Factors Impacting Novice Code Comprehension in a Tutor for Introductory Computer Science

LEIGH ANN SUDOL-DeLYSER, Carnegie Mellon University
JONATHAN STEINHART, Tutor Technologies

Code comprehension activities have been strongly correlated with measures of student performance in computer science classes. This poster presents a model predicting student success based on features collected during the tutoring session. Model selection indicated that experimental condition combined with within tutor cumulative percent correct yielded the best model.

1. INTRODUCTION TO DATA

To test the effect of questions regarding the low-level details of a program compared to questions regarding the more abstract nature of the algorithms, a tutoring environment was constructed. Prior work has shown a strong correlation between students' perception of the global goals of a program in a code comprehension task and their ability to write code to solve problems[1; 2; 3]. Using these tutoring data, we model the students' learning progression through the activities and compare student performance on common tutoring questions. The models were evaluated for predictive factors in student performance. Performance on common questions was dependent upon the factors of the design as well as the students' performance and how many questions they had completed. These results can help give us insight into the type of practice students need to better comprehend code and points to types of feedback to explore in future work.

In the fall of 2010, introductory computer science students from Carnegie Mellon University (N=236) participated in a study to determine the effects of interacting with a code comprehension exercise. Students engaged with the tutor as a part of a homework assignment in their introductory computer science course. Three different instructors teach the course, and students were drawn from all classes.

Students were randomly assigned to conditions pairing Abstraction vs Tracing with Context vs. No-Context. In the contextualized conditions, the examples that students saw included a one-sentence description of the context, and variables names were changed to match the context.

As a part of the tutor, students interacted with 30 code comprehension questions. The 30 questions referred to about 10 different code segments, implementing various algorithms (e.g., sum, search, sort, rotate). Each code segment had a series of three questions associated with it. The first question was a tracing question and was seen by all students, regardless of condition, and is referred to as the common question. The second and third question varied by condition. In the Tracing condition, students were asked to trace the code with two other data sets. In the Abstract condition, students were asked questions about the global goals of the algorithm, or the role of its components. Code examples were presented to all students in the same order, and the first 6 were selected to be easier than the last 4.

2. DATA MODEL CONSTRUCTION

The following factors are included in the models used in this paper. *Tracing*: (categorical), tracing condition? (yes/no) *Context*: (categorical), context present? (yes/no) *Problem Number*: (ordinal), problem number, to account for order effect *Problem ID*: (categorical), to account for specific problem characteristics *Cumulative Percent Correct*: (numerical), the percent correct so far within tutor *Student*: a random intercept to account for random variation in ability between students

We used logistic regression to estimate students' mean probability of success, given characteristics that would be known to a "live" Intelligent Tutoring System. Logistic regression models the log odds of success of the ith student, π_i, as some unknown linear combination of the student's attributes.

Logistic regression models the log odds of success of the ith student, π_i is

$$\log\left(\frac{\pi_i}{1-\pi_i}\right) = X_i\beta \quad \Leftrightarrow \quad \pi_i = \frac{1}{1+e^{-\mathbf{X_i}\beta}}$$

We performed model selection using an initial model fit (using `glm`) to the entire dataset and containing all of the aforementioned variables and all second-order interactions. We then performed stepwise selection (using `step`) to find a model which minimizes Akaike's Information Criterion (AIC). The resulting model kept all of the main effects except problem number, interactions between problem id and each of the other main effects, and the interaction between context and cumulative percent correct.

3. RESULTS

To validate our model, we performed five-fold cross-validation. We fit the aforementioned model to each training fold, which resulted in a predicted success probability for each training case. To convert predicted probabilities (continuous) into predicted success/failure (dichotomous), a threshold value was chosen to maximize Cohen's κ on the training data. The error rate and κ were then measured for each test fold. Then, the resulting five cross-validation error rates and κ values (i.e. one for each fold), were averaged for a mean CV error rate of 0.1294 and approximate κ of 0.4849. Our somewhat crude model seems to do a reasonable job of predicting success, although there is much room for improvement.

In addition to predicting success/failure, our fitted logistic regression model also offers some explanatory insight into the ways in which code base, context, tracing/abstraction, and student ability act and interact. Some observations from the fit include:

—Cumulative percent correct (a rough measure of student ability) generally increased the probability of success, and this boost was greater in the context conditions.

—Code base 2 was easier for those in the tracing conditions but – interestingly – harder overall for those with higher cumulative percent correct(p=0.0528)!

—Except for code base 2, though, tracing generally hurt performance for everyone, and particularly in code base 10.

—Context made code base 8 significantly easier, but otherwise it generally had no effect on success.

4. ACKNOWLEDGEMENTS

We would like to thank the PSLC, especially Brett Leber for help in constructing and deploying the tutor. This work was partially funded by the IES, US Department of Education, through Grant R305B040063 to Carnegie Mellon University. The opinions expressed are those of the authors and do not represent the views of the Institute or the US Department of Education.

REFERENCES

R. J. Barker and E. A. Unger. A predictor for success in an introductory programming class based upon abstract reasoning development. pages 154–158, 1983.

M. Lopez, J. Whalley, P. Robbins, and R. Lister. Relationships between reading, tracing and writing skills in introductory programming. In *ICER '08: Proceeding of the Fourth international Workshop on Computing Education Research*, pages 101–112, New York, NY, USA, 2008. ACM.

A. Venables, G. Tan, and R. Lister. A closer look at tracing, explaining and code writing skills in the novice programmer. In *ICER '09: Proceedings of the fifth international workshop on Computing education research workshop*, pages 117–128, New York, NY, USA, 2009. ACM.