User-centered & Robust Open-source Software: Lessons Learned from Developing & Maintaining *RSMTool*

Nitin Madnani
Educational Testing Service
Princeton, NJ
nmadnani@ets.org

Anastassia Loukina
Educational Testing Service
Princeton, NJ
aloukina@ets.org

Abstract

For the last 5 years, we have developed and maintained *RSMTool* – an open-source tool for evaluating NLP systems that automatically score written and spoken responses. *RSMTool* is designed to be cross-disciplinary, borrowing heavily from NLP, machine learning, and educational measurement. Its cross-disciplinary nature has required us to learn a user-centered development approach in terms of both design and implementation. We share some of these lessons in this paper.

1 Motivation

Automated scoring of open-ended written and spoken responses is a fast growing field in educational NLP. Many automated scoring systems employ machine learning models to predict scores for such responses based on features extracted from the text/audio of such responses. Examples of deployed automated scoring systems include Project Essay Grade[1] for written responses and SpeechRater®[2] for spoken responses (Zechner et al., 2009; Chen et al., 2018). Automated scoring systems may offer some advantages over humans, e.g., higher score consistency (Williamson et al., 2012). Yet like any other machine learning algorithm, models used for score prediction may inadvertently encode discrimination into their decisions due to biases or other imperfections in the training data, spurious correlations, and other factors (Xi, 2010; Romei and Ruggieri, 2013; von Davier, 2016; Zieky, 2016). Given that many such systems are used to score high-stakes standardized tests, the consequences of any form of bias can have a significant effect on people's lives. Therefore, it is critical that automated scoring systems be evaluated as thoroughly as possible to detect any harmful, systematic biases in their predictions. However, this may prove difficult for an NLP or machine learning researcher since they may be unfamiliar with the required psychometric and statistical checks. *RSMTool* incorporates a large, diverse set of psychometric and statistical analyses aimed at detecting possible bias in system performance and makes them available in an easy-to-use package. *RSMTool* is open-source and non-proprietary so that the automated scoring community can not only audit the source code of the already available analyses to ensure their compliance with fairness standards but also contribute new analyses.

2 Introduction

Creating and releasing open-source software is a great way to share knowledge by making highly-specialized methods and techniques accessible to a wider community. Yet many NLP (or machine learning) tools are not designed with a user-centered focus, hindering their wider adoption. Almost a decade ago, Chapman et al. (2011) pointed out that NLP systems were seldom deployed in clinical settings because they were not well integrated into existing user workflows and required substantial input from an NLP expert. More recently, Cai and Guo (2019) found that a "steep learning curve", the need to convert raw data into algorithmic inputs and outputs, and various challenges in getting started are still the most common hurdles reported by software engineers when using open-source machine learning software.

In the subsequent sections, we share our experiences of developing and maintaining *RSMTool*, an open-source Python tool[3] which provides a framework to evaluate systems for automated scoring of

[1] https://www.measurementinc.com/products-services/automated-essay-scoring

[2] https://www.ets.org/accelerate/ai-portfolio/speechrater

[3] https://github.com/EducationalTestingService/rsmtool

written and spoken responses (Madnani and Loukina, 2016; Madnani et al., 2017). *RSMTool* is developed as part of a close collaboration between NLP researchers and specialists in educational measurement, assessment and psychometrics. It combines the latest advances from these disciplines to provide a comprehensive evaluation of automated scoring systems, including model fairness and test-theory based measures.

RSMTool was initially developed as a monolithic command-line tool that accepted input in a single format and generated a static model evaluation report as the only output. Over time, it became clear that this one-size-fits-all approach was not ideal. Operational development of an automated scoring system requires collaboration between many stakeholders including NLP researchers, engineers, psychometricians and business units (Madnani and Cahill, 2018). The interdisciplinary nature of this community led to the emergence of several distinct *RSMTool* user groups. Each of these groups had separate requirements in terms of entry points, inputs, outputs, and documentation. Only by addressing all of these diverse requirements were we able to achieve wider adoption of *RSMTool* for model evaluation (Rupp et al., 2019; Yoon and Lee, 2019; Kwong et al., 2020).

The lessons we share are the salient ones we have learnt along the way: that different users have different needs and that going the extra mile on robustness – tests, documentation, and packaging – is essential to satisfy these needs. We believe that many of the points we discuss will be applicable to a range of NLP tools and, thus, could benefit the wider NLP OSS community.

3 RSMTool

3.1 Motivation

A single evaluation metric such as Pearson's correlation coefficient or Quadratically-weighted Kappa represents only one aspect of system performance. An automated scoring system deployed in a high-stakes application can have a significant impact on people's lives and, therefore, requires a comprehensive evaluation to ensure its accuracy, validity and fairness (Ramineni and Williamson, 2013). The goal of *RSMTool* is to encourage comprehensive reporting of model performance and to make it easier for stakeholders to compare different models along all necessary dimensions before model deployment. This includes not only standard agreement metrics, but also metrics developed within the educational measurement community and not commonly found in existing Python packages, such as measures of system performance based on test theory (Haberman, 2008; Loukina et al., 2020) as well as measures to evaluate fairness of system scores (Williamson et al., 2012; Madnani et al., 2017; Loukina et al., 2019). In this respect, our approach is similar in spirit to "Model cards" proposed by Mitchell et al. (2019) or standardized data statements advocated by Bender and Friedman (2018).

3.2 Architecture

RSMTool combines multiple analyses that are commonly conducted when building and evaluating automated scoring engines in a single package. In a typical use case, a user provides a file or a data frame with numeric system scores, gold-standard (human) scores, and metadata, if applicable. The tool processes the data and generates an HTML report containing a comprehensive evaluation including descriptive statistics on the input data and multiple measures of system performance and fairness among others[4]. *RSMTool* is written entirely in Python and makes heavy use of common Python libraries such as `pandas` (McKinney, 2010) and `scikit-learn` (Pedregosa et al., 2011). Each section of the report is implemented as a separate Jupyter notebook (Kluyver et al., 2016). The user can choose which sections should be included in the final HTML report and in which order.

RSMTool is available on Github with an Apache 2.0 license and has extensive online documentation[5]. It also includes a well-documented API allowing advanced users to integrate various components of *RSMTool* into their own applications. For more details on how *RSMTool* works, see Madnani and Loukina (2016); Madnani et al. (2017).

4 Lesson 1: Users have Different Needs

Over the years, we have identified several groups of users for *RSMTool*. While the ultimate goal for each group is to conduct a comprehensive evaluation of automated scoring systems, their specific needs were very different and could not be addressed by a single one-size-fits-all approach. In what follows, we describe these users and their needs as well as how we addressed them.

[4]See a sample report at `https://bit.ly/fair-tool`.
[5]`https://rsmtool.readthedocs.io`

4.1 Power users: NLP researchers

NLP researchers working on automated scoring were our initial target group when developing *RSMTool*. This group of users uses *RSMTool* to evaluate how an NLP-driven change, e.g., a new scoring feature, affects various aspects of model performance, above and beyond prediction accuracy.

Entry points. We found that the users in this group feel constrained by an end-to-end pipeline. Instead, they prefer to *pick and choose* the *RSMTool* functionality to plug into *their own pipeline* for model building and evaluation while also *using other Python packages*. To achieve this, we created a comprehensive API to expose various pre- and post-processing functions as well as custom metrics contained in *RSMTool*.

Inputs and outputs. We designed the various API endpoints to accept and return standard data types such as `pandas` dataframes and `numpy` arrays. Whenever possible, we used naming and signature conventions similar to other commonly used packages such as `scikit-learn` and `SKLL`, our other open-source package for running batched machine learning experiments[6].

Documentation. Since these users rely mainly on the API, they expect standardized Python API documentation. To this end, all public functions, methods and classes in *RSMTool* code contain PEP257-compliant docstrings[7].

4.2 Minimalists: Data Analysts & Engineers

Operational scoring systems are routinely monitored by running data through the system at regular intervals to ensure that operational metrics continue to be met. The data analysts & engineers responsible for this effort may lack the statistical or programming background to interact with the API directly and generally expect an *out-of-the box pipeline*.

Entry points. A key requirement for this group is a simple way to run the evaluation pipeline in *batch mode*. To address this, we have created command-line tools as well as Python API endpoints that can run the entire evaluation pipeline that a user can easily call in wrapper shell scripts, for example.

Inputs and outputs. This group of users often need to run evaluations on data that may not have all of the information necessary for some of the evaluation notebooks. To accommodate this, we designed the command-line tools and API endpoints to accept custom configuration parameters via `JSON` files or Python dictionaries, respectively. Furthermore, we also produce the outputs of each individual evaluation as `CSV/TSV/XLSX` for further use in monitoring workflows.

Documentation. Minimalists need access to enough information to get started with the tool. Therefore, in addition to API doctrings, we also created plaintext documentation comprising installation instructions, how to run the full pipeline along with the available configuration options, and a detailed tutorial with a real-life example. Nonetheless, we found that such users are often reluctant to read through the (admittedly large) list of configuration options. Therefore, we built in an interactive configuration generator with autocompletion[8] that can help such users create configuration files based on their specific needs.

4.3 Decision Makers: Managers & Business Units

The final decision about the architecture of the scoring system and its deployment is made by multiple stakeholders: business units, psychometricians, assessment specialists, and senior NLP researchers not involved in hands-on development.

Inputs and outputs. The users in this group require a self-contained, concise, clear, and readable evaluation report that can be reviewed, shared or used to create a slide deck or a memo. This group remains the primary user of our HTML reports.

Documentation. A standard request from this group has been to provide a document explaining various aspects of *RSMTool* functionality, structured as a general-purpose memo rather than technical documentation. Therefore, in addition to the user manual, the *RSMTool* documentation contains a general overview of its functionality as well as the formulae used to compute all evaluation metrics.

5 Lesson 2: Go the Extra Mile

Based on our experience developing and maintaining RSMTool, we claim that one way to properly address the different needs of the different types of users is by going the extra mile to write *robust* software. We define robust software as follows: the impact of any code change on its accuracy and performance can be measured (**well-**

[6]https://skll.readthedocs.io

[7]https://www.python.org/dev/peps/pep-0257/

[8]https://rsmtool.readthedocs.io/en/stable/automated_configuration.html#interactive-generation

tested), its documentation is always up-to-date (**well-documented**), and it (along with its dependencies) can be **easily installed** by the users. Of course, many of us in the NLP community are not trained as software engineers and do not have much experience with these practices. We argue that it is necessary to put in the extra work to learn them in order to make a meaningful contribution to the community. Fortunately, there are several open-source and/or free-to-use tools and services that can help make adopting these practices relatively painless.

5.1 Testing & Continuous Integration

A critical component of any software is a comprehensive test suite that covers a large percentage of the overall codebase. Open-source software should be no different. As developers of the software, it is important to ensure that the code does what we (or our users) think it does. Test-driven development (Beck, 2003) is now a common development practice that is well-supported by most programming languages and frameworks. Specifically for Python, there are several open-source packages that make it easy to write tests (`unittest`; `nose`; `pytest`), generate test coverage reports (`coverage`), and reduce code duplication across tests (`parameterized`).

We also strongly recommend the use of continuous integration services like Travis CI[9] and Azure Pipelines[10]. These services are free for open-source projects and can be easily integrated with GitHub such that the entire test suite is automatically run on all major platforms (Linux, macOS, and Windows) whenever a change is proposed to the code, with the proposer of the change not allowed to merge it unless all the tests pass and there is no decrease in test coverage. This reduces the likelihood that a new change is untested or that it introduces a regression in existing functionality.

5.2 Documentation

A recent survey of open-source software users[11] reported that over 90% of them cited incomplete or outdated documentation as the most pervasive problem they encounter in open-source projects. Our experience with RSMTool echoes this: documentation is our most important resource since it helps orient users in how to navigate the project.

[9] https://travis-ci.com

[10] https://azure.com/pipelines/

[11] https://opensourcesurvey.org/2017/#insights

We have already discussed that documentation designed to accommodate different users should include the motivation for the project, installation instructions, tutorials, and a detailed user manual. 21% of respondents in the Cai and Guo (2019) survey cited lack of tutorials and examples as a significant hurdle to the adoption of machine learning packages. The respondents also noted that for users lacking conceptual understanding, a simple "hello world" tutorial makes it hard to progress beyond the initial installation. We recommend including multiple tutorials in the documentation; tutorials that not only allow the users to test that their installation works, but also walk them through using the software to solve an actual problem. Tutorials should explain the reasoning behind each step and include download links for any necessary files.

Additionally, the documentation *must* also make it easy for interested users to contribute to the project by including (a) instructions for setting up a development environment, (b) best practices for writing tests, and (c) adding to the documentation itself. Finally, we also recommend formalizing a release process consisting of specific actions that must be taken to produce a new release and including it as a part of the documentation. This makes it easy for any developer to create a new release and makes the process transparent to the users.

The documentation files must be included in version control as part of the main codebase under a separate sub-directory. To ensure that the documentation stays up-to-date, any new functionality proposed for the code *must* include a documentation component that is reviewed for accuracy as well as readability as part of the code review. Specifically for Python, we recommend using the reStructuredText format for documentation along with Sphinx[12] to build the documentation. Sphinx provides many useful functionalities such as automatic rendering of LaTeX mathematical formulae via MathJax, automatic API documentation from Python function and class docstrings, and generating documentation in multiple formats such as HTML, PDF, and ePub. Freely available services like ReadTheDocs[13] can be integrated with GitHub-based development workflows such that the documentation is automatically built and deployed to a public-facing server for specified branches.

Finally, a different but equally important part

[12] https://sphinx-doc.org

[13] readthedocs.org

of the documentation is the project *changelog*. Each release must be tagged in the git repository and be accompanied by a detailed changelog that clearly describes the different types of changes contained in the release: new features, bugfixes, and backwards-incompatible changes, if any. Each change in the log should ideally be linked to the corresponding issue and pull request on GitHub providing the full context and discussion for the change to the interested user.

5.3 Packages & Dependencies

Another important aspect of open-source software is its ease of installation. Using it should not require an end user to figure out how to compile and install complicated dependencies. A better solution is a self-contained package installable with a standard package management tool that can also automatically install any required dependencies. For Python, we can use either source or wheel (binary) packages that are released to the Python Package Index (PyPI) and can be installed using `pip`. An alternative is to build `conda` packages[14] that can be released either via the default channel[15] or via a community-managed channel[16]. New packages should be built as part of every release and deployed to the appropriate package registry: this can be done either manually or automatically using GitHub actions[17].

Most scientific open-source Python software builds on top of other packages such as `numpy`, `pandas`, `scikit-learn`, `matplotlib`, among others. An important part of building packages is to properly version such dependencies in the package manifest so that the code behaves exactly as expected when installed. The most effective way to achieve this is to *pin* every single dependency to the exact version used during development. However, such a conservative approach is likely to cause conflicts since many open-source packages frequently release minor versions. We recommend leaving most dependencies unpinned except for those where a specific or a minimum version of a dependency is required. This may cause the code to break if an unpinned package releases an incompatible update. To deal with this possibility, we recommend setting up a weekly *scheduled build* in Travis CI that will create a new test environment – pulling in the latest versions of *all* dependencies – and run the tests in the main branch. A hotfix release can quickly be made if said weekly build starts failing. This approach works best if your test suite has high code coverage.

[14]https://conda.io
[15]https://anaconda.org
[16]https://conda-forge.org
[17]https://github.com/features/actions

6 Summary

In this paper, we shared the lessons we have learned as developers and maintainers of open-source software: different groups of users tend to have different needs and to meet these needs without compromising on quality, we must spend extra time and effort on testing, documentation and packaging.

References

Kent Beck. 2003. *Test-Driven Development: By Example*. Addison-Wesley Professional.

Emily M. Bender and Batya Friedman. 2018. Data Statements for Natural Language Processing: Toward Mitigating System Bias and Enabling Better Science. *Transactions of the Association for Computational Linguistics*, 6:587–604.

Carrie J. Cai and Philip J. Guo. 2019. Software Developers Learning Machine Learning: Motivations, Hurdles, and Desires. *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, 2019-Octob:25–34.

Wendy W. Chapman, Prakash M. Nadkarni, Lynette Hirschman, Leonard W. D'Avolio, Guergana K. Savova, and Ozlem Uzuner. 2011. Overcoming barriers to NLP for clinical text: The role of shared tasks and the need for additional creative solutions. *Journal of the American Medical Informatics Association*, 18(5):540–543.

Lei Chen, Klaus Zechner, Su-youn Yoon, Keelan Evanini, Xinhao Wang, Anastassia Loukina, Jidong Tao, Lawrence Davis, Chong Min Lee, Min Ma, Robert Mundkowsky, Chi Lu, Chee Wee Leong, and Binod Gyawali. 2018. SpeechRater 5.0. *ETS Research Report Series*.

Alina von Davier. 2016. Fairness Concerns in Computational Psychometrics. Presented at the panel on Fairness and Machine Learning for Educational Practice, Annual Meeting of the National Council on Measurement in Education, Washington DC.

Shelby J. Haberman. 2008. When can subscores have value? *Journal of Educational and Behavioral Statistics*, 33:204–229.

Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián

Avila, Safia Abdalla, Carol Willing, and Jupyter Development Team. 2016. Jupyter Notebooks — A Publishing Format for Reproducible Computational Workflows. In *Proceedings of the 20th International Conference on Electronic Publishing*. IOS Press.

A. Kwong, J. H. Muzamal, P. Y. Zhang, and G. Lin. 2020. Automated chinese language proficiency scoring by utilizing siamese convolutional neural network and fusion based approach. In *2020 International Conference on Engineering and Emerging Technologies (ICEET)*, pages 1–6.

Anastassia Loukina, Nitin Madnani, Aoife Cahill, Lili Yao, Matthew S Johnson, Brian Riordan, and Daniel F McCaffrey. 2020. Using PRMSE to evaluate automated scoring systems in the presence of label noise. In *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 18–29, Seattle, WA, USA.

Anastassia Loukina, Nitin Madnani, and Klaus Zechner. 2019. The many dimensions of algorithmic fairness in educational applications. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 1–10, Florence, Italy.

Nitin Madnani and Aoife Cahill. 2018. Automated scoring: Beyond natural language processing. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1099–1109, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Nitin Madnani and Anastassia Loukina. 2016. Rsmtool: collection of tools building and evaluating automated scoring models. *Journal of Open Source Software*, 1(3):33.

Nitin Madnani, Anastassia Loukina, Alina von Davier, Jill Burstein, and Aoife Cahill. 2017. Building better open-source tools to support fairness in automated scoring. In *Proceedings of the First ACL Workshop on Ethics in Natural Language Processing*, pages 41–52, Valencia, Spain. Association for Computational Linguistics.

Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, pages 56 – 61.

Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model cards for model reporting. *FAT* 2019 - Proceedings of the 2019 Conference on Fairness, Accountability, and Transparency*, (Figure 2):220–229.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Chaitanya Ramineni and David M. Williamson. 2013. Automated Essay Scoring: Psychometric Guidelines and Practices. *Assessing Writing*, 18(1):25–39.

Andrea Romei and Salvatore Ruggieri. 2013. Discrimination Data Analysis: A Multi-disciplinary Bibliography. In Bart Custers, Toon Calders, Bart Schermer, and Tal Zarsky, editors, *Discrimination and Privacy in the Information Society: Data Mining and Profiling in Large Databases*, pages 109–135. Springer Berlin Heidelberg.

André A. Rupp, Jodi M. Casabianca, Maleika Krüger, Stefan Keller, and Olaf Köller. 2019. Automated essay scoring at scale: A case study in switzerland and germany. *ETS Research Report Series*, 2019(1):1–23.

David M. Williamson, Xiaoming Xi, and F. Jay Breyer. 2012. A Framework for Evaluation and Use of Automated Scoring. *Educational Measurement: Issues and Practice*, 31(1):2–13.

Xiaoming Xi. 2010. How do we go about Investigating Test Fairness? *Language Testing*, 27(2):147–170.

Su-Youn Yoon and Chong Min Lee. 2019. Content modeling for automated oral proficiency scoring system. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 394–401, Florence, Italy. Association for Computational Linguistics.

Klaus Zechner, Derrick Higgins, Xiaoming Xi, and David M. Williamson. 2009. Automatic Scoring of Non-native Spontaneous Speech in Tests of Spoken English. *Speech Communication*, 51(10):883–895.

Michael J. Zieky. 2016. Fairness in Test Design and Development. In Neil J. Dorans and Linda L. Cook, editors, *Fairness in Educational Assessment and Measurement*, pages 9–32. Routledge.