Interpretable Code-Informed Learning Analytics for CS Education

Yang Shi
North Carolina State University
yshi26@ncsu.edu

ABSTRACT: The development of deep learning technologies brings opportunities to computer science education (CSEd). Recent CSEd datasets provide actual code submissions that could potentially offer more information on students' learning for learning analytics, but adding the information needs special considerations. My research focuses on leveraging these code submissions to inform learning analytics. Specifically, it addresses two problems: 1) finding and detecting a meaningful representation of knowledge components (or skills) in students' programming and 2) using them to provide formative feedback. Previous works have defined or discovered skills in CSEd, but these skills do not follow certain properties of learning. My research proposes to incorporate these properties into data-driven model design, and improves the knowledge components so that they are consistent with theoretical properties, and meanwhile also provide better interpretability than typical deep learning models. Following this, my second project will focus on providing personalized support in student learning.

Keywords: Computer Science Education, Educational Data Mining, Code Analysis, Interpretable Deep Learning

1 INTRODUCTION

Computer science (CS) education faces unprecedented opportunities and challenges when meeting the fast development of deep learning (LeCun et al., 2015) technologies these years. With these tools empowered by big data analysis, teachers and students are expected to enjoy better learning and teaching environments, while researchers will have more detailed, accurate, and efficient learning analytics. For example, tools based on deep learning can be used for tracing students' learning progress of knowledge (Piech et al., 2015), and thus students can get formative feedback (Shute, 2008) on problems that they are predicted to be wrong and reflect on the knowledge gaps in certain concepts. Teachers can also use data analysis to learn their students' progress, and inform their pedagogical decisions (Ju et al. 2019). However, using deep learning to analyze CS education data has additional challenges.

Many challenges reside in learning analytics (LA) or educational data mining (EDM) when using deep learning technologies, even without domain-specific data. For example, deep learning models are known as black boxes (Castelvecchi, 2016) despite their advantages in task performance. The uninterpretable models can harm the trustworthiness of the conclusions the model provides, especially in the educational domain (Cohausz, 2022). Recent research in CSEd has been collecting submitted code data along with the typical performance data (Leinonen, 2022), and these datasets add more opportunities to derive interpretability from code data. This adds more information to the models but needs careful design. Furthermore, even with an interpretable model, leveraging

interpretations and propagating intelligent feedback to students are still unaddressed challenges. My research focuses on providing solutions to these challenges. In the first project, I will design a deep neural network to leverage code information for knowledge component (KC) (Koedinger et al., 2012a) detection. The model learning process is guided by learning theory. While the model is designed as a performance predictor, trying to take historical submissions and performances and use them to predict future performance, the middle layer of the model can be interpreted as a representation of students' practiced skills. Learning theory is designed as a constraint on the middle layer, so that it follows properties such as the power law of practice (Cen et al., 2006). The middle layer can be tracked to actual code input, attributing the coding concepts students are capable of or incapable of. The second project addresses the usage of the detected KCs. For a student with detected knowledge gaps on certain concepts, code examples (Brandt et al., 2009) will be shown to students as formative feedback.

2 Related Work

Student Modeling in CS Education: There are many works around the topic of student modeling for CSEd. As more student code datasets are made accessible to the public, student modeling in CS education has seen fast development in recent years. It is a special domain, as programming code is rich in information, but also often bears much noise. Some recent works have been manually analyzing small to medium size student code datasets for student modeling. For example, Paul et al. have created instruments for manually detecting misconceptions from student code (Paul & Vahrenhold., 2013). Davies et al. manually analyzed programming trace data and constructed a library of knowledge gaps, showing that programming logs would reveal more misconceptions than single submission data (Davies et al., 2015). While expert analysis is more detailed and offers insights, they are also expensive, time-consuming, and often vulnerable to "expert's blind spots" (Nathan et al., 2003). Data-driven models, on the other hand, don't have these disadvantages, however, often suffer from low performance when evaluated on expert-assigned labels. They can achieve good performance with little help from experts. For example, Marwan et al. developed a subgoal detector in students' programming data and showed that adding a few expert constraints helped the detection outperform data-driven models (Marwan et al., 2021). However, this would still need explicit expert help. Deep learning models can be leveraged as an alternative way to improve performance, but they suffer from low available labeled datasets, and limited interpretability and thus caused low trustworthiness in the modeling results. My previous work used a semi-supervised learning method to address the limited label issue (Shi et al., 2021b), and my proposed work looks into how to add more interpretability to deep learning models, while they provide accurate modeling results.

KC Discovery and Refinement: KCs in my proposed work refer to programming skills that students learned and should be observable from the programming dataset in the CS domain. Traditionally, KCs are manually analyzed and defined by cognitive task analysis (CTA, Clark et al., 2008), but they are time-consuming and prone to experts' blind spots. As KCs are mapped to problems by Q-matrices (Barnes, 2005), many methods are developed to refine these Q-matrices or to discover new KCs (Cen et al., 2006; Koedinger et al. 2012b). These methods require less human effort, but the resulting Q-matrices can be also less interpretable. For example, one key method is to use learning curves for refinement. Cen et al. proposed that the error rate of students' practice on certain KCs should follow

the power law of practice (Cen et al., 2006). They calculated the score of fitting to evaluate the quality of KCs specified by a Q-matrix. Their method is further extended to an automatic search algorithm for Q-matrix refinement, using A* search (Koedinger et al. 2012b). However, these methods do not have domain information involved and thus cannot attribute errors to the exact skills in students' code. In the computing education domain, Rivers et al. (2016) proposed to use of abstract syntax tree (AST) nodes to represent KCs and found that some KCs do not fit the expected properties (e.g. the power law). My proposed method is inspired by but differs from theirs. It leverages programming code to inform the discovery process and does not require the initialization of the Q-matrix. Instead, it uses a randomly initialized matrix for modeling the KC-problem relationships, updated in the deep learning model training process.

Deep Code Learning: Deep learning has evolved to solve more complex problems in more disciplines recently, including program code analysis. There are many models designed to process code in recent years. Earlier models (e.g. GRU, Bi-LSTM, etc.) only treat code as a sequence of tokens (Reyes et al., 2016), leaving tree-structured information of code unused. Later models such as code2vec (Alon et al., 2019) and ASTNN (Zhang et al., 2019) take this structural information into account, and achieved higher performance in tasks such as function name classification and code summary. In an educational context, recent papers have discussed the usage of such models in student performance prediction (Mao et al., 2021; Shi et al., 2022), bug detection (Shi et al., 2021a; Shi et al. 2021b), code classification (Fein et al., 2021), etc. tasks. However, one key issue still remains in these applications, and that is the lack of interpretability of these models. This undermines the trust of models, as the prediction process of models cannot be explained to teachers or students. For example, in performance prediction, these models only make the predictions when students fail without telling why they will fail on certain problems. Moreover, as recent research suggests, data-centric methods are prone to biases that lead to inequity (Ocumpaugh et al. 2014), which further requires interpretability for the proper application of these models. My work proposes to use learning theory to guide the model training process, introducing a possible way to interpret the middle layer vectors as the learning progress of KCs and thus add more trustworthiness to the prediction outcomes produced by deep learning models.

Intelligent Feedback: There are many previous works using intelligent feedback in tutoring systems. Research shows that formative feedback helps students' learning, but it needs to be specific, on-time, corrective, and positive (Shute, 2008). Due to these specific properties of feedback, it is difficult to provide automated feedback to students and meet these requirements at the same time. Prior works have been only fulfilling partial properties. For instance, autograders can only provide corrective feedback, namely only giving feedback about which test cases the students failed or passed. More importantly, it does not provide feedback on which exact KC the student is not able to achieve. My research entails a solution to provide feedback not only on the problem correctness, and the corresponding test cases, and also on the actual knowledge gaps in students' submitted programs.

3 Research Questions

The limitations of code-informed learning analytics inspired my proposed projects. An overview of the projects is shown in Figure 1. Specifically, these two projects are focused on solving these two research questions:

RQ1: How to enable interpretable deep code learning for performance predictions, guided by learning theory? This will be addressed in Project 1.

RQ2: How to use the interpreted skills to create formative and intelligent feedback to propagate to students? This will be addressed in Project 2.

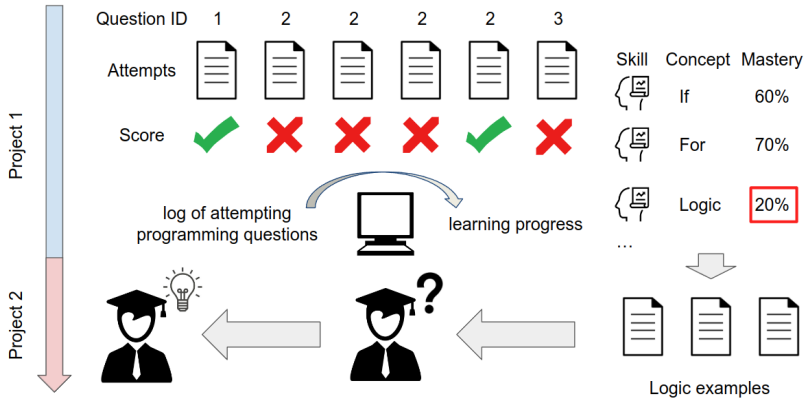

Figure 1: Timeline and concepts of proposed projects

4 Proposed Solution

In Project 1, I will introduce educational context into deep learning models and add interpretability to the models. Originally, the models are only capable of making predictions on students' next problem performance, with massive middle layer parameters uninterpretable. I propose to leverage the layer before prediction as the error rate of certain KCs. To this end, constraints will be needed to guide the learning process of not only the prediction output, but also this layer. Since KCs are often evaluated using learning curves, I propose to hypothesize ideal learning curves for KCs, and calculate the fitness of the layer to the expected learning curve for every KCs. This fitness is added to the loss function to guide the learning of the model. This process will lower the weight of prediction loss, and thus will produce less accurate predictions, and thus hyperparameters should be carefully tuned to find a balance between the fitness of the learning curves for the candidate KCs, and the prediction accuracy of students' performance. The evaluation of the model should also be considered carefully. The candidate KCs should have the property that generates ideal learning curves, but this cannot guarantee that they will represent actual concepts in programming code. Over different problems, they may inconsistently represent code components as well. The values of the candidate KCs should also be examined to check the actual code, and evaluated that 1) if they are consistent within the problem, representing that they are corresponding to certain code components, and 2) if they are consistent throughout different problems, showing if they can be used for unseen problems. More constraints and special design considerations should be added if the concepts represented by the KCs are inconsistent. There are three phases for me to solve this problem: The first one is to create a data-driven model to detect the KC performance from labeled correct submissions. In the second phase, I will use the model and infer the KC performance on incorrect submissions and detect which KCs students don't practice well. Finally, I will incorporate the learning curve analysis into the model and evaluate whether this addition will improve the performance of the KC discovery problem.

Project 2 will be a succeeding project from Project 1. The artifact of Project 1 is a model that given any code submissions, will predict the correctness of students' code submissions, and will show the scores of every KCs and the corresponding code concepts. Low scores on certain concepts can be seen as a lack of skill, and examples will be given to students to learn specific ways to correct their code. A set of examples from various problems will be also processed by the model, and corresponding scores are collected. Students will receive code examples with high scores on the skills that their code achieved a low score, and thus address the specific concepts in their code.

5 Progress and Future Work

During my first 3 years of Ph.D. study, I have published 5 papers in related areas (3 of them are first-authored papers). In LAK'21, I published a paper about how to integrate the code2vec model into CS education, using the model for auto-grading and at the same time discovering student errors from clusters created by the middle layer information of the model. In EDM'21, I presented my paper about using a semi-supervised learning method for bug detection, showing that deep learning models may perform better even without a large dataset, if more unlabeled data is available. In EDM'22, my paper about Code-DKT introduced how to add code information to the DKT model for performance prediction, but since skill-problem mapping is unclear in the CSEd domain. I will submit a paper to AIED'23 to solve address the first research question. The proposed projects lay the foundation for automated teaching and learning support in CS education. While the technical side still has a lot to improve (e.g. performance prediction accuracy, methods to improve code representations in a deep neural model, etc.) as possible future work, there are multiple aspects alongside these projects that are awaiting. The evaluation of interpretability and the quality of the interpretation are yet to be evaluated. Although we could use the additive factors model (AFM) to evaluate the Q-matrices generated, there's a lack of direct evaluation of how the concepts are corresponding to actual knowledge taught in class. For project 2, future evaluations on how students improve using specific feedback are also required. More generally, there is a lot to work on in this field and I plan to work on them after my Ph.D. study.

References

Alon, U., Zilberstein, M., Levy, O., & Yahav, E. (2019). code2vec: Learning distributed representations of code. *In POPL 2019.*

Barnes, T. (2005, July). The Q-matrix method: Mining student response data for knowledge. In *AAAI 2005 educational data mining workshop* (pp. 1-8). AAAI Press, Pittsburgh, PA, USA.

Brandt, J., Guo, P. J., Lewenstein, J., Dontcheva, M., & Klemmer, S. R. (2009, April). Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In *CHI 2009*.

Castelvecchi, D. (2016). Can we open the black box of AI?. *Nature News*, *538*(7623), 20.

Cen, H., Koedinger, K., & Junker, B. (2006, June). Learning factors analysis–a general method for cognitive model evaluation and improvement. In *International conference on intelligent tutoring systems* (pp. 164-175). Springer, Berlin, Heidelberg.

Clark, R. E., Feldon, D. F., van Merriënboer, J. J., Yates, K. A., & Early, S. (2008). Cognitive task analysis. In *Handbook of research on educational communications and technology* (pp. 577-593). Routledge.

Cohausz, L. (2022). Towards real interpretability of student success prediction combining methods of XAI and social science. In *EDM 2022*.

Davies, R., Nyland, R., Chapman, J., & Allen, G. (2015, March). Using transaction-level data to diagnose knowledge gaps and misconceptions. In *LAK 2015*.

Fein, B., Graßl, I., Beck, F., & Fraser, G. (2022). An Evaluation of code2vec Embeddings for Scratch. *In EDM 2022*.

Ju, S., Zhou, G., Azizsoltani, H., Barnes, T., Chi, M. (2019). Identify critical pedagogical decisions through adversarial deep reinforcement learning. *In EDM 2019*.

Koedinger, K. R., Corbett, A. T., & Perfetti, C. (2012). The Knowledge-Learning-Instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive science*, *36*(5), 757-798.

Koedinger, K. R., McLaughlin, E. A., & Stamper, J. C. (2012). Automated Student Model Improvement. *International Educational Data Mining Society*.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436-444.

Leinonen, J. (2022). Open IDE Action Log Dataset from a CS1 MOOC. In *Proceedings of the 6th Educational Data Mining in Computer Science Education (CSEDM) Workshop.*

Mao, Y., Shi, Y., Marwan, S., Price, T. W., Barnes, T., & Chi, M. (2021). Knowing" When" and" Where": Temporal-ASTNN for Student Learning Progression in Novice Programming Tasks. In *EDM 2021*.

Marwan, S., Shi, Y., Menezes, I., Chi, M., Barnes, T., & Price, T. W. (2021). Just a Few Expert Constraints Can Help: Humanizing Data-Driven Subgoal Detection for Novice Programming. In *EDM 2021*.

Nathan, M. J., & Petrosino, A. (2003). Expert blind spot among preservice teachers. *American educational research journal*, *40*(4), 905-928.

Ocumpaugh, J., Baker, R., Gowda, S., Heffernan, N., & Heffernan, C. (2014). Population validity for Educational Data Mining models: A case study in affect detection. *British Journal of Educational Technology*, *45*(3), 487-501.

Paul, W., & Vahrenhold, J. (2013, March). Hunting high and low: Instruments to detect misconceptions related to algorithms and data structures. In *SIGCSE TS 2013*.

Piech, C., Bassen, J., Huang, J., Ganguli, S., Sahami, M., Guibas, L. J., & Sohl-Dickstein, J. (2015). Deep knowledge tracing. *Advances in neural information processing systems*, *28*.

Reyes, J., Ramírez, D., & Paciello, J. (2016, December). Automatic classification of source code archives by programming language: A deep learning approach. In CSCI *2016*.

Rivers, K., Harpstead, E., & Koedinger, K. R. (2016, September). Learning curve analysis for programming: Which concepts do students struggle with? *In ICER 2016.*

Shi, Y., Shah, K., Wang, W., Marwan, S., Penmetsa, P., & Price, T. (2021, April). Toward semi-automatic misconception discovery using code embeddings. *In LAK 2021*.

Shi, Y., Mao, T., Barnes, T., Chi, M., & Price, T. W. (2021, January). More with less: Exploring how to use deep learning effectively through semi-supervised learning for automatic bug detection in student code. *In EDM 2021*.

Shi, Y., Chi, M., Barnes, T., & Price, T. (2022). Code-DKT: A Code-based Knowledge Tracing Model *In EDM 2022*.

Shute, V. J. (2008). Focus on formative feedback. *Review of educational research*, *78*(1), 153-189.

Zhang, J., Wang, X., Zhang, H., Sun, H., Wang, K., & Liu, X. (2019, May). A novel neural source code representation based on abstract syntax tree. In *ICSE 2019.*