Intro Programming Grading Assistant Using Static Code Analysis

Kwabena Aboagye-Otchere, Om Parkash, Kavya Marthineni, Gabriel Owusu

The University of Texas Rio Grande Valley , Edinburg - Texas

Introduction

Introductory programming classes have become overpopulated due to technological requirements across several academic and professional fields. We present a static code analysis tool designed to support introductory programming teachers by analysing students' source code for submitted assignments at statement level and highlighting common problems identified across submissions.

Literature Review

- Students believe programming assignments help them understand programming concepts better.
- Introductory programming classes focus on teaching both program syntax and basic problem solving using programming.
- Instructors provide generalised feedback based on common errors they presume students will have through means such as solution manuals.

Methodology

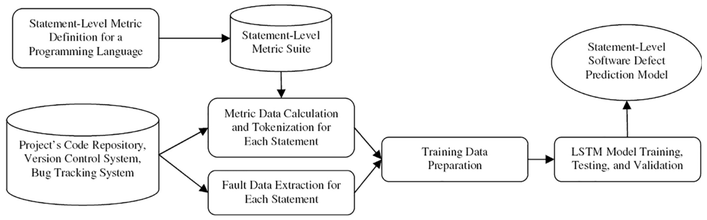

Figure 1. Overview of System Design

ID	Metric	Description
11	Literal String	The number of string literals in a line
12	Integer Literal	The number of integer literals in a line
13	Literal Count	The total number of literals in a line
14	Variable Count	The number of variables in a line
15	IF Statement	The number of `IF` conditions in a line
16	FOR Statement	The number of `FOR` loops in a line
17	WHILE Statement	The number of `WHILE` loops in a line
18	DO Statement	The number of `DO WHILE` loops in a line
19	SWITCH Statement	The number of `SWITCH` in a line
20	Conditional and Loop Count	The number of loops and conditionals in a line
21	Variable Declaration	The number of declared variables in a line
22	Function Declaration Count	The number of declared functions in a line
23	Variable Declaration Statement	The number of statements in which a variable is declared in a lin
24	Declaration Count	The number of declaration statements in a line
25	Pointer Count	The number of pointers in a line
26	User-Defined Function Count	The number of non-library functions called in a line
27	Function Call Count	The number of called functions in a line
28	Binary Operator	The number of binary operators used in a line
29	Unary Operator	The number of unary operators used in a line
30	Compound Assignment Count	The number of compound assignments in a line
31	Operator Count	The total number of operators in a line
32	Array Usage	The number of arrays used in a line

Figure 2. Internal-linear Metrics

ID	Metric	Description
1	Function	Is the line located in a function
2	Recursive Function	Is the line located in a recursive function
3	Blocks Count	The number of nested blocks in which the line is located
4	Recursive Blocks Count	The number of nested recursive blocks in which the line is located
5	FOR Block	The number of nested `FOR` blocks in which the line is located
6	DO Block	The number of nested `DO WHILE` blocks in which the line is located
7	WHILE Block	The number of nested `WHILE` blocks in which the line is located
8	IF Block	The number of nested `IF` blocks in which the line is located
9	SWITCH Block	The number of nested `SWITCH` blocks in which the line is located
10	Conditional Count	The number of single conditions checked to reach a line. This includes the number of components in a compound condition as well as nested conditionals

Figure 3. External-linear Metrics

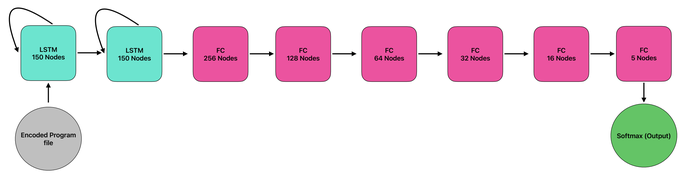

Figure 4. Model Architecture

- We collect C/C++ programs from the Code4Bench dataset which contains 200K programs entailing 2.36M lines of code.
- We encode each line of code using 32 metrics (10 external-linear, 22-internal linear). External-linear metrics provide context to each line of code, and internal-linear metrics represent static features of the syntax on that line specifically.
- We build a classifier that determines if a line of code is error prone, and what kind of error it is.
- We aggregate predictions over programs assessed and provide descriptive statistics showing the frequency of errors encountered across submissions.

Experimental Setup

- We evaluate our algorithm using an accuracy neighborhood (AN)
- We perform 10-fold cross validation across the dataset

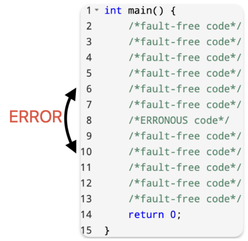

Figure 5. Sample Program Evaluation with AN=2

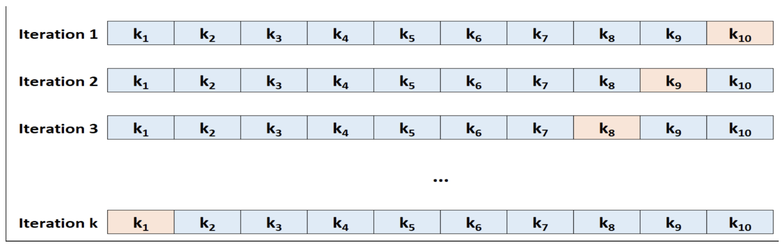

Figure 6. 10-fold cross validation

Results

Accuracy Neighborhood	Accuracy	Precision	Recall	F1-score
4	0.9523	0.9529	0.9522	0.9524
2	0.9486	0.9479	0.9486	0.9479

Table 1. Model average performance across folds

- Using an AN of 4, we achieve an average accuracy of **0.9523** across the 10 folds.
- Using an AN of 2, we achieve an average accuracy of **0.9486** across the 10 folds.

Conclusion

- Deep learning is capable of assisting instructors provide feedback for programming assignments through static code analysis.

Contributions

- Proposed a static code analysis tool that identifies common errors in computer programs capable of analysing code that even has syntax errors.
- Aid feedback provision in intro programming classes with descriptive statistics of common errors encountered across submissions.

Future Works

- Define architecture that allows enrollment of new error classes.
- Explore automated feedback provision by leveraging Large Language Models.

References

- A. Luxton-Reilly et al., 'Introductory programming: a systematic literature review', in Proceedings companion of the 23rd annual ACM conference on innovation and technology in computer science education, 2018, pp. 55–106.
- A. Majd, M. Vahidi-Asl, A. Khalilian, P. Poorsarvi-Tehrani, and H. Haghighi, 'SLDeep: Statement-level software defect prediction using deep-learning model on static code features', Expert Systems with Applications, vol. 147, p. 113156, 2020.
- A. Majd, M. Vahidi-Asl, A. Khalilian, A. Baraani-Dastjerdi, and B. Zamani, 'Code4Bench: A multidimensional benchmark of Codeforces data for different program analysis techniques', Journal of Computer Languages, vol. 53, pp. 38–52, 2019.
- S. Schleimer, D. S. Wilkerson, and A. Aiken, 'Winnowing: local algorithms for document fingerprinting', in Proceedings of the 2003 ACM SIGMOD international conference on Management of data, 2003, pp. 76–85.

kwabena.aboagyeotchere01@utrgv.edu