Online Appendix of "Aligning XAI Explanations with Developers' Expectations"

ZIJIE HUANG, HUIQUN YU, GUISHENG FAN, ZHIQING SHAO, ZIYI ZHOU, and MINGCHEN LI, East China University of Science and Technology, China

1 MANUAL ASSESSMENT

1.1 Assessment Process

Please check "keywords.xlsx". The first sheet called "`Open_Coding`" contains the preliminary results of keyword selection and open coding. The second sheet called "`Axial_Coding`" contains the in-process results of axial coding. The keywords in the prior sheets are ordered in descendent by their frequency of occurrence. The last sheet called "`Taxonomy`" is the final extended version of Table 2 of the submitted paper.

1.2 Assessment Results

Please check the csv file in `./performance/rq3/` named `modified.csv`. The file also contains the *coverage* and *entropy* results of the selected K in RQ3 (K values in {11,10,5,3} for {*Shotgun Surgery*, *Blob*, *Complex Class*, and *Spaghetti Code*}, respectively). The columns titled {coupling, changehistory, cohesion, size, complexity, production, testing} are annotated results. The columns started with selected_ are covered aspects of XAI explanations, which will be used to compute *coverage*.

2 CODE REPLICATION

Please beware that most results in the project are **cached** (except for the figures) to save the time of reviewers and readers from tuning hyperparameters and generating in-process results. You may delete the cached files to regenerate results (if you wish to perform a fresh replication from the very beginning). If so, please follow strictly the order of execution described below, otherwise, the scripts won't function.

2.1 Project Structure

The `./data` directory contains raw dataset or annotated comments for prediction. The `./pk` directory contains temporary results for prediction (to save time from recalculation). The `./performance` directory contains performance data used to generate figures and tables. The `./results` directory contains detailed experimental results. The `.py` files in root path are Python scripts used for the experiment.

2.2 Generating Data and Figures for RQ1

First, `data_rq1.py` should be executed for generating the prediction result without removing project metrics. Then, to generate Fig. 3 in the submitted paper, we need to pick the best-performed hyperparameter and execute `figure_rq1.py` (by calling `RQ1_fig()`, a function with 3 parameters), in which the 1st parameter is the name of the smell, the 2nd one is whether project metrics should be removed, and the 3rd one is the hyperparameter. **The results could be found in**

Authors' address: Zijie Huang, hzj@mail.ecust.edu.cn; Huiqun Yu, yhq@ecust.edu.cm; Guisheng Fan, gsfan@ecust.edu.cn; Zhiqing Shao, zshao@ecust.edu.cn; Ziyi Zhou, zhouziyi@mail.ecust.edu.cn; Mingchen Li, lmc@mail.ecust.edu.cn, East China University of Science and Technology, 130 Meilong Road, Shanghai, China, 200237.

`./results/figs/rq1_{smell_name}_{True,False}.pdf`. To generate the results of the randomly generated values of project metrics, `data_fake_rq1.py` should be executed, and **the XAI results of the faked project metrics (this figure is not presented in the paper) is available in `./results/figs/rq1_{smell_name}_fake.pdf`, demonstrated in Fig.1 in this appendix**, which is very similar to the ones (using real values of project metrics) in the paper. The generation of results in Table 6 (performance after project metrics are removed and AutoSpearman is applied) will be generated in the next subsection.

2.3 Generating Data and Figures for RQ2

2.3.1 RQ2 Data Generation. First, `feature_selection.py` should be executed to generate feature selection results for RQ2 and RQ3. Then, `prediction_rq2_rq3.py` should be executed for generating prediction performance based on AS_Additive, AS_Reductive, and the original AutoSpearman. For RQ2, we focus specifically on the original AutoSpearman. Thus, we pick the best performance (available in `./results/settings/rq2.csv`) and execute `figure_rq2_data_rq3.py`, which is capable of generating XAI explanations, calculating their *coverage* and *entropy*, and drawing the figures of RQ2. Since the *coverage* and *entropy* measurement logic is shared between RQ2 and RQ3, the data of RQ3 are also generated. The raw data of *coverage* and *entropy* under K ranged from 2 to 15 are present in csv files in `./results/explanations/` named {`additive, reductive, orig`}_K.csv. {`additive, reductive, orig`} refers to AS_Additive, AS_Reductive, and the original AutoSpearman, respectively.

2.3.2 Performance Results and Figures. For performance (in Table 6), please refer to `./results/settings/rq2.csv`. For figures (Fig.4), please refer to `./results/figs/rq2.pdf` and `./results/figs/rq2_entropy.pdf`.

2.4 Generating Data and Figures for RQ3

2.4.1 RQ3 Data and Figure Generation. Most data of RQ3 are generated while RQ2 data are generated since they share similar logic of generation. However, the data in RQ3 are demonstrated in a more detailed fashion. Thus, we should execute `figure_rq3.py`.

2.4.2 Performance Results and Figures. For performance (in Table 7), please refer to `./results/settings/additive.csv` and `./results/settings/reductive.csv`. For the 8 subsets in Fig.5-6, please refer to `./results/figs/rq3*.pdf` (* is a wildcard). Table 8-13 are generated based on `./results/compare_rq3.csv`. Table 8 and Table 9 are generated by comparing values between performance values in the filtered rows with specific `head` column value (“all" for Table 8, and “comparison, coupling, changehistory, cohesion, size, complexity, etc." for Table 9 and 14). This process is not performed programmatically. Similarly, for Table 10-13, they are generated by comparing the “features" row of this file.

2.5 Generating Data and Figures for Other Sections

2.5.1 Generating Explanations for Specific Classes (Fig.7-9). execute `./figure_discussion.py`, and the `extract()` function could do the job. The 1st parameter is the name of the smell, the 2nd parameter is the name of the class, and the 3rd parameter is a boolean value indicating whether the predicted explanation (could be contrastive if the prediction is wrong) should be generated.

2.5.2 Generating Correlations between the Feature and the Prediction Targets. execute `./figure_discussion.py`, and the `correlation()` function could do the job. The output would be “`number-fixes spaghetti-code p:7.829217380649043e-63 rho:0.7707118302566077`".

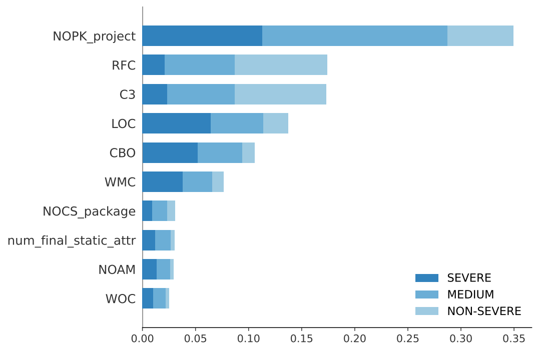
2.5.3 Generating Agreement Values between the SHAP and Information Gain. The data required should already be generated by the `explain_selected()` function in `./figure_rq2_data_rq3.py`. The data are located in `agreement_gain_shap_{additive,reductive,orig}.csv`. We use AS_Additive for *Spaghetti Code* and AS_Reductive for other smells, which is the same as our settings in RQ3.

2.6 Generating RuleMatrix Result

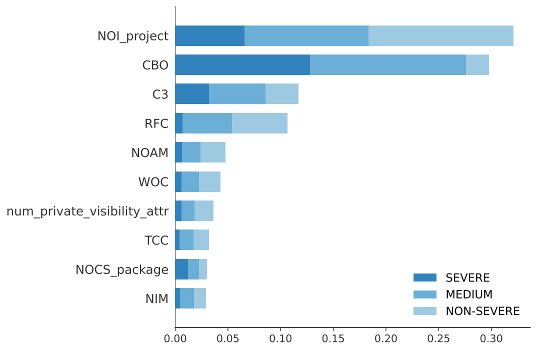
RuleMatrix requires Linux environment and some fix of dependency issues. Thus we are not providing replication. The generated rules are in `./results/rulematrix.txt`, and the figures are in Fig. 2 of this appendix.

3 ADDITIONAL RESULTS

3.1 Feature Importance of Models Built Using Faked Project Metrics



(a) *Blob*



(b) *Complex Class*

Fig. 1. Top-10 feature importance with project metric faked.

3.2 Result of RuleMatrix Application in The Discussion Section

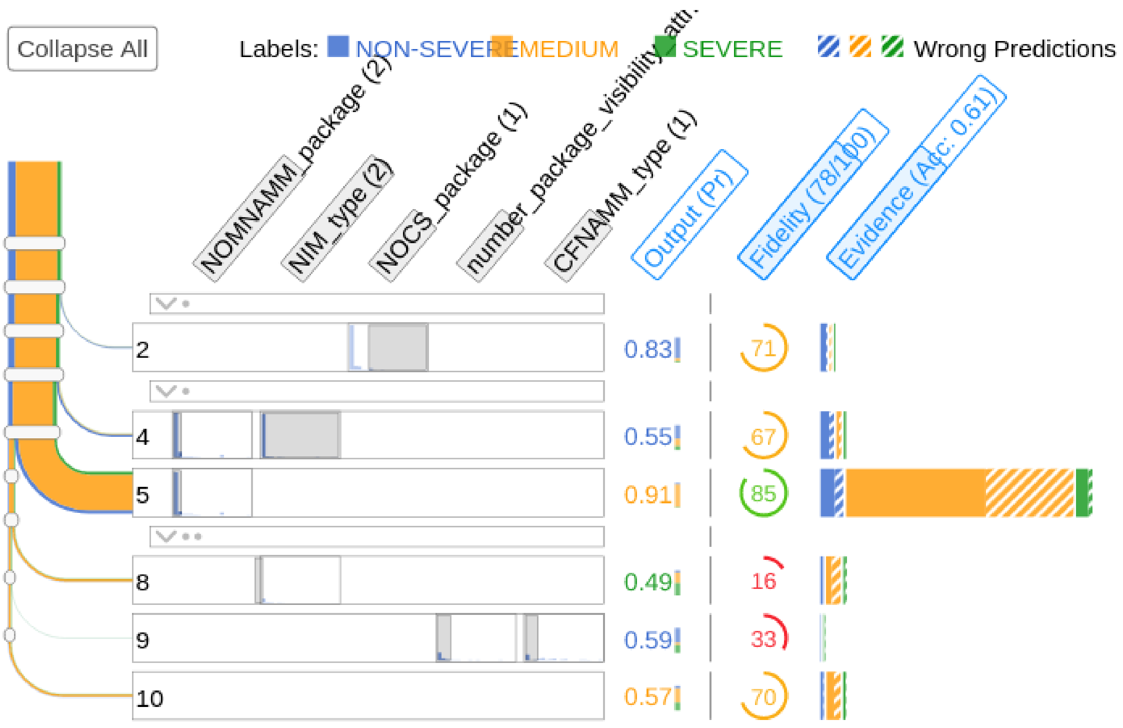


Fig. 2. RuleMatrix Execution Result for *Spaghetti Code*

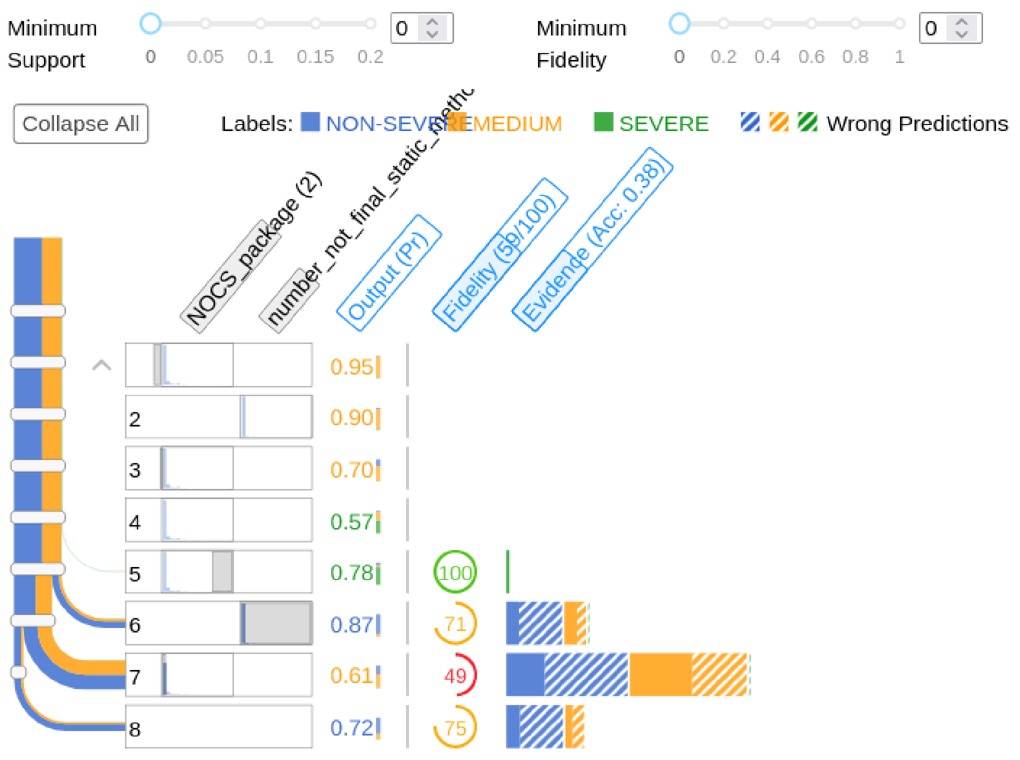


Fig. 3. RuleMatrix Execution Result for *Complex Class*