A Data-Driven Approach to Automatically Assessing Concept-Level CS Competencies Based on Student Programs

Bita Akram[1], Hamoon Azizolsoltani[1], Wookhee Min[1], Eric Wiebe[1], Anam Navied[1], Bradford Mott[1], Kristy Elizabeth Boyer[2], James Lester[1]

[1]North Carolina State University, Raleigh, North Carolina
{bakram, wmin, hazizso, wiebe, anavied, bwmott, lester}@ncsu.edu

[2]University of Florida, Gainesville, Florida
keboyer@ufl.edu

ABSTRACT

The rapid increase in demand for CS education has given rise to increased efforts to develop data-driven tools to support adaptive CS education. Automated assessment and personalized feedback are among the most important tools for facilitating effective learning experiences for novice students. An important first step in providing effective feedback tailored to individual students is assessing their areas of strength and weaknesses with regard to core CS concepts such as loops and conditionals. In this work, we propose a hypothesis-driven analytics approach to assessing students' competencies of core CS concepts at a fine-grained level. We first label programs obtained from middle grades students' interactions with a game-based CS learning environment featuring block-based programming, based on a rubric that was designed to assess students' competency in core CS concepts from their submitted programs. Then, we train a variety of regression models including linear, ridge, lasso, and support vector regression models, as well as Gaussian process regression models to infer students' scores for each of the identified CS concepts. The evaluation results suggest that Gaussian process regression models often outperform other baseline models for predicting student competencies of core CS concepts with respect to mean squared error and adjusted coefficient of determination. Our approach shows significant potential to provide students with detailed, personalized feedback based on their inferred CS competency levels.

KEYWORDS

Automated Program Assessment, Concept-Level CS Assessment, Gaussian Process Regression, Evidence-Centered Assessment Design

1. INTRODUCTION

As programming has become a fundamental skill in the digital economy, the interest in learning how to program at early ages is rapidly growing [15,37]. However, the complexity of syntax in text-based programming has been found to be a barrier for novice learners [13,14]. To address this challenge, block-based programming environments have replaced textual syntax with visual and elaborative blocks that utilize descriptive text, color, and shape to facilitate programming for novice learners [13,25]. This is particularly beneficial for traditionally underrepresented groups in computer science [6]. Despite the syntax barrier elimination, effective and tailored scaffolding and feedback is still required to support students' mastery of computer science (CS) concepts that are essential for programming. Providing students with effective scaffolding and feedback would significantly benefit from reliable assessments that can evaluate student competencies with respect to core CS concepts [12,20]. Effective assessment can inform adaptive pedagogical strategies such as offering hints and feedback and performing tailored problem selection. Automated assessments can bridge the gap between the growth in demand for CS education and the limited supply of qualified teachers.

While research on conducting automated assessment of student-generated programs is gaining momentum, limited previous work has yielded methods to infer students' mastery of fine-grained CS concepts exercised in a particular computational problem. Previous work has focused on predicting an overall score to represent students' general level of mastery in programming [17,19]. However, identifying students' strength and weakness on specific CS concepts could enable instructors to provide students with adaptive scaffolding and tailored practices needed to master the those concepts. In addition, fine-grained assessment of CS competencies can inform intervention strategies for intelligent learning environments to perform student-adaptive hint and feedback generation as well as problem selection. Furthermore, using this information in an open learner model could enable students to focus on areas in which they need more practices by monitoring mastery in CS concepts [7, 35].

We follow a hypothesis-driven learning analytic approach [14] based on Evidence-Centered Design (ECD) [22] to identify core CS concepts highlighted in a learning environment and to assess students' competencies in relation to each of the target concepts. We explore this in the context of a bubble-sort challenge within the ENGAGE game-based learning environment (Figure 1) that requires

implementing a program using a block-based programming interface. Based on the hypothesis-driven learning analytic approach, we first identify the CS concepts that are targeted by this activity. We also collect students' submitted solutions to this challenge in the form of snapshots of their submitted programs. Content area experts then use this information to devise a rubric that can identify students' mastery of targeted CS concepts based on evidence captured from their program artifacts. Examples of CS concepts and practices for the bubble sort challenge are developing appropriate algorithms and programs, and appropriate use of computer science constructs, such as loops and conditionals.

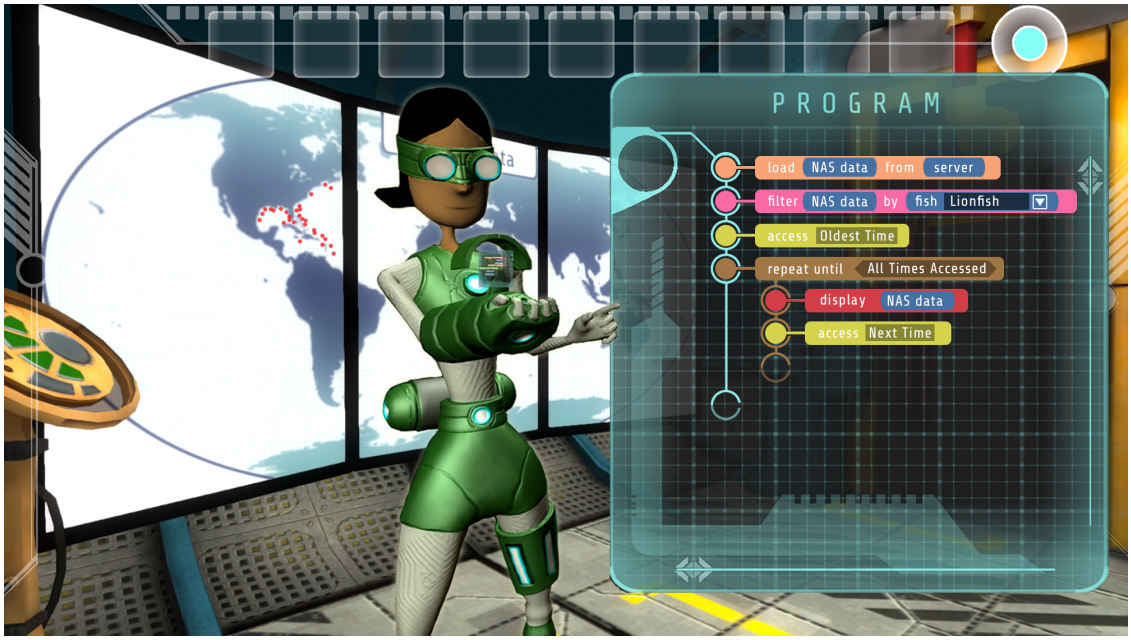

Figure 1. ENGAGE game-based learning environment: students write a program to filter a data set and loop over it.

We use the rubric to label our training dataset. We further extract structural and semantic information from program snapshots by encoding them as structural *n*-grams following the approach in [1]. A variety of regression models including lasso, ridge, support vector regression and Gaussian process regression (GPR) models are applied on the generated feature set to infer students' competencies for their overall grade and for each of the identified target CS concepts. We hypothesize that GPR models are particularly suitable for this type of inference task as they are capable of handling the noise resulting from the subjective process of grading programs. The results demonstrate the effectiveness of these models for predicting students' CS competencies.

2. RELATED WORK

Two approaches to automatic program assessment can be distinguished: dynamic assessment and static assessment [5, 16]. Dynamic assessment is used to assess the correctness of completed programs using pre-defined test data [16, 18, 31]. Static assessments on the other hand, can assess partial programs for partial correctness. To perform this latter form of assessment an important step is transforming the program into an intermediate representation such as abstract syntax trees, control flow graphs, and program dependence graphs. The intermediate representation is then evaluated to determine its degree of correctness, efficiency and quality. Since all programming languages, including block-based programming languages can be represented using the same intermediate representations, static assessment techniques are syntax-free and can be adapted to assess any programming language.

In static assessments, correctness is usually assessed through character analysis, string analysis, syntax analysis, and semantic analysis. Quality is assessed by software metrics such as the number of lines of code, the number of variable statements, and the number of expressions. For example, work by Wang and colleagues presented a semantic similarity-based approach to assess the correctness of a C program by comparing it against a correct program model [34]. In this work, they first reduced the state space of programs by conducting a set of program standardizations including expression, control structure, and function invocation standardization. Then, they calculated a similarity factor based on size, structure and statement similarity subfactors weighted by grading criteria.

Most work on static assessment utilizes a variety of similarity measurements to calculate the relative correctness of a program in reference to other scored programs [30, 33]. However, the approaches described above typically yield an overall score rather than a fined-grained analysis of student competencies on specific CS concepts, skills, and knowledge. In an educational context, this detailed diagnostic information is essential for providing students with the targeted scaffolding and support they need. In this paper, we propose an automated assessment framework that can provide students and their instructors with an automated assessment tool that is both detailed and interpretable.

3. Learning Environment and Dataset

In this study, we collect data from middle-grade students' interactions with the ENGAGE game-based learning environment

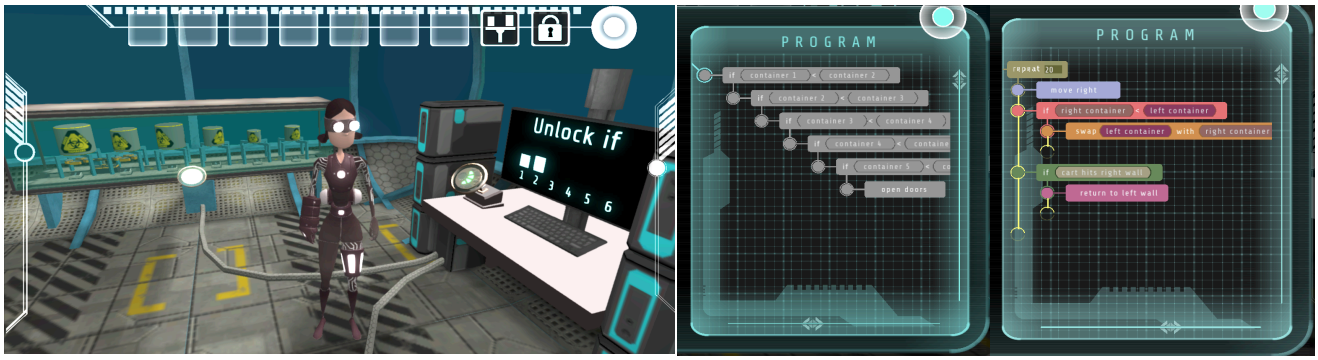

Figure 2. ENGAGE game-based learning environment. (Left) The bubble sort task in the game-based learning environment. (Right) Program for the bubble sort task: the read-only code for opening the door and an example of a correct implementation of the bubble sort written by a student.

that is a computational thinking (CT) focused education game for middle school students (ages 11-13) [2]. The CS content of the ENGAGE game is based on the AP CS Principles curriculum [9]. Students learned CS competencies ranging from abstraction and algorithmic thinking to computational problem solving and programming. The computational challenges within the game were designed to prepare students for computer science work in high school, and to promote positive attitudes toward computer science. This game features an underwater research station that has lost connection with the outside world and students are sent as computer science specialist to investigate the issue (Figure 1, Left) [21]. To successfully complete the game, students need to move around the station and solve different computational thinking problems with block-based programming.

The focus of this work is on a particular CT challenge where students need to implement a bubble sort algorithm using block-based programming (Figure 1, Right) to escape a room. Students implement an algorithm to sort six randomly positioned containers within a containment device. Once the containers are sorted student can open the door and escape the room. To write the bubble-sort algorithm students have access to a limited number of necessary blocks including: a *repeat* block that repeats every nested block for a certain amount of time specified by students; a *conditional* block that checks whether the right container is smaller than the left container; a *conditional* block that checks whether the cart has hit the right wall; a *swap containers* block that swaps a container with its adjacent right container; a *move* block that moves the cart one position toward the right; and a *reset* block that brings the cart to the left most position.

On average, students played the game over the course of two weeks. As students interacted with the game, all of their interactions with the game were logged, such as dragging programming blocks to write a program and programs being executed. For this study, we collected data from middle grade students' interactions with the bubble sort algorithm. Data was collected from five classrooms in three schools in the United States. The data used for this study is from 69 consented students.

4. METHOD

A supervised learning approach is utilized to infer students' competency for each of the CS concepts identified for the bubble-sort algorithm task. To infer students' grades, we first label their program snapshots utilizing the rubric presented in Table 1. We then transform students' program snapshots to a feature set utilizing a novel *n*-gram encoding approach following [1]. Finally, we infer students' scores by applying regression models on the structural *n*-gram-based feature set.

In comparison to [1], our work focuses on assessing students' mastery of identified, individual CS concepts underlying the bubble sort challenge. By utilizing this assessment framework, we can train separate regression models utilizing the *n*-gram feature set to assess student programs based on each individual CS concept. In this study, we utilize a variety of regression models including linear, lasso, ridge, support vector regression (SVR), and Gaussian process (GP) regression models to predict algorithmic quality of students' programs.

4.1 Rubric Design

Students' programs are labeled utilizing a rubric that is devised following and evidence-centered assessment design (ECD) approach [27]. An important first step in ECD is domain modeling where relevant CS concepts are identified through the collaborative work of domain experts and teachers [11, 22]. The CS concepts are then used to develop the specifications of an assessment of the CS concepts. The conceptual assessment framework consists of the following: 1) the *student model*, which represents what students know or can do; 2) the *evidence model*, which contains evidence that drives the student model; and 3) *the task model*, which contains tasks, interactions with which can generate evidence [22, 29].

Evidence is derived from students' actions during the learning tasks to predict their mastery of CS concepts. An important requirement is to match evidence derived from student programs to proficiency in each CS concept covered in the assessment. In this work, the student model represents students' knowledge of particular CS concepts, the evidence model is based on evidence rules that extract the program structures from their programs representing their knowledge of each identified CS concept, and the task model is the bubble sort challenge in the ENGAGE game-based learning environment. Following this approach, we design evidence rules specific to the task at hand to provide assessment arguments for the proficiency of the CS concepts in our student model. Table 1 shows the rubric for assessing the CS concepts identified through the domain modeling phase [3, 4].

4.2 Data Annotation

Our training dataset contains 1,570 programs submitted by 69 students when solving the bubble sort challenge. The algorithmic "Effectiveness" and "Conciseness" scores are two mutually exclusive metrics designed to capture core qualities of programs. For example, a program that contains all the necessary coding constructs to receive full points for "appropriate use of conditional statements" might contain redundant copies of the same coding constructs that interfere with the correctness of the algorithm. This deficiency is captured in the "Conciseness" score. A similar program might have the wrong ordering of the coding constructs that negatively affects the correctness of the algorithm. This is captured through by the "Effectiveness" score. In this rubric, the range of possible scores for the "design and implementation of effective and generalizable algorithm" is between 0-10. Similarly, this range is between 0 to 3 for "Appropriate use of loop statements," between 0 to 6 for "Appropriate us of conditional statements," and between 0 to 3 for "Appropriate combination of loops and conditional statements." The overall score (overall algorithmic quality score) range is between 0 to 22.

Two annotators with CS background annotated 20% of the submissions for algorithmic effectiveness and conciseness scores. Using Cohen's kappa [8], an inter-rater agreement of 0.848 for "effectiveness" and 0.865 for "conciseness" was achieved. The two annotators discussed their disagreements and one annotator tagged the remainder 80% of the dataset. These annotations are served as the ground-truth for our data corpus. It is important to note that the annotation process introduces noise into the training dataset [23]. This is because different scorers may have different perceptions of a program's algorithmic "Effectiveness" and "Conciseness." As a result, the dataset is inherently noisy, which must be taken into account when designing the models for the automated assessment framework. To handle this uncertainty, we adopt a Gaussian process regression model that returns a distribution for the inference values including an average with standard deviation.

4.3 Feature Engineering

We use abstract syntax trees (ASTs) as the intermediate representation for our automated assessment task. After transforming students' program snapshots to their corresponding ASTs [28], we encode them as structural *n*-grams to extract features that are representative of the semantic information in students' programs following the previous work [1]. *Hierarchical* and *ordinal n*-grams are two important structures in an AST. The parent child relationship between different blocks are encoded in *hierarchical* structures and the placement order of blocks are encoded in ordinal structures. To enable the proposed automated assessment to assign partial scores to incomplete solutions, we need to extract *n*-grams with varying lengths of *n* to capture the most fine-grained structural information present in an AST.

An AST generated from a sample program is demonstrated in Figure 3. A partial hierarchical (left) and ordinal (right) *n*-gram encoding is also demonstrated in this Figure. In Figure 3, each colored circle shows the *n*-gram encoding of a specific *n*. In this example, encoding of *n*-grams of size one is represented with green ovals, *n*-grams of size two with blue ovals and *n*-grams of size three with purple ovals. The frequency values for each *n*-gram encoded feature are shown beside the AST. All of the other *n*-gram feature values are zero since they are not in this AST. We then merge the two feature sets together to build the final feature set containing both hierarchical and ordinal *n*-gram encodings corresponding to each program. Note that unigrams are repeated in both hierarchical and ordinal *n*-gram encoding of the ASTs, and thus, only one copy of unigram features is used in the final feature set. The occurrence of similar *n*-grams for *n* values more than one (unigrams) in both hierarchical and vertical encodings demonstrate presence of different structures in in the AST and thus, both will be preserved.

Table 1. Assessment items, and detailed rubrics for each item.

CS Concepts and Practices	Detailed Rubric
Design and implementation of effective and generalizable algorithms	• The program contains all necessary code elements. • The code elements have the correct order, and hierarchy (Effectiveness). • The program does not contain redundant code elements that falsify the logic of the algorithm (Conciseness).
Appropriate use of loop statements	• The repeat block is present. • The iteration value is set to a positive number. • It encompasses at least one block.
Appropriate use of conditional statements	• Both necessary conditional • statements are used. • A conditional statement checks the size of two adjacent containers and swaps them if they are not ordered properly. • A conditional statement checks if the arm has reached the right wall and reset it to the left wall.
Appropriate combination of loops and conditional statements	• There is at least one instance of each conditional nested under a repeat statement. • There is at least one instance of two conditionals at the same level.

Preliminary explorations revealed that including sequences of lengths larger than 4 for hierarchical *n*-grams and 3 for ordinal *n*-grams exponentially increases the sparsity of the dataset. To address the sparsity issue, we capped the *n*-gram size at 4 for the hierarchical *n*-gram encoding and at 3 for the ordinal *n*-gram encoding. The final feature set consists of sequences of length one (i.e., unigrams) to sequences of length four for hierarchical (i.e., 4-grams) and three for ordinal (i.e., 3-grams) that are repeated at least three times (again to address the data sparsity issue) throughout the dataset, resulting in 184 distinct features.

4.4 Inferring Program Scores

We infer students' overall program scores in addition to their scores for each of the essential CS concept by training a variety of regression models on the structural *n*-gram-encoded feature set. As

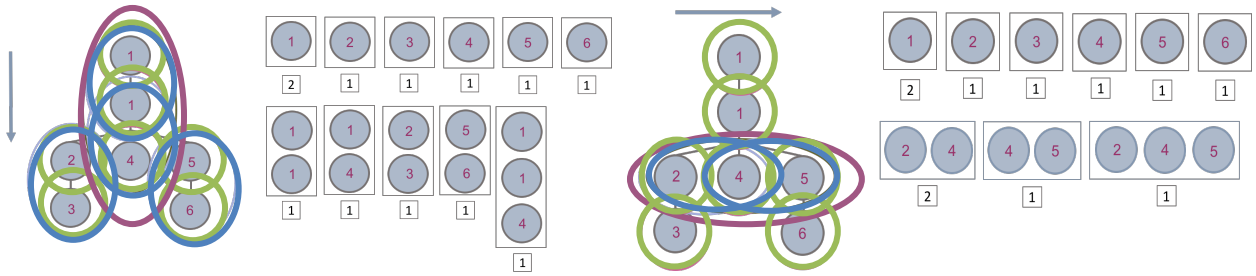

Figure 3: AST generated from a sample program submitted for the bubble sort challenge and its hierarchical and ordinal *n*-gram encoding. (Left) An AST and its partial *hierarchical* unigrams, bigrams, and *3*-grams marked by green, blue and purple ovals respectively on the left and the partial feature set generated from hierarchical *n*-gram encoding of the AST along with feature-level frequencies on the right. (Right) An AST and its *ordinal* unigrams, bigrams, and *3*-grams marked by green, blue and purple ovals respectively on the left and the partial feature set generated from partial ordinal *n*-gram encoding of the AST along with feature-level frequencies on the right.

our baseline model, we use linear regression. We use four additional regression models including lasso regression [32], ridge regression, support vector regression (SVR) [10], and Gaussian process (GP) regression [26]. Lasso and Ridge regression are utilized since they can reduce overfitting and variance issues in comparison with linear regression. SVR and GP regression on the other hand are used since kernel methods can do well with datasets with proportionally large number of features. More importantly, GP regression can handle the noise resulting from the subjective nature of human grading [6, 36]. To infer students' overall program scores as well as their scores for "Design and implementation of effective and generalizable algorithms,", "Appropriate use of loop statements," "Appropriate use of conditional statements," and "Appropriate combination of loops and conditional statements," we train each regression model utilizing the *n*-gram encoded feature set mentioned above, while predicting the scores of each core concept.

To infer students' grades using the *n*-gram encoded feature set, we use the Python scikit-learn library [24] to perform linear, lasso regression, ridge regression, SVR, and GP regression. We first split our dataset to 80% training and 20% held-out test sets. We the use a 5-fold cross-validation approach to tune the hyperparameters of lasso, ridge, and SVR regression based on the training set. We also use the 5-fold cross-validation approach to identify the appropriate kernel for the GP regression model. Gaussian process regression model uses an internal limited-memory BFGS approach to tune its other hyperparameters such as length scale and noise level. After tuning the hyperparameters of each regression model, we train the models on the training set and evaluate it on the held-out test set. This process is repeated to infer each CS concept score separately. The results of applying each of the regression models to infer each of the CS concept scores is presented in Table 2.

4.4.1 Ridge Regression

We used the set [0.05, 0.1, 0.5, 1.0, 10] to tune the value for λ, the penalty coefficient, and found λ=10 to be the best value for inferring the "Overall grade" and the "Design and implementation of effective and generalizable algorithm" scores based on cross-validation. Furthermore, we found λ=0.5 the best value for "Appropriate use of loop statements" score, and λ=1 the best value for "Appropriate use of conditional statements" and "Appropriate combination of loops and conditional statements" scores.

4.4.2 Lasso Regression

We used the set [0.05, 0.1, 0.5, 1.0, 10] as in ridge regression to tune the value for λ and found λ=0.05 to be the best value for all the inferred scores.

4.4.3 Support Vector Regression

For our regression task, we explored with linear, polynomial, and radial basis function (RBF) kernels. For each kernel, we tuned the hyperparameters of penalty parameter (*C*), epsilon, and kernel coefficient (gamma). For polynomial kernels, we also tuned the parameter of the kernel projection (*coef0*) and degree hyperparameters. Utilizing cross-validation, we found the polynomial kernel with a degree of four to be the best kernel for our dataset when inferring the "Overall Score", "Appropriate use of of loop statements," and "Appropriate combination of loops and conditional statements" scores. Also, the grid search returned *C*=1, *coef0*=10, epsilon=0.2, gamma= 0.0001 as the best parameters for this kernel. For inferring the "Appropriate use of conditional statements" score we found the radial basis function kernel with *C*=100, epsilon=0.1, and gamma= 0.001 to be the best parameter values. Finally, we found the radial basis function kernel with *C*=100, epsilon=0.2, and gamma= 0.001 to be the best parameter values when inferring the "Design and implementation of effective and generalizable algorithm" score.

4.4.4 Gaussian Process Regression

We expect the GP regression to outperform other regression techniques due to its capability of handling noise and its propriety for our dataset. After conducting a hyper-parameter tuning for the kernels such as radial basis functions (RBF), rational quadratic, and Matern kernels, we found RBF to perform the best on our dataset for all the inferred scores. Utilizing a limited-memory BFGS optimization technique the GP regression model tuned other hyperparameters including the length vector and noise level during the training process.

Table 2. Average predictive performance of regression models trained with the structural *n*-gram feature set.

Grade	Overall Grade		Design and implementation of effective and generalizable algorithm		Appropriate use of loop statements		Appropriate use of conditional statements		Appropriate combination of loops and conditional statements	
Regression	MSE	R^2	MSE	R^2	MSE	R^2	MSE	R^2	MSE	R^2
Linear	7.88E+10	-2.58E+9	1.8E+10	-3.9E+9	2.8E+9	-2.1E+9	2.43E+9	-4.9E+8	2.0E+9	-1.6E+9
Ridge	4.84	0.84	1.62	0.64	0.36	0.74	0.59	0.88	0.09	0.93
Lasso	5.67	0.81	2.38	0.49	0.88	0.35	0.73	0.85	0.48	0.62
SVR	3.74	0.88	1.56	0.66	0.57	0.57	0.49	0.9	0.08	0.93
Gaussian Process	1.67	0.94	1.02	0.78	0.18	0.86	0.28	0.94	0.02	0.98

5. DISCUSSION

Effective automated assessment of students' programming efforts has become increasingly important. This work investigates an *n*-gram encoding approach to encode students' programs into their essential structural and semantic features. Utilizing the *n*-gram encoding approach, we can extract structural information with varying levels of granularity. Utilizing this feature set labeled by the ECD-based designed rubric enables our models to learn evidence from programs that are representative of students' mastery of identified CS.

After extracting an *n*-gram encoded feature set from students' programs, we apply a variety of regression models to infer their scores for each of the targeted CS concepts. We conduct an 80-20 split on our dataset to generate training and held-out test sets. We train our models on the training set and evaluate the trained models on the held-out test. This process is repeated for each CS concept. The results of our prediction demonstrate the effectiveness of the *n*-gram encoded feature set in capturing important semantic and structural information in students' programs, as all regression models outperformed the linear regression model. As expected, GP regression also outperformed other baseline models in terms of both mean squared error and R-squared across all prediction tasks. This is expected, since GP regression is well-equipped to handle noise in the data set and is particularly appropriate for datasets with a large number of features relative to the number of data points.

We utilized an evidence-centered assessment design (ECD) approach to label the training dataset. ECD holds significant promise for guiding educators in designing mindful assignments for learners by focusing on key conceptual ideas rather than surface-level features of the program. This means that an ECD-derived rubric can provide granular information structured around core CS concepts, which guides development of robust automated assessment models, but also provide immediate formative data to instructors. Thus, as new problems and activities are introduced into a course, the first-pass human scoring with the rubric provides immediate actionable formative information while also training automated assessment tools that can provide ongoing, future adaptive support.

Though we show the application of this automated assessment framework on one particular task, it can be generalized to assess any well-structured programs as the feature representations are readily scalable to other programming tasks. Furthermore, our rubric design approach can be used as a guideline for rubric design and assessment for non-expert CS teachers. A teacher dashboard incorporating the automated assessment framework can further be utilized to analyze and aggregate the results and inform teachers about students' learning and the quality of their instruction.

6. CONCLUSION AND FUTURE WORK

Effective scaffolding of programming efforts for novice programmers require accurate automated assessment of their competency in each core CS concept. In this paper, we presented an automated assessment framework for assessing programs' algorithmic quality following a hypothesis-driven learning analytic approach. We investigate a hierarchical, ordinal feature representation method based on *n*-gram-encoded hierarchical and ordinal coding constructs that extract two-dimensional structural information from students' programs, and investigated Gaussian process regression to induce models that can accurately predict students' grades for individual CS concepts based on their submitted programs. Evaluation results suggest that Gaussian process regression models utilizing *n*-gram-encoded features that extract salient semantic and structural information from programs achieved the highest predictive performance with respect to mean squared error and R squared. These results suggest that Gaussian process regression models are robust in dealing with noise that underlies our human-annotated dataset.

In the future it will be important to investigate the potential for utilizing a data-driven approach for devising a rubric based on identified correct solutions. Furthermore, the effectiveness of the *n*-gram encoded feature set can be further evaluated by performing

an automatic feature-selection process and compare the results with expert selected features. Finally, it will be instructive to explore the potential of the *n*-gram encoded feature for creating an unsupervised learning approach to accurately inferring students' program scores without requiring labeled training data.

7. ACKNOWLEDGMENTS

This research was supported by the National Science Foundation under Grants DRL-1640141. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

[1] Akram, B., Azizolsoltani, H., Min, W., Wiebe, E., Navied, A., Mott, B., Boyer, K., and Lester, J. 2020. Automated Assessment of Computer Science Competencies from Student Programs with Gaussian Process Regression. *To appear In Proceedings of the 13th Conference on Educational Data Mining*.

[2] Akram, B., Min, W., Wiebe, E., Mott, B., Boyer, K., and Lester, J. 2018. Improving Stealth Assessment in Game-based Learning with LSTM-based Analytics. In *Proceedings of the 11th International Conference on Educational Data Mining*, 208–218.

[3] Akram, B., Min, W., Wiebe, E., Navied, A., Mott, B., Boyer, K., and Lester, J. 2020. A conceptual assessment framework for k-12 computer science rubric design. In *Proceedings of the 51th ACM Technical Symposium on Computer Science Education*, 1328–1328.

[4] Ala-Mutka, K. 2005. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education* 15, 2, 83–102.

[5] Amershi, S. and Conati, C. 2009. Combining Unsupervised and Supervised Classification to Build User Models for Exploratory Learning Environments. *Journal of Educational Data Mining, Article* 1, 1, 18–71.

[6] Bouchet, F., Harley, J., Trevors, G., and Azevedo, R. 2013. Clustering and profiling students according to their interactions with an intelligent tutoring system fostering self-reggulated learning. *Journal of Educational Data Mining*, 05, 01, 104–146.

[7] Cohen, J. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement* 20, 1, 37–46.

[8] College Board. 2017. AP Computer Science Principles Including the Curriculum Framework. In *AP Computer Science Principles: The Course*. New York.

[9] Cortes, C., and Vapnik, V. 1995. Support-vector networks. *Machine Learning* 20, 3, 273–297.

[10] Cui, Y., Chu, M., and Chen, F. 2019. Analyzing Student Process Data in Game-Based Assessments with Bayesian Knowledge Tracing and Dynamic Bayesian Networks. *Journal of Educational Data Mining* 11, *0*1, 80–100.

[11] Fields, D., Giang, M., and Kafai, Y. 2014. Programming in the wild: trends in youth computational participation in the online scratch community. In *Proceedings of the 9th Workshop in Primary and Secondary Computing education*, ACM, 2–11.

[12] Grover, S., and Basu, S. 2017. Measuring Student Learning in Introductory Block-Based Programming. In *Proceedings of the 48th ACM SIGCSE Technical Symposium on Computer Science Education*, 267–272.

[13] Grover, S., Basu, S., Bienkowski, M., Eagle, M., Diana, N., and Stamper, J. 2017. A Framework for Using Hypothesis-Driven Approaches to Support Data-Driven Learning Analytics in Measuring Computational Thinking in Block-Based Programming Environments. *ACM Transactions on Computing Education* 17, 3, 1–25.

[14] Hansen, A., Dwyer, H., Iveland, A., Talesfore, M., Wright, L., Harlow, D., and Franklin, D. 2017. Assessing Children's Understanding of the Work of Computer Scientists: The Draw-a-Computer-Scientist Test. In *Proceedings of the 48th ACM SIGCSE Technical Symposium on Computer Science Education*, 279–284.

[15] Phantola, P., Ahoniemi, T., Karavirta, V., and Seppälä, O. 2010. Review of Recent Systems for Automatic Assessment of Programming Assignments. In *Proceedings of the 10th Koli calling International Conference on Computing Education Research*, 86–93.

[16] Deirdre Kerr and Gregory K W K Chung. 2012. Identifying Key Features of Student Performance in Educational Video Games and Simulations through Cluster Analysis. *Journal of Educational Data Mining* 4, 1, 144-182.

[17] Lajis, A., Baharudin, S., Kadir, D., Ralim, N., Nasir, H., and Aziz, N. 2018. A Review of Techniques in Automatic Programming Assessment for Practical Skill Test. *Journal of Telecommunication, Electronic and Computer Engineering* 10, 2, 109–113.

[18] Mao, Y., Lin, C., Chi, M., 2018. Deep Learning vs. Bayesian Knowledge Tracing: Student Models for Interventions. *Journal of Educational Data Mining* 10, 02, 28–54.

[19] Meerbaum-Salant, O., Armoni, M., and Ben-Ario, M. 2013. Learning computer science concepts with scratch. *Computer Science Education* 23, 3, 239–364.

[21] Min, W., Frankosky, M., Mott, B., Rowe, P., Wiebe, E., Boyer, E., and Lester, J. 2015. DeepStealth: Leveraging Deep Learning Models for Stealth Assessment in Game-based Learning Environments. In *International Conference on Artificial Intelligence in Education*, 277–286.

[22] Mislevy, R., Haertel, G., Riconscente, M., Rutstein, D., and Ziker, C. 2017. Evidence-Centered Assessment Design. In *Assessing Model-Based Reasoning Using Evidence- Centered Design*. SpringerBriefs in Statistics, 19–24.

[23] Multon, K. 2010. Interrater reliability. *Encyclopedia of Research Design*. SAGE, New York, 626–628.

[24] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., 2011. Scikit-

learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

[25] Price, T., Dong, Y., and Lipovac, D. 2017. iSnap: Towards Intelligent Tutoring in Novice Programming Environments. In *Proceedings of the 48th ACM SIGCSE Technical Symposium on Computer Science Education*, ACM, 483–488.

[26] Rasmussen, C.. 2004. Gaussian Processes in machine learning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 63–71.

[27] Rupp, A, Pearson, K., Sweet, S., Crawford, A., Levy, I., Fay, D., Kunze, K., Cisco, M., Mislevy, R., and Pearson, J. 2012. Putting ECD into Practice: The Interplay of Theory and Data in Evidence Models within a Digital Learning Environment. *Journal of Educational Data Mining* 4, 1 , 49–110.

[28] Shamsi, F., and Elnagar, A. 2012. An Intelligent Assessment Tool for Students' Java Submissions in Introductory Programming Courses. *Journal of Intelligent Learning Systems and Applications* 04, 01 , 59–69.

[29] Snow, E., Haertel, G., Fulkerson, D. and Feng, M. 2010. *Leveraging evidence-centered assessment design in large-scale and formative assessment practices*. In *Proceedings of the 2010 Annual Meeting of the National Council on Measurement in Education (NCME).*

[30] Striewe, M., and Goedicke, M. 2014. A review of static analysis approaches for programming exercises. In *Computer Assisted Assessment. Research into E-Assessment*. Springer, 100–113.

[31] Taherkhani, A., and Malmi, L. 2013. Beacon- and Schema-Based Method for Recognizing Algorithms from Students' Source Code. *Journal of Educational Data Mining* 5, 2, 69–101.

[32] Tibshirani, R. 1996. Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)* 58, 1 267–288.

[33] Truong, N.,Roe, P., and Bancroft, P. 2004. Static Analysis of Students' Java Programs. In *Proceedings of the 6th Australasian Conference on Computing Education,* 317-325.

[34] Wang, T., Su, X., Wang, Y., and Ma, P. 2007. Semantic similarity-based grading of student programs. *Information and Software Technology* 49, 2, 99–107.

[35] Winne, P., and Baker, R. 2013. The Potentials of Educational Data Mining for Researching Metacognition, Motivation and Self-Regulated Learning. *Journal of Educational Data Mining* 5, 1, 1–8.

[36] Zen, K., Iskandar, D,. and Linang O. 2011. Using Latent Semantic Analysis for automated grading programming assignments. In *International Conference on Semantic Technology and Information Retrieval*, 82–88.

[37] 2016. K-12 Computer Science Framework. Retrieved August 25, 2018 from http://www.k12cs.org.