Static Analysis of Data Science Software

Caterina Urban

To cite this version:

Caterina Urban. Static Analysis of Data Science Software. SAS 2019 - 26th Static Analysis Symposium, Oct 2019, Porto, Portugal. pp.17-23, 10.1007/978-3-030-32304-2_2 . hal-02397699

HAL Id: hal-02397699

https://hal.inria.fr/hal-02397699

Submitted on 6 Dec 2019

Static Analysis of Data Science Software

Caterina Urban[1,2]

[1] INRIA, Paris, France
[2] DIENS, École Normale Supérieure, CNRS, PSL University, Paris, France
caterina.urban@inria.fr

Abstract. Data science software is playing an increasingly important role in every aspect of our daily lives and is even slowly creeping into mission critical scenarios, despite being often opaque and unpredictable. In this paper, we will discuss some key challenges and a number of research questions that we are currently addressing in developing static analysis methods and tools for data science software.

1 Introduction

Nowadays, thanks to advances in machine learning and the availability of vast amounts of data, computer software plays an increasingly important role in assisting or even autonomously performing tasks in our daily lives.

As data science software becomes more and more widespread, we become increasingly vulnerable to *programming errors*. In particular, programming errors that do not cause failures can have serious consequences since code that produces an erroneous but plausible result gives no indication that something went wrong. This issue becomes particularly worrying knowing that machine learning software, thanks to its ability to efficiently approximate or simulate more complex systems [22], is slowly creeping into mission critical scenarios[3].

However, programming errors are not the only concern. Another important issue is the vulnerability of machine learning models to *adversarial examples* [39], that is, small input perturbations that cause the model to misbehave in unpredictable ways. More generally, a critical issue is the notorious difficulty to interpret and explain machine learning software[4].

Finally, as we are witnessing widespread adoption of software with far-reaching societal impact — i.e., to automate decision-making in fields such as social welfare, criminal justice, and even health care — a number of recent cases have evidenced the importance of ensuring software *fairness* and non-discrimination[5] as well as data *privacy*[6]. Going forward, data science software will be subject to

[3] https://www.airbus.com/innovation/future-technology/artificial-intelligence.html
[4] http://www.technologyreview.com/s/604087/the-dark-secret-at-the-heart-of-ai
[5] https://www.nytimes.com/2017/10/26/opinion/algorithm-compas-sentencing-bias.html
[6] https://www.nytimes.com/2012/02/19/magazine/shopping-habits.html

more and more legal regulations (e.g., the European General Data Protection Regulation adopted in 2016) as well as administrative audits.

It is thus paramount to develop method and tools that can keep up with these developments and enhance our understanding of data science software and ensure it behaves correctly and reliably. In this paper, we will discuss challenges and a number of research questions that we are currently addressing in this area.

2 Key Challenges

A number of key challenges differentiate static analysis methods for data science software from static analysis of regular software. We discuss them below.

Dirty Data. Data is often incorrect, inaccurate, incomplete, or inconsistent and needs to be cleaned before it can be used. According to recent surveys, data preparation occupies between 50% and 80% of the time of a data scientist[7]. Moreover, data preparation code is the most fragile in a data science pipeline as it generally heavily relies on implicit assumptions on the data. For this latter reason, static analysis methods for data preparation code involve an *additional level of indirection* compared to more classical static analyses that infer properties about program variables.

Inscrutability. The behavior of machine learning models is poorly understood [15]. Some mathematical properties of these models have been discovered [26] but the mathematical theory generally still lacks behind. Therefore, static analysis methods for machine learning have *no semantics to build upon* as in traditional application scenarios, e.g. [11].

Meaningless Accuracy. The performance of machine learning models is measured by their accuracy on the testing data. However, this measure does not provide any general guarantee on the model behavior on other, previously unseen, data. Thus, static analysis methods for machine learning must be data-independent, lest they remain limited to local properties, e.g., [14].

Lack of Specifications. It is often hard to formally specify the correct behavior of a machine learning model, e.g., it is not obvious how to specify an obstacle that a machine learning model should recognize[8]. Generally, some specification is reconstructed at the system level, by combining together information coming from multiple system components, e.g., from a machine learning model and multiple sensors. Similarly, without a formal specification to refer to, static analysis methods also *need to be decomposed*, each component dedicated to a well-identified property that can be formalized.

Scalability and Precision. Static analysis methods for machine learning models only need to handle relatively simple operations such as matrix multiplications and activation functions. However, scaling to certain model architectures used in practice while retaining enough precision to prove useful

[7] https://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html

[8] https://www.tesla.com/blog/tragic-loss

properties, remains a challenge. To this end, new static analysis methods should be designed that employ dedicated and clever partitioning strategies [34] or specialized new (combinations of) abstract domains [37].

3 Research Questions

To address the above challenges we are currently working in the four main directions that we present below.

Implicit Assumptions on Data. Data preparation code is generally disregarded as glue code and, for this reason, is usually poorly tested. This, together with the fact that this code is often written in a rush and is highly dependent on the data (e.g., the use of magic constants is not uncommon for this kind of code), greatly increases the likelihood for programming errors to remain unnoticed.

To address these issues, we have developed a static analysis that automatically the infers implicit assumptions on the data that are embedded in the code. Specifically, we infer assumptions on the *structure* of the data as well as on the data *values* and the *relations* between the data. The analysis uses a combination of existing abstract domains [7,8,29, etc.] extended to indirectly reason about the data rather than simply reasoning about program variables.

The inferred assumptions can simply provide feedback on one's expectations on both the program and the data. Alternatively, they can be leveraged for testing the data, e.g., via grammar-based testing [18]. Finally and more interestingly, they can be used to automatically check and guide the cleaning of the data.

Data Usage. A common source of errors in data science software is data being mistakenly ignored. A notable example is economists Reinhart and Rogoff's paper "Growth in a Time of Debt", which was widely cited in political debates and was later demonstrated to be flawed. Indeed, one of the flaws was a programming error, which entirely excluded some data from the analysis [19]. Its critics hold that this paper led to unjustified adoption of austerity policies in the European Union [27]. The likelihood of data remaining accidentally unused becomes particularly high for long data science pipelines.

In recent work [41], we have proposed a static analysis framework for automatically detecting unused data. They key ingredient of the framework is the notion of *dependency* between the data and the outcome of the program. This yields a *unifying framework* that encompasses dependency-based analyses that arise in many other contexts, such as secure information flow [38], program slicing [42], as well as provenance and lineage analysis [5], to name a few.

Algorithmic Bias. It is not difficult to envision that in the future most of the decisions in society will be delegated to software. It is thus becoming increasingly important to be able to detect whether the software is operating fairly or it is reinforcing biases and perpetuating prejudices.

To this end, we have designed a general static analysis framework for proving *causal fairness* [24]. Within this framework we have developed a scalable static analysis for feed-forward multi-layer neural networks. The analysis is a combination of a forward and backward analysis; the forward analysis effectively splits the analysis into independent parallelizable tasks and overall reduces the analysis effort. One can tune the precision and cost of the analysis by adjusting the size of the tasks and the total time allotted to the analysis. In this way, one can adapt the analysis to the context in which it is being deployed and even make it incremental, i.e., by resuming the tasks on which the analysis is imprecise once more resources become available.

Global Robustness. Finally, we are working on generalizing the framework discussed above to proving *global robustness* of machine learning models. Note that this property is concerned with certifying the whole input space and is thus much harder than local robustness [14,32]. Specifically, we are designing a framework parametric in the chosen notion of (abstract) distance between input data points. In order to scale to larger neural networks with more complex architectures, we are studying new combinations of existing abstract domains [28] as well as new specialized abstract domains.

4 Related Work

We now quickly survey some of the related work in the area, broadly defined. The discussion is by no means exhaustive but only intended to suggest some useful reference pointers for further exploration.

Spreadsheet Analyses. There has been considerable work on testing, analyzing, and debugging spreadsheets [2,6,35, etc.]. These mostly target errors (e.g., type errors) in the data rather than in the software (i.e., the spreadsheet formulas).

Adversarial Examples. Since neural networks were shown to be vulnerable to them [39], a lot of work has been focused on constructing adversarial examples [31,40, etc.] and harnessing them for adversarial training [16,20,30, etc.].

Robustness Analyses. Comparatively little work instead has been dedicated to testing [32] and verifying [21,23,33, etc.] neural network robustness. The challenge remains to scale to large and complex network architectures used in practice, with a few recent notable exceptions, e.g., [14,37]. On the other hand, robustness analyses for neural networks deal with much simpler control structures compared to regular programs [4,17,25, etc.].

Fairness and Privacy. Work on testing and verifying other properties such as fairness and privacy[1,10,13, etc.] is generally limited to standard machine learning software or rather small neural networks, with recent exceptions, e.g., [3].

Probabilistic Programs. Finally, data science programs can also be seen as probabilistic programs, for which a vast literature exists [9,12,36, etc.]. We refer to [36] for an in-depth discussion of the related work in this area.

5 Conclusion

We discussed the challenges and some of our progress in developing static analyses for data science software. Much more work remains to be done and, as our automated future presses for results, we hope that this exposition encourages the formal methods community as a whole to contribute to this effort.

References

1. A. Albarghouthi, L. D'Antoni, S. Drews, and A. V. Nori. FairSquare: Probabilistic Verification of Program Fairness. *PACMPL*, 1(OOPSLA):80:1–80:30, 2017.
2. D. W. Barowy, D. Gochev, and E. D. Berger. CheckCell: Data Debugging for Spreadsheets. In *OOPSLA*, pages 507–523, 2014.
3. O. Bastani, X. Zhang, and A. Solar-Lezama. Verifying Fairness Properties via Concentration. *CoRR*, abs/1812.02573, 2018.
4. S. Chaudhuri, S. Gulwani, and R. Lublinerman. Continuity and Robustness of Programs. *Communications of the ACM*, 55(8):107–115, 2012.
5. J. Cheney, A. Ahmed, and U. A. Acar. Provenance as Dependency Analysis. *Mathematical Structures in Computer Science*, 21(6):1301–1337, 2011.
6. T. Cheng and X. Rival. An Abstract Domain to Infer Types over Zones in Spreadsheets. In *SAS*, pages 94–110, 2012.
7. G. Costantini, P. Ferrara, and A. Cortesi. A Suite of Abstract Domains for Static Analysis of String Values. *Software - Practice and Experience*, 45(2):245–287, 2015.
8. P. Cousot and R. Cousot. Static Determination of Dynamic Properties of Programs. In *Second International Symposium on Programming*, pages 106–130, 1976.
9. P. Cousot and M. Monerau. Probabilistic Abstract Interpretation. In *ESOP*, pages 169–193, 2012.
10. A. Datta, M. Fredrikson, G. Ko, P. Mardziel, and S. Sen. Use Privacy in Data-Driven Systems: Theory and Experiments with Machine Learnt Programs. In *CCS*, pages 1193–1210, 2017.
11. J. Feret. Static Analysis of Digital Filters. In *ESOP*, pages 33–48, 2004.
12. A. Filieri, C. S. Pasareanu, and W. Visser. Reliability Analysis in Symbolic Pathfinder. In *ICSE*, pages 622–631, 2013.
13. S. Galhotra, Y. Brun, and A. Meliou. Fairness Testing: Testing Software for Discrimination. In *FSE*, pages 498–510, 2017.
14. T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. T. Vechev. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *S & P*, pages 3–18, 2018.
15. L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal. Explaining Explanations: An Approach to Evaluating Interpretability of Machine Learning. *CoRR*, abs/1806.00069, 2018.
16. I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. In *ICLR*, 2015.

17. E. Goubault and S. Putot. Robustness Analysis of Finite Precision Implementations. In *APLAS*, pages 50–57, 2013.
18. M. Hennessy and J. F. Power. An Analysis of Rule Coverage as a Criterion in Generating Minimal Test Suites for Grammar-Based Software. In *ASE*, pages 104–113, 2005.
19. T. Herndon, M. Ash, and R. Pollin. Does High Public Debt Consistently Stifle Economic Growth? A Critique of Reinhart and Rogoff. *Cambridge Journal of Economics*, 38(2):257–279, 2014.
20. R. Huang, B. Xu, D. Schuurmans, and C. Szepesvári. Learning with a Strong Adversary. *CoRR*, abs/1511.03034, 2015.
21. X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety Verification of Deep Neural Networks. In *CAV*, pages 3–29, 2017.
22. K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer. Policy Compression for Aircraft Collision Avoidance Systems. In *DASC*, pages 1–10, 2016.
23. G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *CAV*, pages 97–117, 2017.
24. M. Kusner, J. Loftus, C. Russell, and R. Silva. Counterfactual Fairness. In *NIPS*, pages 4069–4079, 2017.
25. R. Majumdar and I. Saha. Symbolic Robustness Analysis. In *RTSS*, pages 355–363, 2009.
26. S. Mallat. Understanding Deep Convolutional Networks. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374, 2016.
27. J. Mencinger, A. Aristovnik, and M. Verbič. The Impact of Growing Public Debt on Economic Growth in the European Union. *Amfiteatru Economic*, 16(35):403–414, 2014.
28. A. Miné. Symbolic Methods to Enhance the Precision of Numerical Abstract Domains. In *VMCAI*, pages 348–363, 2006.
29. A. Miné. The Octagon Abstract Domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.
30. M. Mirman, T. Gehr, and M. T. Vechev. Differentiable Abstract Interpretation for Provably Robust Neural Networks. In *ICML*, pages 3575–3583, 2018.
31. A. M. Nguyen, J. Yosinski, and J. Clune. Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images. In *CVPR*, pages 427–436, 2015.
32. K. Pei, Y. Cao, J. Yang, and S. Jana. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *SOSP*, pages 1–18, 2017.
33. L. Pulina and A. Tacchella. An Abstraction-Refinement Approach to Verification of Artificial Neural Networks. In *CAV*, pages 243–257, 2010.
34. X. Rival and L. Mauborgne. The Trace Partitioning Abstract Domain. *Transactions on Programming Languages and Systems*, 29(5):26, 2007.
35. G. Rothermel, M. M. Burnett, L. Li, C. DuPuis, and A. Sheretov. A Methodology for Testing Spreadsheets. *Transactions on Software Engineering and Methodology*, 10(1):110–147, 2001.
36. S. Sankaranarayanan, A. Chakarov, and S. Gulwani. Static Analysis for Probabilistic Programs: Inferring Whole Program Properties from Finitely Many Paths. In *PLDI*, pages 447–458, 2013.
37. G. Singh, T. Gehr, M. Püschel, and M. T. Vechev. An Abstract Domain for Certifying Neural Networks. *PACMPL*, 3(POPL):41:1–41:30, 2019.

38. G. Smith. Principles of Secure Information Flow Analysis. In *Malware Detection*, volume 27 of *Advances in Information Security*, pages 291–307. Springer, 2007.
39. C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing Properties of Neural Networks. In *ICLR*, 2014.
40. P. Tabacof and E. Valle. Exploring the Space of Adversarial Images. In *IJCNN*, pages 426–433, 2016.
41. C. Urban and P. Müller. An Abstract Interpretation Framework for Input Data Usage. In *ESOP*, pages 683–710, 2018.
42. M. Weiser. Program Slicing. *Transactions on Software Engineering*, 10(4):352–357, 1984.