Real-time programming exercise feedback in MOOCs

Zhenghao Chen, Andy Nguyen, Amory Schlender, Jiquan Ngiam
Coursera
381 East Evelyn Ave
Mountain View, CA, USA
{zhenghao, anguyen, aschlender, jngiam}@coursera.org

ABSTRACT

We present an active learning system for coding exercises in Massively Open Online Courses (MOOCs) based on real-time feedback. Our system enables efficient collection of personalized feedback via an instructor tool for automated discovery and classification of bugs.

1. INTRODUCTION

Active learning is a learning approach that "requires students to do meaningful learning activities" in contrast to traditional lecture-based approaches where "students passively receive information from the instructor" [2]. In active learning, timely feedback is important as it helps learning and reduces the risk of learner disengagement due to repeated failure to complete learning activities.

MOOCs have leveraged in-videos quizzes as an active learning strategy, but these quizzes have traditionally been limited to multiple choice questions. One reason that introducing higher order tasks, such as coding exercises, has been challenging is that it is difficult to provide good feedback. Most automated code grading systems allow for efficient grading through unit testing, but these methods are often limited in the forms of feedback they can provide.

Feedback that helps learners understand their errors can improve learning outcomes. Stamper et al. [5] demonstrated significant problem completion rate improvements in a logic course when feedback was available to learners. This has motivated related developments in data-driven methods to generate such feedback [3, 4, 1].

In this demo, we will show a system that enables instructors to efficiently generate and provide real-time feedback for programming exercises in MOOCs through extensions to Executable Code Blocks (ECBs) [6] and the Codewebs engine [1]; these exercises can be embedded throughout the learning experience to enable rich active learning.

2. EXECUTABLE CODE BLOCKS

Executable code blocks (ECBs) [6] enable learners to write and execute code directly in their web browser. The primary advantage of ECBs is that they can be tightly integrated into the course experience. For example, immediately after a concept is explained in a video, a learner can be asked to implement the specific concept in an ECB.

ECBs usually employ unit testing strategies to evaluate if a learner's implementation is correct. We extend ECBs such that when a learner makes an incorrect submission, they can request additional feedback that highlights potential errors in their submission and provides hints that guide the learner towards correcting these errors (see figure 1). These hints are provided efficiently by an instructor through an extension of the Codewebs engine.

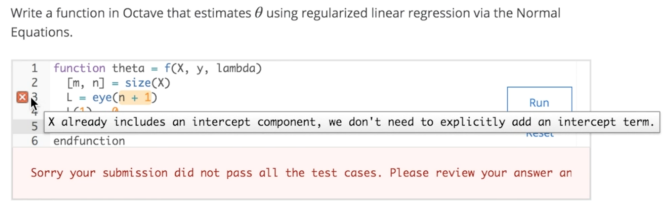

Figure 1: Hints provided in an ECB for an incorrect submission.

3. CODEWEBS ENGINE

We use the Codewebs engine [1] to localize errors in learner code submissions and identify common classes of errors. We describe here the relevant process of doing so automatically at a high level, and refer the reader to [1] for details.

The Codewebs engine operates on the abstract syntax tree (AST) representation of code submissions. Let n be a node in the AST, T_n be the subtree rooted at n, and P_n be the subtree rooted at the *parent of* n. The local context of T_n, denoted by T_n^c, is P_n with T_n removed (see figure 2).

We say that T_n^c is a buggy context if submissions containing T_n^c are more likely to be incorrect than by random chance. The Codewebs engine declares that P_n is a bug if T_n^c is a buggy context but no subtree of T_n has a buggy context. Given a bug P_n, the Codewebs engine then searches for a correction C such that replacing P_n with C results in a correct program.

We extend Codewebs in two ways. First, we modify the localization process to consider local contexts that are semantically equivalent[1]. This allows us to discover more bugs across submissions that might have *syntactically* distinct but *semantically* equivalent contexts. We also use this to improve correction discovery in a similar way (see figure 3) and improve correction searching to handle instances where multiple bugs occur within a submission.

Second, we introduce the concept of bug groups or error modes. Two bugs B and B' belong to the same group *iff* B

[1]We follow the definition of semantic equivalence used in [1].

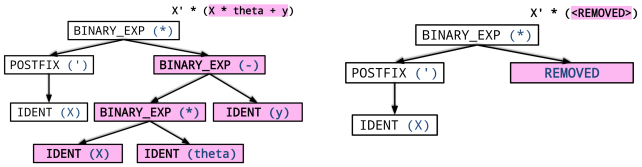

Figure 2: Left: Subtree P_n containing subtree T_n in pink. Right: T_n^c, the local context of subtree T_n.

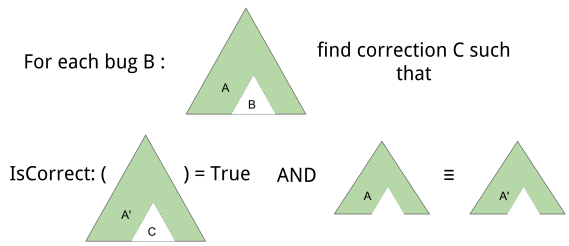

Figure 3: Visual illustration of finding corrections for bug B, C is a correction for B if we can find a correct submission where C is surrounded by A' and A' is semantically equivalent to A.

is semantically equivalent to B' and the correction for B is semantically equivalent to the correction for B'.

4. INSTRUCTOR ANNOTATIONS

By grouping bugs together, instructors can provide a hint for each error mode (instead of for individual submissions). These hints power the feedback features mentioned in section 2 (see figure 1).[2]

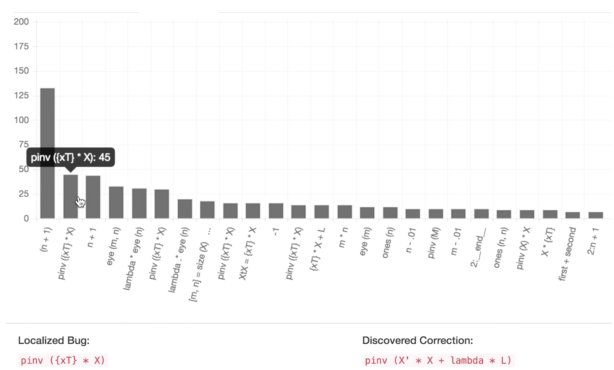

Figure 4: Instructor tool for exploring common errors based on bug equivalences classes.

Furthermore, we can provide instructors with a tool (see figure 4) to explore these common error modes. This tool orders bug groups by the frequency at which they appear in learner submissions. This enables instructors to quickly understand the most common errors made by learners. This breakdown is useful for course material improvement as they can expose common learner misconceptions.

5. RESULTS

We introduced 3 ECBs into the Machine Learning MOOC on Coursera involving tasks of varying levels of complexity (e.g., implementing the cost function for regularized linear regression). Each ECB required between 10 and 20 lines of code each to solve.

For each ECB we collected between $3,118$ and $5,550$ submissions, consisting of between around $1,000$ and $3,000$ distinct ASTs (see table 1). These submissions were used to train the Codewebs model. We find that a relatively small number of error groups (40) is required to achieve good coverage

[2]It is also possible to show learners automatically generated corrections when instructor input is not available.

	Submissions	% Correct	Unique ASTs	% Coverage (40 bug groups)
RLR Normal Eqn	$3,118$	52.8%	$1,338$	61.0%
Matrix Inv Cost Fn	$3,892$	19.5%	$1,440$	49.5%
Matrix Inv Grad	$5,550$	11.5%	$3,050$	36.7%

Table 1: 3 ECBs added to the Machine Learning MOOC on Coursera

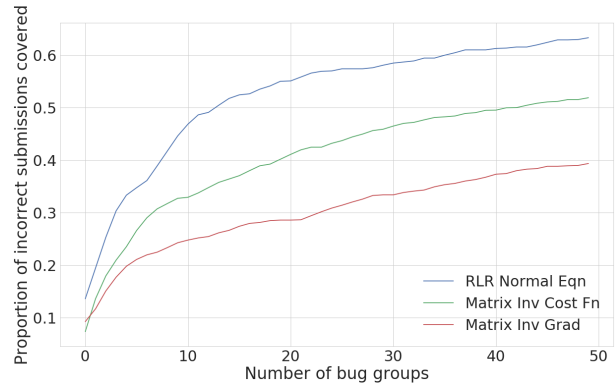

Figure 5: Percentage of incorrect submissions by number of error modes.

of a large fraction of incorrect submissions (see figure 5). Between 28.6% and 55.0% of incorrect submissions contain at least 1 of the 20 most common error modes, and between 36.7% and 61.0% contain at least 1 of the 40 most common error modes (see figure 5).

A teaching assistant was recruited to label the top 40 discovered error groups, and we are now running tests to understand the effects of this intervention on learning outcomes.

6. REFERENCES

[1] A. Nguyen, C. Piech, J. Huang, and L. Guibas. Codewebs: Scalable homework search for massive open online programming courses. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 491–502, New York, NY, USA, 2014. ACM.

[2] M. Prince. Does active learning work? a review of the research. *J. Engr. Education*, pages 223–231, 2004.

[3] K. Rivers and K. R. Koedinger. Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. *International Journal of Artificial Intelligence in Education*, 27(1):37–64, 2017.

[4] R. Singh, S. Gulwani, and A. Solar-Lezama. Automated feedback generation for introductory programming assignments. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 15–26, New York, NY, USA, 2013. ACM.

[5] J. C. Stamper, M. Eagle, T. Barnes, and M. Croy. *Experimental Evaluation of Automatic Hint Generation for a Logic Tutor*, pages 345–352. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[6] C. Wong. Active learning experiences with code executable blocks. `https://building.coursera.org/blog/2016/09/30/active-learning-experiences-with-code-executable-blocks/`.