# Automated Assessment of Quality of Jupyter Notebooks Using Artificial Intelligence and Big Code

**Priti Oli, Rabin Banjade, Lasang Jimba Tamang, Vasile Rus**
Department of Computer Science, Institute of Intelligent System
University of Memphis, Memphis, TN, USA
{poli,rbnjade1,ljtamang,vrus}@memphis.edu

**Abstract**

We present in this paper an automated method to assess the quality of Jupyter notebooks. The quality of notebooks is assessed in terms of reproducibility and executability. Specifically, we automatically extract a number of expert-defined features for each notebook, perform a feature selection step, and then trained supervised binary classifiers to predict whether a notebook is reproducible and executable, respectively. We also experimented with semantic code embeddings to capture the notebooks' semantics. We have evaluated these methods on a dataset of 306,539 notebooks and achieved an F1 score of 0.87 for reproducibility and 0.96 for executability (using expert-defined features) and an F1 score of 0.81 for reproducibility and 0.78 for executability (using code embeddings). Our results suggest that semantic code embeddings can be used to determine with good performance the reproducibility and executability of Jupyter notebooks, and since they can be automatically derived, they have the advantage of no need for expert involvement to define features.

## Introduction

In this paper, we present a machine learning-based approach to automatically assess the quality of Jupyter notebooks, which, together with Python, are key elements in fully harnessing the data revolution in many domains. Data and Data Science have the potential to improve products, services, and processes and are already impacting science, business, economics, engineering, and government. Thus, the quality of Jupyter notebooks can have a significant impact across many domains.

Jupyter Notebooks are the most widely-used tools for data science that make it easier to create and share live code, results, visualizations, and narrative text. The popularity of Jupyter notebooks for literate programming and computational narrative is corroborated by the fact that there are over 2.5 million Jupyter notebooks in GitHub as of September 2018, which is ten times more than in 2015 (Wang, Li, and Zeller 2020). (Kery et al. 2018) reveal three use cases of Jupyter notebooks: preliminary scratchpad, production pipeline, and work intended to be shared. When used as production pipelines and for sharing, the quality of notebooks is very important as errors can easily propagate. Unfortunately, many shared notebooks are of low quality, which is a major challenge that needs to be addressed, as explained next.

A recent study found that only 24.11% of a large number (863,878 attempted executions) of notebooks in GitHub ran without errors and only 4.03% produced the same results. The study reveals the need to have ways to assess the quality of Jupyter notebooks at creation or publishing time. Bad notebooks can propagate bad practices among professionals in various fields and among members of the scientific community. since Jupyter notebooks are heavily used for literate programming and sharing of scientific results (Shen 2014), the quality of shared notebooks can have cascading effects.

To address these issues, we describe here methods to automatically assess the quality of existing Jupyter notebooks. The methods can be used either at notebook creation time or at publishing time. Thus, the proposed methods and resulting tools can be used by notebook authors as a quality assurance step, prior to publishing their notebooks.

Our definition of notebook quality is based on the principle that any shared notebook of good quality should execute correctly and be easily read, understood, and their results reproduced by other users, e.g., fellow programmers, scientists, or other professionals such as business analysts(Wilson et al. 2014). Therefore, we define the quality of Jupyter notebooks using the following two proxies: (i) executability and (ii) reproducibility. Executability is the ability of a program to run without any execution errors. In our case, we define executability as the ability to run all the cells of a Jupyter notebook without any errors. Similarly, reproducibility is the extent to which consistent results are obtained when experiments are repeated. In our case, we define reproducibility as the ability of all the cells of a Jupyter notebook to produce the same set of results for the same set of inputs. Therefore, our operational goal is to develop automated methods that can predict whether a notebook is executable and reproducible. To this end, the proposed general approach to assessing notebooks' quality is based on recent advances in Artificial Intelligence/Machine Learning, e.g., deep learning, and Big Code, i.e., large repositories of notebooks such as GitHub.

To assess the quality of Jupyter notebooks, we experimented with the following two approaches: classical Machine Learning (ML) and Deep Learning (DL), i.e., automatically derived code representations in the form of embeddings. For the classical ML approach, we defined a set of features/predictors after which we trained various ML algorithms. For the predictors to be used, we did an extensive analysis of the literature on software quality metrics. For instance, we started with the metrics suggested by Pimentel and colleagues (2019) and others (Biswas et al., 2019) such as the length of notebook titles, the placement of imports, the presence of dependency requirements files, and the use of relative paths to access the data. Similarly, for the DL-based approach, we adopted a highly successful approach that has been developed recently by the DL community: automatically learn suitable representations, i.e., embeddings (Efstathiou & Spinellis, 2019). Such representations are known to improve the performance of downstream learning tasks or applications such as contextual search and analogical reasoning in the case of natural language semantics. We investigate here to what extent such learned representations are suitable for detecting the quality of Jupyter notebooks. DL methods have been proven to be very good at analyzing large repositories of data related to a given task and discover statistical regularities. Similarly, our hypothesis is that they will be good at capturing regularities about software artifact quality.

In sum, we developed, validated, and report results with two categories of novel methods to assess the quality of Jupyter notebooks. The goal is to explore a process to develop methods that automatically assess the quality of notebooks, investigate how good machine learning methods are at predicting notebook quality, offer insights into what are the major characteristics of quality notebooks, and what best practices are needed in terms of development processes using notebooks to assure their quality during creation and publication.

We answer here the following two key research questions:

- Q1: Is it possible to predict the reproducibility and executability of a Jupyter notebook based on expert-defined features?
- Q2: Can semantic notebook embeddings accurately predict notebooks' reproducibility and executability?

## Related Work

Several studies have focused on various aspects of Jupyter notebooks. (Rule, Tabard, and Hollan 2018) described three case studies about how notebooks are used to document and share exploratory data analyses. They found that one in four notebooks in a set of over 1 million notebooks in GitHub had no explanatory text. (Kery et al. 2018) highlights that, as notebooks go through many changes and grow in size they often are difficult to understand, limiting sharing and, when shared, limits their use to others. To address this issue (Head et al. 2019) proposed and developed code gathering tools, extensions to computational notebooks that help analysts find, clean, recover, and compare versions of code in cluttered, inconsistent notebooks.

(Pimentel et al. 2019) presented a study of various structural characteristics of Jupyter notebooks as well as their reproducibility. In their work, they collected a large corpus of more than 1 million notebooks, which we also use in our work reported here, and executed each notebook to determine their executability (execution with no errors) and reproducibility (reproduced the same results). Based on their analysis, they outline a set of recommendations in the form of suggested best practices. Our work here proposes an automated method to predict reproducibility and executability and in the process automatically extract and identify the most discriminating features that define reproducibility and executability.

An interesting line of research relevant to our work is on automatically inferred representations of code, i.e., code embeddings. (Zhang et al. 2019) uses a combination of unsupervised representation learning and weak supervision for computing joint representations of code from both abstract syntax trees and surrounding natural language comments. Similarly, (Pradel and Sen 2018) proposed DeepBugs to identify name-based bug detection using semantic representations of code. Likewise, (Efstathiou and Spinellis 2019) proposed distributed code representations for six different programming languages: Java, Python, PHP, C, C++, and C#. They used fastText for learning semantic representations and studied dissimilarities between code and natural language, proposing various applications and limitations. In related work, (Kanade et al. 2019) used BERT embedding of source code for five benchmark classification tasks: variable misuse classification, wrong binary operator, swapped operand, function-docstring mismatch, and exception type. While prior work focused on either the study of various aspects of reproducibility of Jupyter notebooks or on the representation of source code, we explore a combination of the two, i.e., using effective Jupyter notebook representations for assessing reproducibility.

## Approach

As noted, we model both reproducibility and executability as a binary classification task. We explored both a standard ML method where the features or predictors are expert-defined and a novel method based on code embeddings, i.e., automatically inferred representations of notebooks in our case.

**Feature based classification** For the standard ML method, we started with identifying a set of features to predict both the reproducibility and executability of Jupyter notebooks. We defined a set of features based on an extensive analysis of the literature on software quality metrics. (Wilson et al. 2014) describes the following code characteristics as indicators of software quality and best practices for scientific computing: modularizing code, documentation of design and purpose, the embedding of documentation, consistent code style, and formatting, consistent, distinctive, and meaningful names, use of version control, re-use of code, testing of code using assertion, and unit test. Based on these expert-defined features, we hand-picked 42 features for our classification.

**Feature selection** Instead of using all the 42 hand-picked features, we selected a subset as a way to avoid over-fitting a model with too many features. The goal is to design a simpler model that should generalize better according to Occam's Razor principle. Using the Random Forest feature selection algorithm, we identified 32 features that are more discerning in terms of information gain for both reproducibility and executability. Based on this feature importance analysis and selection, we note that the length of meaningful words, meaningful lines, the number of markdown cells, use of modules, use of assertion, and exception handling are the most important features for reproducibility and executability. Table 1 shows the 32 selected features included in the model for binary classification of both reproducibility and executability.

| Feature | Description |
|---|---|
| code_cells | no. of code cells |
| markdown_cells | no. of markdown cells |
| raw_cells | no. of raw cells |
| functions_decorators | no. of functions with decorators |
| classes_decorators | no. of classes with decorator |
| total_classdef | no. of class definitions |
| ast_module | no. of modules in the repository |
| ast_statements | no. of statements |
| ast_raise | no. of statements with exception |
| ast_try | no. of try statements |
| ast_tryexcept | no. of try-except statement |
| ast_tryfinally | no. of try-finally |
| ast_assert | no. of assertion made for test |
| total_import | no. of modules imported |
| len | length of words (markdown) |
| lines | no. of lines (markdown) |
| meaningful_lines | no. of lines in markdown |
| meaningful_words | no. of words in markdown |
| local_importfrom | no. of local module imported |
| global_assign | no. of global assignments |
| class_assign | no. of class assignments |
| local_assign | no. of local assignments |
| class_functiondef | no. of class function definitions |
| local_functiondef | no. of local function definition |
| ast_ifexp | no. of if expression |
| ast_lambda | no. of lambda expression |
| ast_dict | no. of dictionary |
| ast_set | no. of set |
| ast_listcomp | no. of list comprehension |
| ast_lambda | no. of lambda functions |
| ast_dictcomp | no. of dictionary comprehension |
| ast_param | no. of parameters used |

Table 1: List of selected features based on information gain.

## Notebooks Quality Embeddings

In this approach, we used deep neural networks to automatically learn embeddings of source code. We could have used both the code and the narrative text of the notebooks to generate the embeddings. However, in the work presented here, we only used the code to generate the notebook embeddings and for this reason, we refer to them as code embeddings. We fine-tuned Elmo, a deep learning-based contextualized semantic representation model to get distributed vector representations of source code in IPython notebook (Peters et al. 2018). The output from the ELMO embedding is a vector of size 1,024, which we used as a feature vector for the classification step. The Elmo embedding was then fed to a dense layer of 256 nodes with a ReLU activation. The output of the dense layer was fed to a prediction layer with softmax activation to predict one of the two binary classes for each of the two dimensions of notebook quality that we operate with: executability and reproducibility. The loss was calculated using binary cross-entropy; an Adam optimizer was used for iterative updating of weights, i.e., training.

## Experiments and Results

In this section, we first present the data we used for our experiments, the experimental setting, and the results obtained.

### Data Description

To evaluate the methods described above, we used a dataset of annotated Jupyter notebooks from (Pimentel et al. 2019). The dataset consists of notebooks written in Python, R, Julia, and Scala. We only retained the Python notebooks that are related to data science and containing only Roman characters. Also, we discarded any notebooks that have less than two cells. After discarding the duplicate notebooks and notebooks flagged as assignments or homework, we ended up with a total of 306,539 notebooks which we use for our experiments.

Of the 306,539 selected notebooks, only 4.4% were found to be reproducible and only 4.69% executable. Because only a small fraction of the notebooks in the dataset were reproducible/executable, to avoid problems of highly skewed data for further analyses, we decided to generate a balanced subset through under-sampling. After under-sampling, the resulting dataset of 32,188 notebooks is balanced: 16,094 reproducible notebooks and 16,094 non-reproducible notebooks. Similarly, we obtained a balanced dataset of 34,372 notebooks for executability: 17,186 executable notebooks and 17,186 non-executable notebooks.

### Results for Machine Learning Models based on Expert-defined Features

We have used four standard machine learning classifiers in our experiments: Decision Trees, Random Forests, Multi-layer Perceptron (MLP), and Naive Bayes.

***Reproducibility:*** Table 2 shows the results for the different classifiers for predicting reproducibility. We can see that Random Forest outperforms other classifiers on all performance metrics (precision, recall, F1, kappa) in predicting the reproducibility of the Jupyter notebooks. All classifiers perform significantly better than chance as indicated by Cohen's kappa scores.

***Executability:*** Table 3 shows the performance of the different classifiers for predicting executability. We note that Random Forest again has the best performance across all metrics in predicting the executability of Jupyter notebooks.

| Classifier | F1 Score | Precision | Recall | Cohen's Kappa |
|---|---|---|---|---|
| Decision Tree | 0.82 | 0.81 | 0.81 | 0.64 |
| Random Forest | 0.87 | 0.87 | 0.87 | 0.73 |
| MLP | 0.73 | 0.76 | 0.73 | 0.46 |
| Naive Bayes | 0.64 | 0.67 | 0.64 | 0.29 |

Table 2: Results for reproducibility.

| Classifier | F1 Score | Precision | Recall | Cohen's Kappa |
|---|---|---|---|---|
| Decision Tree | 0.93 | 0.93 | 0.93 | 0.86 |
| Random Forest | 0.96 | 0.96 | 0.96 | 0.91 |
| MLP | 0.89 | 0.90 | 0.89 | 0.79 |
| Naive Bayes's | 0.71 | 0.79 | 0.71 | 0.43 |

Table 3: Results for executability.

### Results with Code Embeddings

We ran the previously described DL classifiers with Elmo embeddings for 4 epochs on the balanced datasets for reproducibility and executability, respectively. Table 4 summarizes the results. We can see that the DL classifier performs well relative to the classical ML classifiers for reproducibility. The performance of the DL approach in the case of executability is still good but less comparable to that of the classical ML approach. However, given the fact that the DL approach does not need expert-defined features, it is a more scalable and portable approach, i.e., easily portable to assess the quality of notebooks for other languages.

| Quality metric | F1 Score | Precision | Recall | Cohen's Kappa |
|---|---|---|---|---|
| Reproducibiltiy | 0.81 | 0.81 | 0.81 | 0.42 |
| Executability | 0.78 | 0.78 | 0.78 | 0.35 |

Table 4: Reproducibiltiy and executability performance results using the deep-learning approach (code embeddings).

## Conclusions and Future Work

This paper presented our investigation of the role of both classical machine learning and deep learning-based approaches to predict the quality of Jupyter notebooks Notebook quality was operationalized using the two criteria of reproducibility and executability. While the classical machine learning approach performed better overall, the deep learning approach has the advantage of being more scalable and portable. The proposed methods can be used to check the quality of publicly released notebooks or future notebooks before being shared and will thus lead to improved notebooks with broad downstream impact resulting in better science through better data analysis and software processes and artifacts.

Plans for future work include experimenting with methods that represent code using Abstract Syntax Tree and generate embeddings from both the code and the narrative text of the notebooks.

## Acknowledgments

This work has been supported by two NSF awards: the Learner Data Institute (NSF; award 1934745) and CSEdPad: Investigating and Scaffolding Students' Mental Models during Computer Programming Tasks to Improve Learning, Engagement, and Retention (NSF award 1822816). The opinions, findings, and results are solely the authors' and do not reflect those of NSF.

## References

Efstathiou, V., and Spinellis, D. 2019. Semantic source code models using identifier embeddings. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 29–33. IEEE.

Head, A.; Hohman, F.; Barik, T.; Drucker, S. M.; and DeLine, R. 2019. Managing messes in computational notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–12.

Kanade, A.; Maniatis, P.; Balakrishnan, G.; and Shi, K. 2019. Pre-trained contextual embedding of source code. *arXiv preprint arXiv:2001.00059*.

Kery, M. B.; Radensky, M.; Arya, M.; John, B. E.; and Myers, B. A. 2018. The story in the notebook: Exploratory data science using a literate programming tool. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–11.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Pimentel, J. F.; Murta, L.; Braganholo, V.; and Freire, J. 2019. A large-scale study about quality and reproducibility of jupyter notebooks. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 507–517. IEEE.

Pradel, M., and Sen, K. 2018. Deepbugs: A learning approach to name-based bug detection. *Proceedings of the ACM on Programming Languages* 2(OOPSLA):1–25.

Rule, A.; Tabard, A.; and Hollan, J. D. 2018. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–12.

Shen, H. 2014. Interactive notebooks: Sharing the code. *Nature* 515(7525):151–152.

Wang, J.; Li, L.; and Zeller, A. 2020. Better code, better sharing: on the need of analyzing jupyter notebooks. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*.

Wilson, G.; Aruliah, D. A.; Brown, C. T.; Hong, N. P. C.; Davis, M.; Guy, R. T.; Haddock, S. H.; Huff, K. D.; Mitchell, I. M.; Plumbley, M. D.; et al. 2014. Best practices for scientific computing. *PLoS Biol* 12(1):e1001745.

Zhang, J.; Wang, X.; Zhang, H.; Sun, H.; Wang, K.; and Liu, X. 2019. A novel neural source code representation based on abstract syntax tree. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 783–794. IEEE.