ML for Code Prediction

PROBLEM

Software companies are always striving to improve their own coding tools to improve programming productivity. It is a beneficial, self-reinforcing cycle to better the tools you use to create more tools. Common examples include linting, and auto-complete routines in IDEs (Integrated Development Environment). An important aspect of this quest is to understand a piece of code in a holistic manner, not just predicting the next word in code, but debugging by predicting intentions, suggesting comments, even moving out-of-place codes.

CHALLENGE

The programming challenge involves creating a ranking algorithm to sort a dataset of shuffled, anonymized Jupyter Notebooks that is submitted to Kaggle coding challenges recently. The notebooks mainly run various Machine Learning algorithms on python, due to the nature of the platform.

SOLUTION

The main difficulty of this challenge is that there is no straightforward machine learning algorithm for ranking. Therefore one must improvise and adapt. Our solution involves first and foremost reframing the problem as one of classification, by aiming to train classifiers, when fed with pairs of texts or code, would determine the likelihood that one would be on top of the other. This has a small chance of success on its own, as there is not much information to be gleaned from single cell on Jupyter Notebook. Nevertheless, when a document is taken as a whole, the network of these comparison relations may then be utilized to predict the overall ranking of code and comment snippets to a reasonable degree.

A second aspect of the dataset we may be able to exploit is the internal structure of code. If the notebook runs without error, then there is a certain ordering between its L-values (assigned variable) and R-values (assignment value). In short, one must declare a variable before one can use it. This information should be deterministic if the code is well-maintained, and wouldn't need the help of machine learning to figure, but a lot of coding repositories are unfinished or buggy codes, so a probabilisitic analysis could lead to results.

CONCLUSION

We have built a plausible framework to predict code ordering and its relationship to comments and Markdown scripts. This could potentially lead to increased productivity in various ways for anyone who would code as part of their job.