Assignment 1

WASP Software Engineering Course

Amirhossein Ahmadian

June 2022

During the individual session for practicing Pynblinter, we went through two Jupyter notebooks with Python code that were already run and now given as the input to the linter tool. These notebooks were related to my course assignments in deep learning and stochastic optimization. I found Pynblinter very easy to install and use, in the sense that basically you can see the results by running a single line of command. The warnings and recommendations that I received from the tool were mostly related to the general structure of notebooks. For both of the notebooks, the nonlinear (not in order) execution of cells was a detected issue. The way to fix it is to run each cell of the notebook once in order from the top to the bottom. This can ensure correct results and reproducibility, although I think while working on a notebook in early stages, in many cases you have to re-run the cells in different orders, especially when you are trying to fix a problem or explore some idea in your implementation. The other message shown for my notebooks was the recommendation to shorten the codes in each of the notebook cells as well as the whole notebook. Keeping code cells short is in practice equivalent to writing more structured and concise code, in the sense that it encourages use of functions/classes and avoid repetition in coding. Moreover, the whole notebook can be made smaller by writing some parts of code (e.g., auxiliary functions) outside of the notebook, and importing them to the notebook using 'import' command. I agree with this recommendation in general, as 'better structured code' is a well-known good practice in programming, though I think sometimes using many external dependency files can make the notebook non-standalone and harder to deploy. Other messages shown by Pynblinter were simpler, including warning on having empty cells in the notebook, and recommendation to use the import command only in the first code cell of the notebook. I also learned that Pynblinter can save its output in a file with JSON format.

At the moment, in my PhD research (which is in the area of machine learning and deep learning), I am more used to using IDE software (e.g., VS Code) and GitHub repositories to develop and manage my programming projects, and I rarely work with notebooks. However, I got some tips and pointers regarding the advantages of leveraging notebooks in research, which may attract me to notebooks (and therefore Pynblinter) more in future. Now, I understand that notebooks can be a nice way to present and share your ideas with your supervisor, for instance, or with anyone collaborating in the same project. For example, one style that I may follow later is to present the results of my latest experiments and their explanations in the format of a Jupyter notebook in the regular meetings with my PhD supervisor, such that we can play with some parameters and see the updated results of algorithms during the meeting. Also, in case I collaborate on coding with some other students/researchers, we may use a notebook as a shared place for some parts of the project, including running test/example cases and investigating results, for which having a linter would be handy. Nevertheless, application of notebooks in meetings and collaboration during a PhD study is partly limited by the traditions in the research group, time available for meetings, and type

of the problems of interest (e.g., it may take very long to re-run a deep learning code). Another way that I can leverage notebooks in my PhD project is to accompany my published papers with a Jupyter notebook (instead of /in addition to code on GitHub) with the purpose of reproducibility, illustrating the experimental results, and providing supplementary information, a fashion that I have seen with a few papers in deep learning. I believe applying a linter on the notebook before officially publishing it would be particularly important in such cases to ensure its readability and reproducibility.

Pynblinter is currently focused on more high level and structural features of notebooks, and does not check the syntax and details of the notebook code cells (e.g., coding style) like what a regular Python linter does. This was also acknowledged by the tool author, who emphasized that Pynblinter does not perform much standard static code analysis at the current version but its integration with a standard Python linter is in progress for future versions. This integration is very helpful in my opinion since a large part of a notebook is 'coding' as we know it, while most popular coding tools (IDEs and linters) today do not have a good support for notebooks. Adding regular code linting to a notebook linter is not trivial of course, because there are some differences between the intended behavior of code in notebooks and code in standalone scripts. Specifically, the code in a notebook is sometimes run to only demonstrate an output, which means for example you may compute a value without assigning it to any variable. Another interesting line in the Pynblinter project could be to set up an extensive documentation or knowledge base (preferably as a Wiki) which introduces and explains all the warnings and recommendations that one may get from the linter tool as output. More precisely, each message (warning/recommendation/etc.) can be linked to a page in the knowledge base, where additional advice for solving the issue and related examples are provided. Adding a graphical user interface, which can visually mark the problematic cells inside the notebook for instance, is my other suggestion. Moreover, since usually a primary part of any good notebook is text descriptions and figures, it could be wonderful to integrate the linting tools with tools such as spell checker, and add options for checking proper rendering of figures. In fact, during my personal session, I realized that an intended figure in one of the notebooks was broken (due to missing file) but this issue was not detectable by Pynblinter.

I think static analysis for machine learning is challenging generally speaking because of the rapid advancement and high diversity of methods in the area of ML and deep learning. On the data side, we should not overlook the differences that exist in various forms of data structures. Although tabular data are a substantial part of databases that provide input to ML methods, other forms of data such as image, voice, graphs, and text are obviously not less valuable in today's applications. Some of these forms are less structured than others and thus more complicated to check. Besides, application-specific aspects of data are also broad, and new modalities of data emerge from time to time in this sense (e.g., data in 'genetics' and 'internet-of-things' can be both sequential but may need different standards). I wonder if it is possible to design a data linter that can cover the full range of different structures and applications of data. On the code and model side, although linters can help a lot, I believe one limitation is that some properties of a ML algorithm in which developers are interested can be almost impossible to obtain without executing the algorithm, and thus not reachable by static analysis tools. For example, 'convergence' of a complicated training algorithm (specifically, in case of Generative Adversarial Networks) is of great importance but is very hard to be predicted in a mere static fashion, as far as I understand. The problem even becomes more complicated when some randomness exists in the ML algorithm (e.g., stochastic optimization).