Assignment 1
WASP Software Engineering Course

Valency Oscar Colaco

September 14, 2022

Jupyter Notebooks or Google's Colab are the most popular choice of tools for software developers (rapid prototyping) or machine learning specialists in the early stages of the ML workflows. The interactive nature of these notebooks makes them very effective for data-centric programming, and their self-documenting nature makes it possible to express analytical insights with great ease. The notebook format, however, is prone to bad programming practices and this causes issues during the production and deployment stages of the software development lifecycle. To prevent these issues and foster the creation of better, well structures and documented notebooks, Pynblint was developed. Pynblint is a static analyzer for python Jupyter notebooks that was developed at the University of Bari, Italy. This tool ensures that notebooks (and related repositories) adhere to a set of best programming practices, and generates specific recommendations when breaches are found.

During my individual session with Luigi, we went through one notebook that was related to the training of a Random Forest for handwritten digit recognition. The tool was easy to install within Google Colab, but it refused to run (due to multiple conflicting dependencies). We encountered the same issue when we tried to run the tool on my WSL (Windows Subsystem for Linux) Ubuntu Virtual Machine. The issue was addressed when we set up a python virtual environment, a self-contained directory tree where there was no conflicting or overlapping dependencies. My initial observation was that this 'conflicting dependencies' issue greatly affected the utility of the tool (and its appeal as a static analyzer) as creating a virtual environment each time just to use the tool is not feasible. As soon as we fixed that problem, we ran a preliminary test on my notebook.

While most of the recommendations generated by the tool were perfectly valid best practices for structured code followed by Industry and Academia, there was one recommendation that just didn't feel right. The non-linear code execution recommendation pops up when you have cells in your notebook that are not run in order. As a machine learning researcher, a significant part of rapid prototyping for ML models is 'hyperparameter tuning' in which we tweak the parameters of the model to get a desired outcome. This involves going back, changing the parameters and evaluating the model again and again. This process inevitably leads to a non-linear execution of the cells in the notebook (which isn't necessarily a bad thing). On the other hand, this tool doesn't require extensive command line knowledge which is excellent for novice prototypers. In course of conversation with Luigi, he also told me that they are working on a web-based version of Pynblint, and making the tool proactive, i.e., capable of automatically fixing a selected set of violations, which I think is a step in the right direction.

I use notebooks almost on a daily basis as I prefer the interactive and real-time code execution experience. While I eventually move to an IDE (like VScode) for the final stages of programming, almost all my prototyping begins with notebooks. This makes Pynblint a particularly useful tool in my WASP research as it can help me produce structured code that follows empirically validated best practices from Industry and Academia. Apart from static analysis, linters can also be used for educational purposes to help students understand the best programming practices used in software development. Researchers have shown that the open-source repositories are filled with poor-quality, unstructured and hard-to-follow notebooks. Integration of a tool like Pynblint into code hosting platforms (like GitHub or BitBucket) can greatly help improve the readability and consistency of publicly available python notebooks.

While Pynblint checks the high-level structural features of notebooks, it does not check the syntax of the code itself, which is counter-intuitive as this is one of the basic functions of any python linter. This tool could be even more useful if we could import third-party repositories, for example, code that performs formal verification, into our notebooks. This would have the added benefit of both static analysis of the code and formal verification of the software at the same time, which would tremendously improve its utility, i.e., well-structured code and formally-verified software.