Software Engineering, Assignment 1: Pynblint Reflection Report

Jennifer Andersson

August 1, 2022

On Thursday, June 30, I took part in a private Pynblint trial session with creator Luigi Quaranta. Pynblint is a static analysis tool developed for Python Jupyter Notebooks. Having limited personal experience with Jupyter Notebooks, my preparatory research prior to the private trial session with Quaranta and the following discussion with him gave me a lot of new insights as to when and why notebooks can be a beneficial part of the research workflow, especially in ML. In my previous work, I have mainly used the notebook format for teaching and tutorials, but during my session I learned about the power of using notebooks for presenting and showcasing research results in the early stages of the working process. My biggest takeaway in this regard is that notebooks can be a great way to showcase results and ML prototypes in a reproducible way to colleagues and collaborators.

During the discussion, I learned a lot about how static analysis tools for notebooks can be used for tutorials and teaching. Pynblint allows for best practice violation warnings for notebooks, referred to as linting rules. In a teaching setting, which is the context in which my trial notebooks have been used before, this can be especially effective to ensure that a collection of notebooks used in a particular course are structured similarly and named coherently. Incorporating such a tool makes it easier to follow best practice, which in turn makes it easier for anyone to quickly understand and use notebooks written by others. I found some linting rules to be particularly useful in the tutorial context. For example, the linting rule *non-executed cells are present in the notebook* can be used as a check that fill-in cells (to be filled in by students) are not accidentally executed when the notebook is accessed by the students, or that cells that should be executed as part of a task are not executed prior to distribution. For similar reasons, the linting rule *non-linear-execution* can be very useful both in a teaching and in a research context, to help ensure reproducibility of the conveyed results. This is important in ML not the least, since random seeds, hyperparameters and the order in which data is generated and processed in different parts of the code has a high impact on the model performance and results produced.

I will continue to use notebooks for teaching obligations as part of my WASP PhD, and Pynblint is a tool that will definitely ease the process of creating tutorial notebooks by taking away some of the manual inspection load. We did not get into a deeper discussion about using Pynblint for ML applications specifically, but our more general discussion convinced me of the power of using notebooks for presenting research results in a coherent and reproducible way, which is streamlined by the Pynblint static analysis tool. I will start using notebooks for presenting early and intermediate stage research results to my supervisor and other collaborations, and it will be interesting to see how the numerous linting rules will be beneficial to this part of the research process. From our discussion, my impression is that using the Pynblint static analysis tool may be more beneficial in larger projects where many people are collaborating and there are multiple notebooks in each project, making it all the more important to follow best practices and a shared structure. For instance, I learned about specific linting rules beneficial in these circumstances, such as recommending the user to make data available, which concerns reproducibility and can be especially important in an ML setting. Though I did not have the opportunity to try Pynblint for a larger project, the properties allowing for this work structure seem to be a very good addition in the ML community, as large projects with a lot of intermediate showcasing of model prototypes and results – where notebooks can be beneficial - should be on the increase, and it is very good to be able to get as much feedback as possible without having to re-execute a complex chain of codes.

We also discussed possible improvements to the Pynblint tool, especially regarding the use of Jupyter notebooks for teaching purposes. This is a very common application of notebooks at uni-

versities and in other avenues where tutorials are frequently used. During my session, I learned that linting rules can be customized and inactivated in the current version of Pynblint. As an extension of this, one feature that would be very useful in tutorial notebooks in particular is to have an additional linting rule for fill-in cells where students are supposed to finish a code snippet or write a small part of a larger code, preferably with the possibility for the user to mark which cells or part of cells (for example within a for loop) this feature should be applied. This can allow for warning the user when there is already code where the students should fill in the code themselves, or silence other warnings like unexecuted cells. Other improvement suggestions include the integration of Pynblint with for example Jupyter lab, as using the terminal for static analysis can be impractical for some users.

Static analysis can be beneficial to assure the use of best practice in the structure of large ML projects. However, one aspect that limits the use of static analysis tools in ML is how heavily dependent an ML project is on data. Improvements to Pynblint in this regard include handling dimension accuracy in a large pipeline and including a data linting tool that can statically analyze ML datasets depending on best ML practices. As discussed with Quaranta, this is possible (at least for some standard data structure and standard ML methods) but not straightforward to incorporate in static analysis tools, especially when datasets grow very large and models have specific data requirements. Another main limitation is the black box-structure commonly encountered in ML projects, using readily available model implementations where the model is iteratively updated as the code is executed. In-depth analysis of such code without execution often requires deep mathematical knowledge and knowledge about the implementation of specific ML architectures and libraries, based on information available at run-time. In general, dynamic analysis tools are better equipped for analyzing these aspects of such codes, as executing the code can provide a lot more information about convergence properties and what goes on inside the model.