Static Analysis for Notebooks Reflection

Vincent Szolnoky

I learnt that there are indeed tools for static analysis for Jupyter notebooks and currently enforce many of the same rules as static analyzers for non-ML code. Additionally they aim to make sure that a certain structure is followed in the notebook which is a good step in ensuring consistency across all notebooks.

I probably will not use pynblint for a simple reason which is that I don't write my code in Jupyter notebooks specifically. My main gripe with Jupyter notebooks is that they allow non-linear execution of code blocks, which is actually one of the problems pynblint tries to warn against. However I believe that this is an inherent flaw in Jupyter notebooks that should be allowed in the first place. Notebooks should be written as a Directed Acyclic Graph (DAG) instead. Therefore I write my code in Org Babel documents (part of Emacs) which allows one to write code blocks using *noweb* references which is a core part of the literate programming paradigm (https://en.wikipedia.org/wiki/Noweb). Using noweb references, one can write code blocks such that they are always executed linearly and that running any code block will also run any other code blocks it references. For my code this has allowed me to completely isolate individual experiments with, for example, different paramteters but **with out** duplicating code unnecessarily.

A large problem with Jupyter notebooks that should be addressed is the use of "escape hatches", for example running shell commands. These are in general very hard to ensure reproducibility. If one uses them to download data or a library then it should be ensured that the same version is downloaded each time, which is rarely the case. Therefore I think pynblint should incorporate a linter which in the best case could detect when these escape hatches lead to unreproducible results. Alternatively report them as warnings so that the developer can try find an alternative way for handling them.

I think the limits for static analysis for ML code is the fact that the complexity is often not in the code itself but what is actually being modelled. Therefore for static analyzers to really be useful for ML, they have to understand the implementation of the code. A simple example of this could be accidentally using an incorrect loss function or activation function when building and training a neural network. Similarly, data is a large part of ML code that is not related to the code itself. Data leakage, i.e. test data that is part of the training set, is a common problem that should be detected early by a static analyser.