Quantitative Input Feature Usage

Denis Mazzucato, Marco Campion, and Caterina Urban, Inria & École Normale Supérieure | Université PSL
{denis.mazzucato,marco.campion,caterina.urban}@inria.fr

Nowadays, data science plays a crucial role by helping organizations and individuals make critical decisions based on data-driven insights in a variety of fields, including healthcare, finance, and avionics. Therefore, disastrous outcomes may result from programming errors in these safety-critical settings. As our reliance on data science grows, so does our vulnerability to errors, especially when they do not result in software failures but instead produce a plausible yet erroneous outcome. Such bugs are hard to spot since they provide no indication that something went wrong. A potential source of such errors is when an input feature of an application has an unexpected impact on the program computation compared to the developers' expectations. The likelihood that a programming error causes some input feature to be more, or less, influent than what the application expects is even higher for data science applications, where data goes through a long pipeline of layers that filter, merge, and manipulate it. A notable example is the Reinhard and Rogoff article "Growth in a Time of Debt" [3], which justified austerity measures around the world in the following years, and was later discovered to be flawed. Notably, one of the several programming errors discovered in the article is the unusual usage of the input value relative to Norway's economic growth in 1964, with an excessive impact on the article conclusions. This structural flaw in computations indicates a key methodological problem that compromised the authors' claims. Therefore, it is crucial to achieve a high level of confidence in data science software.

Previous efforts have mostly targeted qualitative properties about input feature usage, addressing whether an input feature is either used or not [5], yet providing no information on whether the input feature is slightly or heavily used. For instance, such analyses would not have caught the above mentioned issue in Reinhard and Rogoff's paper. To address this need for less stringent requirements, we present a novel quantitative input feature usage framework to discriminate between input features with different impact on the program's outcome. This framework can be used to identify input features that have a disproportionate impact on the computation. Such knowledge could either certify intended behavior or reveal potential flaws, by matching the developers' intuition on the expected impact of their input features with the actual result of the quantitative study. Our quantitative approach subsumes other similar qualitative properties, such as input data usage [5] or fairness certification [6]. We characterize the impact of an input feature with a notion of dependency between input features and the outcome of the program, taking into account non-determinism and non-termination. Compared to other quantitative notions of dependency, e.g., quantitative information flow [4], the key difference is about the information we measure. Our quantitative definition quantifies the effect of the input features on the program outcome while quantitative information flow counts how many bits are used to compute the result.

More specifically, we introduce a backward static analysis within the formal framework of abstract interpretation [2] to automatically compute an upper bound of the impact of input features. We derive our static analysis by a hierarchy of consecutive abstractions to exactly capture the information needed to quantify input feature usage, forgetting the unnecessary details. This clear design principle ensures that our abstract semantics soundly approximate our semantic property. Meaning that the upper bound of the impact produced by the abstraction always over-approximates the real one. The abstract domain to enable computation of the impact is based on the (backward) disjunctive polyhedra domain and an abstraction representing a cumulative range of different program outputs, called the bucket abstraction. We offer a few variations of this abstract domain depending on the definition of impact. For example, one abstraction computes the volume of intersections between regions of input values leading to different output buckets: the bigger the intersection volume the greater the impact.

We provide a preliminary implementation that shows the practical applicability of our approach, computing an upper bound of the input features impact for feed-forward ReLU-activated neural networks. As future work, we plan to compare our approach with feature importance metrics [1] and show that our sound technique can produce useful input usage information. We also plan to conduct a fairness study by identifying input features susceptible to bias. Furthermore, we are going to consider general programs with non-trivial control flow and illustrate how our tool overcomes program analysis challenges in presence of loops and conditional statements.

Bibliography:

[1] L. Breiman. Random Forests. 2001
[2] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. 1977.
[3] C. M. Reinhart, and K. S. Rogoff. Growth in a Time of Debt. 2010.
[4] G. Smith. On the Foundations of Quantitative Information Flow. 2009.
[5] C. Urban, P. Müller. An Abstract Interpretation Framework for Input Data Usage. 2018.
[6] C. Urban, M. Christakis, V. Wüstholz, and F. Zhang. Perfectly parallel fairness certification of neural networks. 2020.