehensive testing in ML is that there is no standard mechanism to communicate which tests have been performed on the ML model, and which ones of them succeeded vs. failed. While frameworks such as model cards [23] provide a great transparency mechanism, most implemented model cards in practice reports largely on traditional held-out testing results, and sometimes disaggregated fairness evaluation. Lack of such standardized mechanisms to communicate the test results of a comprehensive suite of test cases puts the end users and stakeholders at a disadvantage.

3 Towards More Comprehensive Testing in ML

Establishing guarantees of an ML model's behaviour in real-world contexts should become a core criteria for its adoption in real-world application scenarios. In this section, we put forth a set of recommendations towards rigorous testing practices that can provide strong, reliable, and transparent guarantees for ML system reliability. Our recommendations pertain to practices along three layers: what *conditions* are tested for, what *processes* guide the testing, and what *artifacts* communicate the test results to the model consumers. We draw inspiration from mature testing practices in different engineering disciplines, especially software engineering, as well as testing-oriented engineering paradigms such as test-driven development. Our recommendations are aimed at applications where guarantees on real-world performance and reliability are crucial; however, these recommendations are also relevant in research scenarios in high-stakes domains, such as health care.

3.1 *Conditions* that are tested for

As outlined in Section 2, current ML testing predominantly focuses on measuring average case behaviour of models, in isolated test environments, to statistically establish superiority of data and/or algorithms used to build the models. This averaging is done in at least three ways, each of which reduces the informativeness of the test results: i) all data points are weighted equally, ii) all contexts are treated as equal, and iii) all error types are weighted equally. We argue that in real-world applications, one must shift the focus to an engineering-motivated goal of ensuring that the model will work as desired in the contexts it will be deployed in. While testing against a target distribution is an effort in this direction, it does not address all the shortcomings. In software testing, for instance, unit testing is a step used to isolate and examine a small piece of code against potential failure modes (or test cases) that are designed by the developer/tester who knows the expected behaviour of that module. What could unit testing look like in the case of ML testing? Recent research on designing test sets that focus on failure cases close to natural data distribution [15], introducing adversarial examples for robustness [11], and behavioral testing of NLP models [28] can all be considered as instances of unit testing. Similarly, it is important to test for emergent failure modes when different component ML models are integrated as a part of a larger system, since the compounding of errors may follow patterns undetectable during isolated testing [21]. In addition, failure cases are not all similar and from the same priority. For example misclassifying a can on the street can be an edge case of a self-driving car, but its severity is not as much as misclassifying a pedestrian and traffic sign in the same scenario. These two examples are not of the same severity and shouldn't be treated as such.

Recommendations:

- Consider going beyond the average case behavior of the system, and design test cases that cover corner cases of interest specific to the task.
- Consider the context in which the system will be deployed and assign the severity of failing the tests, in order to prioritize testing and subsequent fixes.
- Consider test cases that account for the societal disparities across different subgroups and how failures might impact different subgroups in different rates.
- Make sure that test cases are also designed for scenarios where the ML model is integrated with other ML and non-ML components.

3.2 *Processes* that guide the testing

In the traditional software development process, testing takes up significant amount of time, resources, and effort [13]. Even moderate-sized software projects takes up hundreds of person-hours dedicated to writing test cases — conditions to test for and pass/fail criteria, implementing them, and meticulously documenting the results of those tests. In fact, software testing is considered an art [24] requiring its own technical and non-technical skills [30, 22], and entire career paths are built around testing [4]. *Test-driven development*, often associated with agile software engineering frameworks, is a practice which integrates testing considerations in all parts of the development process [1, 10]. These processes rely on a deep understanding of the software requirements as well as user behavior modeling, in order to anticipate failure modes during deployment, as well as continued expansion of the test suite. In contrast, ML testing is often relegated to a small portion of the ML development process, and predominantly focuses on a static snapshot of data to provide performance guarantees. Despite the growing research into fairness and robustness failures in ML models, these efforts are often relegated to auxiliary steps that are not fully integrated into a typical ML development process. Identifying failure modes in a deployed ML system is not always straightforward; it require a deep understanding of the societal ecosystems surrounding ML interventions [29].

Recommendations:

- Consider thinking, planning and integrating testing in all stages of development process, research problem ideation, setting objectives, and implementation.
- Consider building a practice around documenting desirable behaviours of the ML system as test cases, and how to bring diverse perspectives in designing this test suite.
- Consider participatory approaches (e.g., [21]) to to ensure that the test suite accounts for the societal contexts and the embedded values within which the ML system will be deployed.

3.3 *Artifacts* that document the test results

Recent work has pointed to the importance of standard frameworks for transparency [9, 23] and accountability [27] in ML based interventions. While transparency artifacts such as datasheets [9] and model cards [23] are frameworks derived from practices in other mature domains such as hardware documentation practices or food nutrition labeling, these frameworks currently do not provide a way to communicate comprehensive test results. As [13] points out, software testing produces a number of artifacts including execution traces, test results, as well as test coverage information. In addition to increased transparency, such information serve also as a way to iteratively build new test cases for newer versions of the software. ML transparency mechanisms should ideally be expanded to include such comprehensive test artifacts.

Recommendations:

- Consider expanding ML transparency mechanisms such as model cards [23] to include a more comprehensive set of test results.
- Consider documenting the processes that went into building the set of test cases, so that the consumer of the ML system has a better understanding of its reliability.
- Consider documenting internal/external auditing practices (e.g., [27] for both for model outputs and processes.

4 Conclusion

In this paper, we presented an overview of methodological shortcomings in traditional ML model testing. While we recognize the importance of testing against distribution shift, we argue in favor of going beyond the sole focus on distribution driven testing, and take into consideration edge cases and severity of failures while assessing an ML model's performance. We draw inspiration from decades of work within software engineering testing practices, focused on assessing a software system against various stress conditions, measuring severity of failure cases and assessing test priorities. Based on this, we propose a comprehensive set of recommendations around the conditions to test for, the process that guide the testing, and the artifacts produced as the result of the testing process.

References

[1] Dave Astels. *Test driven development: A practical guide*. Prentice Hall Professional Technical Reference, 2003.

[2] Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on fairness, accountability and transparency*, pages 77–91. PMLR, 2018.

[3] Kyla Chasalow and Karen Levy. Representativeness in statistics, politics, and machine learning. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 77–89, 2021.

[4] Sean Cunningham, Jemil Gambo, Aidan Lawless, Declan Moore, Murat Yilmaz, Paul M Clarke, and Rory V O'Connor. Software testing: a changing career. In *European Conference on Software Process Improvement*, pages 731–742. Springer, 2019.

[5] Alexander D'Amour, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen, Jonathan Deaton, Jacob Eisenstein, Matthew D Hoffman, et al. Underspecification presents challenges for credibility in modern machine learning. *arXiv preprint arXiv:2011.03395*, 2020.

[6] Alexander D'Amour, Hansa Srinivasan, James Atwood, Pallavi Baljekar, D Sculley, and Yoni Halpern. Fairness is not static: deeper understanding of long term fairness via simulation studies. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 525–534, 2020.

[7] Krishnamurthy Dj Dvijotham, Jamie Hayes, Borja Balle, Zico Kolter, Chongli Qin, Andras Gyorgy, Kai Xiao, Sven Gowal, and Pushmeet Kohli. A framework for robustness certification of smoothed classifiers using f-divergences. 2020.

[8] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *International conference on machine learning*, pages 1180–1189. PMLR, 2015.

[9] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. Datasheets for datasets. *arXiv preprint arXiv:1803.09010*, 2018.

[10] Boby George and Laurie Williams. A structured experiment of test-driven development. *Information and software Technology*, 46(5):337–342, 2004.

[11] Karan Goel, Nazneen Fatema Rajani, Jesse Vig, Zachary Taschdjian, Mohit Bansal, and Christopher Ré. Robustness gym: Unifying the NLP evaluation landscape. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Demonstrations*, pages 42–55, Online, June 2021. Association for Computational Linguistics.

[12] Giovanni Guida and Giancarlo Mauri. Evaluation of natural language processing systems: Issues and approaches. *Proceedings of the IEEE*, 74(7):1026–1035, 1986.

[13] Mary Jean Harrold. Testing: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 61–72, 2000.

[14] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

[15] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15262–15271, 2021.

[16] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In *International conference on machine learning*, pages 1989–1998. PMLR, 2018.

[17] Zhichao Hua, Dong Du, Yubin Xia, Haibo Chen, and Binyu Zang. {EPTI}: Efficient defence against meltdown attack for unpatched vms. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 255–266, 2018.

[18] Karen Sparck Jones and Julia R Galliers. *Evaluating natural language processing systems: An analysis and review*, volume 1083. Springer Science & Business Media, 1995.

[19] Matt J Kusner, Joshua R Loftus, Chris Russell, and Ricardo Silva. Counterfactual fairness. *arXiv preprint arXiv:1703.06856*, 2017.

[20] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown. *arXiv preprint arXiv:1801.01207*, 2018.

[21] Donald Martin, Jr., Vinodkumar Prabhakaran, Jill Kuhlberg, Andrew Smart, and William S. Isaac. Extending the machine learning abstraction boundary: A complex systems approach to incorporate societal context, 2020.

[22] Gerardo Matturro. Soft skills in software engineering: A study of its demand by software companies in uruguay. In *2013 6th international workshop on cooperative and human aspects of software engineering (CHASE)*, pages 133–136. IEEE, 2013.

[23] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 220–229, 2019.

[24] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[25] Vinodkumar Prabhakaran, Ben Hutchinson, and Margaret Mitchell. Perturbation sensitivity analysis to detect unintended model biases. *arXiv preprint arXiv:1910.04210*, 2019.

[26] Stephan Rabanser, Stephan Günnemann, and Zachary C Lipton. Failing loudly: An empirical study of methods for detecting dataset shift. *arXiv preprint arXiv:1810.11953*, 2018.

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

[28] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond Accuracy: Behavioral Testing of NLP models with CheckList. In *Association for Computational Linguistics (ACL)*, pages 4902–4912, 2020.

[29] Nithya Sambasivan, Erin Arnesen, Ben Hutchinson, Tulsee Doshi, and Vinodkumar Prabhakaran. Re-imagining algorithmic fairness in india and beyond. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 315–328, New York, NY, USA, 2021. Association for Computing Machinery.

[30] Mary Sánchez-Gordón, Laxmi Rijal, and Ricardo Colomo-Palacios. Beyond technical skills in software testing: Automated versus manual testing. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, pages 161–164, 2020.

[31] Sahil Singla, Besmira Nushi, Shital Shah, Ece Kamar, and Eric Horvitz. Understanding failures of deep networks via robust feature extraction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12853–12862, 2021.

[32] Michał Zajac, Konrad Zołna, Negar Rostamzadeh, and Pedro O Pinheiro. Adversarial framing for image and video classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 10077–10078, 2019.