# Technical Commentary

Martin C. Rinard
MIT EECS and CSAIL
{rinard}@csail.mit.edu

Big data combined with machine learning has revolutionized fields such as computer vision, robotics, and natural language processing. In these fields, automated techniques that detect and exploit complex patterns hidden within large data sets have repeatedly outperformed techniques based on human insight and intuition.

But despite the availability of enormous amounts of code (big code) that could, in theory, be leveraged to deliver similar advances for software, programming has proved to be remarkably resistant to this kind of automation. Much programming today consists of developers deploying keyword searches against online information aggregators such as Stack Overflow to find, then manually adapt, code sequences that implement desired behaviors.

The current paper presents new techniques for leveraging big code to automate two programming activities: 1) selecting understandable names for JavaScript identifiers and 2) generating type annotations for JavaScript variables. The basic approach leverages large JavaScript code bases to build a probabilistic model that predicts names and type annotations given the surrounding context (which includes constants, JavaScript API calls, and variable uses in JavaScript expressions and statements).

When run on programs with the original variable names obfuscated, the implemented system was able to recover the original variable names over 60% of the time. The results for type annotations are even more intriguing — the implemented system generates correct type annotations for over half of the benchmark programs. For comparison, the programmer-provided annotations are correct for only a bit over a quarter of these programs. The system is accessible via the Internet at jsnice.org with hundreds of thousands of users.

These results demonstrate how this approach can help JavaScript programmers produce more easily readable and understandable programs. One potential longer range consequence could be the gradual emergence of a de facto standard for aspects of JavaScript programs such as variable names and the relationship between program structure and types. More broadly, the results also highlight the substantial redundancy present in JavaScript code worldwide and raise questions about just how much human effort is really required to produce this code.

So why was this research so successful? First, the investigators chose a problem that was a good fit for machine learning over big code. Current machine learning techniques do not provide correct results; they instead only provide results that look like previous results in the training set. A variable name or type annotation predictor does not have to always be correct; it only needs to be correct enough of the time to be useful. And JavaScript programs share enough variable name and type annotation patterns to support a reasonably accurate model.

A second reason is technical, specifically the development of a program representation that exposes relevant relationships between variables and the surrounding context, including how variables are used in JavaScript statements and expressions. Features exposed in this program representation enable the immediate application of conditional random fields, a standard technique in machine learning for structured prediction previously shown to be effective for solving problems in areas such as natural language processing and computer vision, to solve the learning and prediction problem. The development of a new approximate MAP inference algorithm for this domain enables the performance required for interactive use when working with thousands of labels per node (in contrast to many previous applications, which only work with tens of labels per node).

So what can we expect to see in the future from this line of research? The most obvious next steps include a variety of automated programming assistants for tasks such as code search, code completion, and automatic patch generation. Here the assistant would interact with the programmer to guide the process of turning vague, uncertain, or underspecified goals into partially or fully realized code, with programmer supervision required to complete and/or ensure the correctness of the resulting code.

It is less clear how to make progress on programming tasks with more demanding correctness, autonomy, or novelty requirements. One critical step may be finding productive ways to integrate probabilistic reasoning with more traditional logical reasoning as applied to computer programs. Future research, potentially inspired in part by the results presented in this paper, will determine the feasibility of this goal.