Predictive Student Modeling in Block-Based Programming Environments with Bayesian Hierarchical Models

Andrew Emerson[1] Michael Geden[1] Andy Smith[1] Eric Wiebe[1] Bradford Mott[1]
Kristy Elizabeth Boyer[2] James Lester[1]
[1]North Carolina State University, Raleigh, North Carolina, USA
[1]{ajemerso, mageden, pmsmith4, wiebe, bwmott, lester}@ncsu.edu
[2]University of Florida, Gainesville, Florida, USA
[2]keboyer@ufl.edu

ABSTRACT

Recent years have seen a growing interest in block-based programming environments for computer science education. Although block-based programming offers a gentle introduction to coding for novice programmers, introductory computer science still presents significant challenges, so there is a great need for block-based programming environments to provide students with adaptive support. Predictive student modeling holds significant potential for adaptive support in block-based programming environments because it can identify early on when a student is struggling. However, predictive student models often make a number of simplifying assumptions, such as assuming a normal response distribution or homogeneous student characteristics, which can limit the predictive performance of models. These assumptions, when invalid, can significantly reduce the predictive accuracy of student models.

To address these issues, we introduce an approach to predictive student modeling that utilizes Bayesian hierarchical linear models. This approach explicitly accounts for individual student differences and programming activity differences by analyzing block-based programs created by students in a series of introductory programming activities. Evaluation results reveal that predictive student models that account for both the distributional and hierarchical factors outperform baseline models. These findings suggest that predictive student models based on Bayesian hierarchical modeling and representing individual differences in students can substantially improve models' accuracy for predicting student performance on post-tests. By improving the predictive performance of student models, this work holds substantial potential for improving adaptive support in block-based programming environments.

CCS CONCEPTS

• Social and professional topics → Computing education; • Applied computing → Education

KEYWORDS

Predictive Student Modeling, Bayesian Hierarchical Modeling, Block-Based Programming

ACM Reference format:

Andrew Emerson, Michael Geden, Andy Smith, Eric Wiebe, Bradford Mott, Kristy Elizabeth Boyer, and James Lester. 2020. Predictive Student Modeling in Block-Based Programming Environments with Bayesian Hierarchical Models. In *Proceedings of Twenty-Eighth ACM International Conference on User Modeling, Adaptation, and Personalization (UMAP'20). ACM, New York, NY, USA, 10 pages.*

1 Introduction

Introductory computer science education poses significant challenges to students, including unfamiliar syntax and obscure compiler errors. To address these challenges, block-based programming environments have been introduced to simplify introductory programming by enabling students to create programs by dragging and connecting blocks. Block-based programming is becoming an increasingly widely used approach to providing novice students with a smoother transition into computing, which increases overall interest and improves learning in computer science [28]. Block-based environments eliminate the majority of syntax and compiler errors that often frustrate novice learners, and have been shown to reduce students' overall cognitive load [30].

Despite these benefits, block-based environments currently provide limited support to students beyond the structure of the blocks themselves, such as hints or feedback related to semantic errors in the program, relying on instructors to fill any gaps in student knowledge or ability [19]. Predictive student modeling provides a potential solution, as models of student performance can inform learning environments in real time when a student is struggling [7, 8, 12]. Predictive student models can inform adaptive support systems by providing real-time formative assessments of student knowledge. However, many approaches to student modeling rely on techniques that can detrimentally impact the predictive performance of student models and their interpretation due to statistical assumptions related to the underlying data [3].

Block-based programming environments pose significant challenges for predictive student modeling. Programming activities are typically multi-step, with both multiple possible correct solutions, as well as a wide variance in incorrect solutions. In addition, there is often a large variation in the number of activities completed by individual students, particularly when activities are completed outside of the classroom in online environments. Further, students exhibit a wide range of incoming abilities in programming, thereby substantially increasing the need to model individual student competencies.

In this work we present a novel approach to predictive student modeling for block-based programming. First, we capture student programming performance using a novel encoding based on a distance metric to an expert solution. Using this encoding, we utilize a novel Bayesian hierarchical regression framework using alternative distributions and participant-matching for estimating random effects. By choosing distributions that more accurately reflect the distribution of the response variable and its associated residuals, predictive performance is improved and inference using the parameters remains valid. A hierarchical component to the Bayesian models accounts for both student-level characteristics as well as programming activity-level characteristics. Additionally, by using a Bayesian framework, implemented using L1 (Lasso) regularization on the features, we maintain interpretability that many student models lose by relying on more complex, less scrutable computational techniques.

Results from a comparison of the normal, Poisson, and negative binomial distributions in the proposed Bayesian regression model shows that both the Poisson and negative binomial distributions, which are both used to model count data, outperform the normal distribution in modeling student performance. We then add random effects corresponding to the specific student and activity for which the model is making predictions. The trained models demonstrate substantial improvements in predictive performance over the baseline Bayesian linear models, and they reveal that models augmented with the student-level random effects outperform all other models. However, the activity-level random effect provided only marginal improvements compared to the student-level effects. These results highlight the effectiveness of the framework in modeling student knowledge in a block-based programming environment, as well as demonstrating the potential positive impacts of incorporating non-normal distributions and student-level parameters into student modeling frameworks.

2 Related Work

Predictive student modeling is a type of student modeling that seeks to predict future student performance based on students' prior behaviors in a learning environment. For example, many student models built using Bayesian Knowledge Tracing (BKT) [6] seek to predict future performance on activities involving a given concept based on students' correct or incorrect responses to a previous series of activities involving that concept. Recent work has sought to improve the accuracy of these models by augmenting them with student-specific parameters [31], utilizing more complex underlying computational models [17], and incorporating additional factors such as inter-skill similarity [13]. For open-ended tasks, other models have shown improved accuracy by representing students' performance on a continuous rather than binary scale (i.e., incorrect or correct) [33]. Sao Pedro et al. were able to apply a combination of human coded behaviors and machine learning techniques to detect inquiry behaviors in an open-ended science learning environment [24]. Additionally, several groups have developed models based on evidence-centered design [16, 23, 26], seeking to model student knowledge by linking observed actions in a learning environment to conceptual knowledge through an evidence model. These models typically evaluate their predictions of student knowledge based on how well they predict a student's performance on a summative assessment.

Most student modeling related to programming has centered on text-based programming activities, including attempts to cluster similar student behavior [5], or identify which concepts are causing students the most difficulty [22]. Because most models of learners in text-based programming tend to focus on issues uncommon to block-based environments, such as syntax and compiler errors, there is a growing body of research focused specifically on learners in block-based programming environments built upon *Snap!* or Google's Blockly. Price et al. utilized an edit distance metric to compare the current state of a student's program to previously logged coding trajectories in order to generate next-step hints [20]. Mao et al. used Recent Temporal Patterns to predict student performance in an open-end programming task [14]. Grover et al. combined both hypothesis-driven approaches with data-driven frameworks to measure computational thinking ability in block-based programming activities [11]. Other research has utilized both problem-solving behaviors [2, 15] and block-based programming trajectories [1] derived from student interactions in a game-based learning environment for computational thinking to predict student performance on a post-test assessment.

Previous work in predictive student modeling has typically relied on the assumptions that the response variable follows the normal distribution and is independent and identically distributed. These assumptions, when incorrect, can significantly reduce the predictive accuracy of student models. For example, Arthurs et al. investigated the response variable distribution and found that using the logit-normal distribution improved predictive performance compared to the normal distribution [3]. Yudelson et al. further described how assuming homogeneous student data in BKT can lead to limited predictive performance [32]. Other work has shown significant improvements by incorporating hierarchical components into their models that account for differences in groups of data, treating student data differently depending on its context [4, 25]. This work builds on these families of prior work to create student models for block-based programming that significantly improve the accuracy of predictive student models by accounting for both distributional assumptions and heterogeneity in student problem-solving interactions.

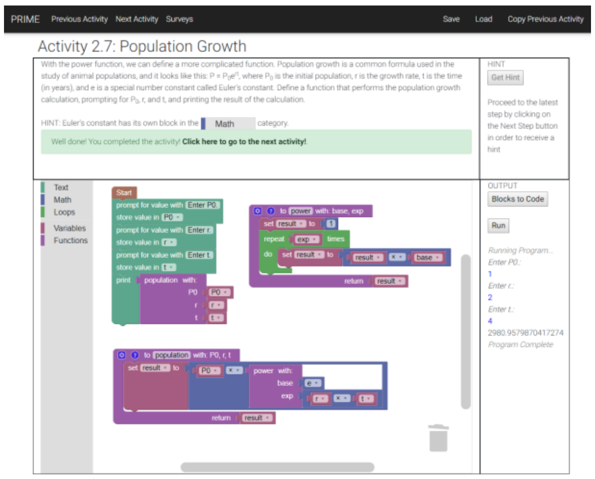
3 Methods

To investigate predictive student modeling for block-based programming environments, we collected programming interaction data of students interacting with a block-based programming environment for introductory computer science education. The programming environment captured interaction data in real time, logging students' programming behaviors as they attempt to solve programming activities. In this section, we describe the block-based programming environment, the programming activities, and the study in which data were collected.

3.1 The PRIME Learning Environment

The analyses conducted in this paper utilizes data collected from PRIME, an adaptive block-based programming environment for undergraduate, non-computer science majors learning introductory programming concepts. The environment currently has over twenty programming activities that each build upon core computer science competencies: input/output, numeric data types, mathematical expressions, variables, iterations (both definite and indefinite), abstraction, functions, parameters, return values, Boolean data types, conditionals, and debugging. The block-based programs in PRIME are created using a customized version of Google's Blockly block-based programming framework (Figure 1). Figure 1 illustrates a completed programming activity in PRIME.

The current study focuses on the first three units of PRIME, which cover a set of topics for introductory undergraduate computer science courses. Unit 1 provides a brief tutorial on the learning environment, covers topics such as basic input/output, numeric data types, math expressions, and definite loops. Unit 2 primarily focuses on functions, parameters, and return variables, and Unit 3 introduces Boolean data types, conditionals, and indefinite loops. Each of these three units consists of a series of short programming activities, and each unit is designed to take approximately 1 hour to complete for a student of average ability. Units 1 and 2 contain seven activities each, and Unit 3 consists of six, for a total of 20 programming activities. The activities build upon concepts and require students to create more complex programs using blocks that introduce more advanced topics.



Activity 2.7: Population Growth

With the power function, we can define a more complicated function. Population growth is a common formula used in the study of animal populations, and it looks like this: $P = P_0e^{rt}$, where P_0 is the initial population, r is the growth rate, t is the time (in years), and e is a special number constant called Euler's constant. Define a function that performs the population growth calculation, prompting for P_0, r, and t, and printing the result of the calculation.

HINT: Euler's constant has its own block in the Math category.

Well done! You completed the activity! **Click here to go to the next activity!**

Proceed to the latest step by clicking on the Next Step button in order to receive a hint

```
Start
prompt for value with Enter P0.
store value in P0
prompt for value with Enter r.
store value in r
prompt for value with Enter t.
store value in t
print population with:
                  P0 P0
                   r r
                   t t

to power with: base, exp
  set result to 1
  repeat exp times
  do set result to result × base
                                return result

to population with: P0, r, t
  set result to P0 × power with:
                          base e
                          exp r × t
                  return result
```

```
Running Program...
Enter P0.:
1
Enter r.:
2
Enter t.:
4
2980.9579870417274
Program Complete
```

Figure 1. Screenshot of the PRIME learning environment.

Students program their solution to each activity through a workspace where they can drag and drop blocks of code. The default workspace consists only of the "Start" block, which serves as the entry point for the program (i.e., analogous to the "main" function in many programming languages). The set of blocks available to students expands as the activities introduce new topics. In addition to utilizing the blocks' functionality, students can run their code, provide input to the program, request hints if they are unable to make progress, and save code to use in a later activity.

3.2 Study Design

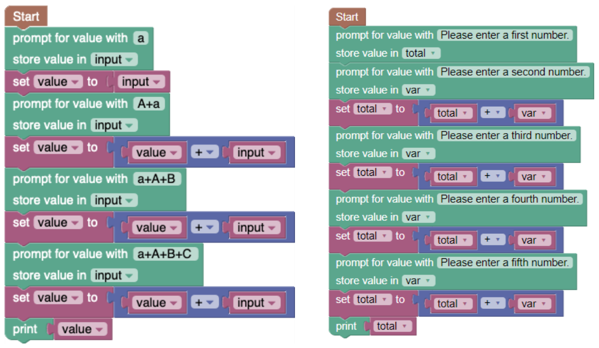
We deployed PRIME at a large public university in the United States to collect student programming interactions. Students in the study were enrolled in one of two sections of an online introductory course required for all engineering students. The students in the study completed a pre- and post-test before and after interacting with PRIME to assess their computer science knowledge. This multiple-choice assessment was validated by three content area experts and demonstrated item-level reliability and appropriate difficulty using IRT analysis. Cronbach's alpha for the pretest was .880 and for the post-test was .896.

A total of 116 students completed both the pre- and post-tests. After removing students with missing data and only including those who had attempted at least one activity, this resulted in a final set of 99 students. A paired samples t-test indicated a significant difference between pre-test score (M = 15.49, SD = 6.39) and post-test score (M = 17.04, SD = 6.52), indicating that students' assessment scores improved after interacting with PRIME (t(99) = 4.62, $p < 0.001$).

Study participants had an average age of 19 with 30.30% of participants reporting their gender as female. Of the 99 students, 67.7% reported their race as White, 16.2% as Asian, 7.1% as Hispanic or Latino, 5.1% as Black or African American, and 4.0% as other. There were a total of 1172 activities attempted by the 99 students (M = 11.58, SD = 6.07), with 769 completed successfully (M = 7.77, SD = 5.74).

Within the 20 activities included in this study, there were a total of 20 distinct blocks, presented in a toolbox, that could be included in solutions. While each of these blocks are not included in every activity, the full set of blocks include: *print*, *text*, *set variable*, *get variable*, *math number*, *math constant*, *math arithmetic*, *prompt*, *repeat loop (definite)*, *function definition (no return)*, *function call (no return)*, *function definition (return)*, *function call (return)*, *logic operation*, *Boolean*, *logic comparison*, *if*, *else if*, *else*, and *repeat loop (indefinite)*.

The PRIME learning environment logs each student's interactions. The system uses expert-designed test cases to detect when the programming activity has been fully completed. Students can progress to the next activity at any time without fully completing the current activity. By using the results of the test cases, we determined the set of activities each student completed, as well as the activities they attempted but did not complete. In Figure 2 below, an example student solution is shown next to an expert-designed solution for one of the activities. This activity, the Accumulator activity, asks students to display the sum of five numbers entered by the user with the use of only two variable blocks. The student solution did not pass the test cases and was incorrect.



```
Start
prompt for value with a
store value in input
set value to input
prompt for value with A+a
store value in input
set value to value + input
prompt for value with a+A+B
store value in input
set value to value + input
prompt for value with a+A+B+C
store value in input
set value to value + input
print value
```

```
Start
prompt for value with Please enter a first number.
store value in total
prompt for value with Please enter a second number.
store value in var
set total to total + var
prompt for value with Please enter a third number.
store value in var
set total to total + var
prompt for value with Please enter a fourth number.
store value in var
set total to total + var
prompt for value with Please enter a fifth number.
store value in var
set total to total + var
print total
```

Figure 2. Student solution (left) and expert solution (right) to the Accumulator activity.

4 Bayesian Linear Models of Student Performance

To predict student performance on a test administered after the programming interactions with PRIME, we constructed linear models using Bayesian Lasso regression. The dataset used in this work was relatively small (99 participants), which makes linear models more appropriate in order to prevent overfitting as well as maintain interpretability of the model parameters. We implemented the Bayesian regression models using double exponential priors on the parameters, which is equivalent to L1 regularization (Lasso regression), serving as a form of variable selection.

In addition to the prior distributions assigned to the model parameters, we varied the distribution chosen to model the response variable, the student's post-test score. Choosing an appropriate distribution to model the response variable is critical, as it can influence the model's predictive accuracy and interpretability. A simple example of this is when making a binary classification. In binary classification, the response variable is desired to be on a scale of 0 to 1, which is why the logistic function is chosen to map the linear combination of input variables and model parameters to this scale. Choosing a different distribution in this case, such as the normal distribution, would result in predictions outside the desired range of 0 to 1.

In this case, the post-test score can only be positive, as it is a count of the number of questions that a student answered correctly out of a fixed number of questions. Thus, distributions designed for count-based data, such as the Poisson and negative binomial distribution, may be more appropriate than the typical normal distribution [21]. However, the Poisson distribution requires the assumption that the mean of the response variable is equal to its variance. The set of post-test scores from this study had a mean of 17.04 and variance of 42.57. This issue, overdispersion, requires a more flexible distribution for the mean and variance to differ.

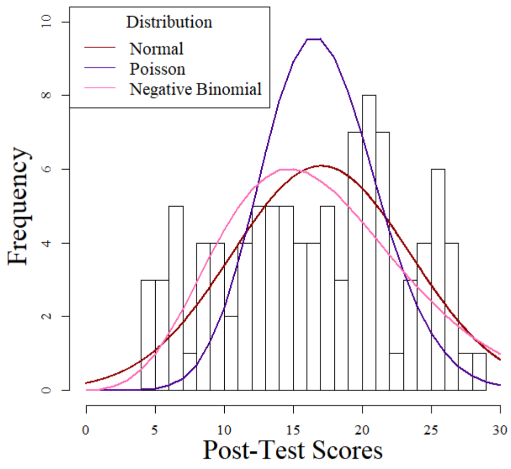


Figure 3. Post-test score histogram with the normal, Poisson, and negative binomial distributions fitted.

To accommodate over-dispersion, one method is to use the negative binomial distribution, which adds flexibility to the existing model by adding a parameter *m*. Utilizing such a distribution is more appropriate for modeling post-test score because it addresses assumptions made by the normal distribution—namely, that post-test score can only be positive. We compare the use of the normal, Poisson, and negative binomial distributions for predicting student post-test scores. Figure 3 shows how well each distribution fits the response variable based on maximum likelihood estimates of each distribution's parameters. This reveals that both the normal and Poisson distributions do not account for the tails of the data nearly as well as the negative binomial distribution does.

In standard linear regression models, another assumption that is made is that the same model applies to all observations in the data. If the data are grouped, such as having multiple observations per student and activity, it is more appropriate to treat observations from the same group as similar. Random effects are a way to describe these group-level characteristics. In terms of modeling students, a linear student model will have a set of fixed effects that describe the relationship between the predictor variables and the response variable, but there are student-level individual differences. Similarly, each programming activity varies in difficulty and ultimately has a different effect on the response variable. In this work, we explore the representation of both student-level and programming activity-level characteristics using random effects and use population-level fixed effects to represent extracted block-based programming features, which we will describe in the next section.

4.1 Representing Student Programs

To represent the student programming solutions, we first extracted the set of blocks used in the set of attempted programming activities from each student. We treated each student-activity attempt pair as an observation in the dataset, for a total of 1146 observations by the 99 students. Of these, 769 were correct solutions and 377 were incorrect/incomplete solutions. After extracting the programs, we represented each observation as the block-wise distance from an expert-designed solution to the given programming activity. The set of expert-designed correct solutions comprised of 20 unique blocks from the PRIME block toolbox. We created a vector for each observation that calculated the difference in the number of each block used between the expert solution and the student's solution. Thus, the feature vector consists of 20 integers, each representing the difference for a specific block. For example, if the expert solution used 5 *print* blocks for a particular activity and the student used 3 *print* blocks for their attempt, this difference would be 2. As different programming activities require different blocks, the distribution of block utilization frequency varies by activity.

As the expert solution to the programming activity only represents one possible solution to the activity, we also included a binary value of whether the student completed the activity, as defined by passing all test cases for that activity. This feature representation aims to capture if the student's code was close (in edit distance) to an expected solution while also accounting for unique or creative solutions different from the expected. To give each feature vector context of a student's prior knowledge, we also use pre-test score as a final feature for a complete feature representation of dimension 22. Since each student completes multiple activities, we ultimately aggregate predictions from each student-activity pair to result in one final prediction per student. To perform this aggregation, we averaged the predictions for each student using each of the activities they attempted.

4.2 Student-Level Random Effects

To capture the characteristics exhibited by each student, we created a random effect for each student in the training set. For each observation in the training set, the feature vectors were also given a unique identifier that ranged between one and the number of students in the training set, *k*, grouping observations by student. During training, the standard linear regression then adds an extra term, θ, that serves as an additional intercept to the model. When evaluating the model on the test set, the regression uses the intercept of the student in the training set with the closest feature vector. To calculate this distance, we used the nearest neighbor in terms of Euclidean distance. By performing the mapping in this manner, the evaluation can be made aware of shared characteristics between different students. The final regression model that predicts student post-test score after each student-activity pair with a student-level random intercept is shown below using the normal distribution as the linking function.

$$Y_{ij} \sim Normal(\alpha + \Sigma_{k=1}^{p} X_{ik}\beta_k + \theta_i, \sigma^2) \qquad (1)$$

Y_{ij} is the post-test score for student i attempting activity *j*. Note that the post-test score will be the same for student *i* across all activities that they attempt, but we make this prediction several times. α is a fixed intercept added to all predictions, X_{ik} (of which there are 22) is the input feature k for student i , and β_k is the parameter learned for feature *k*. Finally, θ_i is the random intercept added corresponding to the specific training set student. In evaluation, the random intercept belonging to the observation closest to the test set observation will be used. σ^2 is the variance. When other distributions are used to model this regression, the usage of the linear combination of features and intercepts will vary to model those distribution's mean and variance.

4.3 Activity-Level Random Effects

In addition to the student-level characteristics that can be modeled to help identify types of students and account for differences in prior knowledge, accounting for the differences in the specific programming activities a student attempted can inform predictive models. In PRIME, there are 20 activities that are ordered in such a way that introduces topics based on common curricula in computer science education. Thus, the activities build upon one another and increase in overall complexity. Without accounting for these differences when predicting the post-test score for each observation, the model treats the features for each activity as fixed effects. We introduce a random intercept for the specific activity that the student attempted for the given observation, γ. This parameter is added to the standard regression formulation to denote differences in difficulty and requirements of each activity. In training, the model learns the random intercept parameter for each of the 20 activities, and in evaluation chooses the corresponding intercept to add to the regression based on the activity of the given observation. Using both the student-level and the activity-level random intercepts, the final regression equation is shown below, again showing the normal distribution link function.

$$Y_{ij} \sim Normal(\alpha + \Sigma_{k=1}^{p} X_{ik}\beta_k + \theta_i + \gamma_j, \sigma^2) \quad (2)$$

This variation of the regression acts exactly the same as the student-level-only random effects model, but now the extra term, γ, is added. During training, the model learns the value of γ for each of the 20 activities. In evaluation, the model adds the random intercept corresponding to the activity of the observation.

5 Results

We compare the predictive accuracy of each model, evaluated using 10-fold student level cross-validation. We perform cross-validation to compare the performance of the standard linear regression (without random effects) for the normal, Poisson, and negative binomial distributions. We then compare the performance of the best fitting fixed effect model (i.e., negative binomial) with the two versions of random intercepts (i.e., student, activity). We report R^2, mean squared error (MSE), and mean absolute error (MAE) averaged across each cross-validation fold. In addition to performing cross-validation, we report the Watanabe-Akaike information criterion (WAIC) [27] for estimating the out-of-sample performance. In selecting the better performing model, the model with a lower WAIC is preferred. WAIC is calculated using the entire dataset (i.e., no folds). This method has the advantage over the more traditional cross-validation by eliminating the requirement of training several models. WAIC is a computationally efficient method that is well suited for Bayesian regression models, and is reported in addition to cross-validation as a convenient alternative [9].

All models in this work are trained using Markov chain Monte Carlo (MCMC) sampling in R using JAGS [18]. All models were checked for convergence using the Gelman-Rubin diagnostic, frequently used for evaluating MCMC convergence [10]. For each fold of the fixed-effect models, we draw 5000 MCMC samples after omitting the first 2000 for burn-in. The burn-in samples are omitted as part of a procedure to ensure the convergence of the Markov chain in MCMC sampling. The final model used to predict the post-test scores of test set observations uses the means of the 5,000 samples for each model parameter. For the models that use random effects, we draw 15,000 MCMC samples after omitting the first 4,000 to allow for better convergence with the higher number of model parameters. Within each model, both the parameters corresponding to the coefficients of the features, β, and the random effects are given prior distributions in the Bayesian model. For each β, we used double exponential priors with mean 0 to function as Lasso priors. By doing this, many of the features in the final model will be 0, as only a select few features will be chosen as significant. For each of the random effects, we chose the normal distribution with mean 0 and a large variance as an uninformative prior. The strength of the prior distributions for all model parameters are uninformative and weak. Therefore, we are allowing the data to have a larger effect on the posterior distributions for all model parameters.

We first compare the predictive performance of using different distributions to model student post-test scores. Next, we compare the addition of random effects to the model with the best performance—the negative binomial distribution.

5.1 Comparison of Response Variable Distribution

The first comparison shows the difference in predictive performance when different distributions are chosen to model the response variable, post-test score. Table 1 displays the performance of each model.

Table 1. Comparison of predictive performance of the normal, Poisson, and negative binomial distributions.

Regression Model	R^2	MSE	MAE	WAIC
Normal	0.618	13.013	2.981	6491.815
Poisson	0.639	12.813	2.879	**6306.875**
Negative Binomial	**0.641**	**12.696**	**2.877**	6348.865

As shown above, the negative binomial distribution best models the post-test score. The regressions using the negative binomial and Poisson distributions fit the data better than the regression using the normal distribution, though many of the metrics for the negative binomial distribution are only marginally different from the Poisson distribution. We select the negative binomial as the better choice due to the increased model flexibility, as the Poisson makes the assumption that the mean and variance are equal.

5.2 Hierarchical Bayesian Modeling with the Negative Binomial Distribution

The second comparison shows the difference in predictive performance when random effects (RE) are added to the base model. In this case, we chose the base model to be the Bayesian Lasso regression with the negative binomial (NB) distribution. Table 2 shows the performance of each model compared to the base regression model.

Table 2. Comparison of predictive performance of the negative binomial (NB) regression, NB + student-level RE, and NB + student-level + activity-level RE.

Regression Model	R^2	MSE	MAE	WAIC
NB	0.641	12.696	2.877	6348.865
NB+Student RE	0.681	11.608	**2.717**	**5606.372**
NB+Student+Activity RE	**0.683**	**11.585**	2.725	5624.745

As shown in the table, the NB+Student RE model significantly outperforms the baseline NB model across all metrics. However, there is very little difference in the performance between the student-level random effects model and the model that additionally incorporates the activity-level random effects. We will use the student-level random effects model as the best performing model moving forward because it uses less parameters overall and predicts post-test performance with near equal accuracy, and it is therefore more parsimonious.

5.3 Variability of Student-Level Random Effects

Bayesian linear models offer the advantage of interpretability relative to more complex models. Specifically, the MCMC samples produce a distribution for each parameter that can be used to estimate the uncertainty of the model for how useful each feature is. Additionally, the Bayesian Lasso coefficients produce a confidence interval, which can help in variable selection. The best performing model was the student-level random effects regression. Figure 4 shows the variability of the random effects for each student, emphasizing the usefulness of incorporating these student-level characteristics for student modeling. This figure highlights the differences in students, where the random intercept added to the regression equations varies widely for each student. The blue line represents the confidence of this effect. When evaluating the predictive model using these trained random effects, the nearest neighbors indexing selects the random intercept of the closest feature vector within the training set to the new observation. As this figure shows, selecting the most similar random effect is a crucial step in calculating the final regression output.

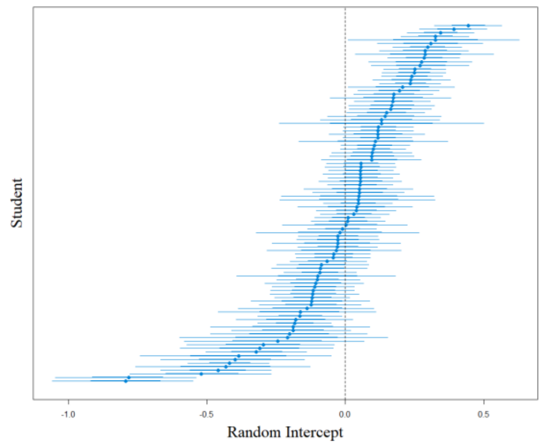


Figure 4. Student-level random intercept values.

6 Discussion

The evaluation revealed that predictive student models incorporating student-level characteristics improved model accuracy. After evaluating Bayesian regression models that used the normal, Poisson, and negative binomial distributions, we found that the regression using the negative binomial distribution best predicted student post-test score. Additionally, the modeling assumptions made using the negative binomial distribution were upheld compared to the assumptions of the other distributions. Adding random effects to the negative binomial Bayesian regression model was found to significantly increase predictive performance. Adding student-level random effects yielded a significant performance increase, while the addition of activity-level random effects had only a modest positive impact. Using non-normal distributions and hierarchical modeling offer the potential to improve student models so that they can more effectively and proactively support struggling students.

The Bayesian framework presented in this work for modeling performance in block-based programing has several advantages. First, by making a prediction after each time a student attempts an activity, it is possible to handle varying amounts of data for different students. Some students only attempt a few activities, and some students attempt all activities. By averaging the predictions made after each student-activity pair, the model can handle both of these situations. Second, the addition of student-level random effects in the hierarchical model enables the incorporation of student-specific characteristics when making predictions. The nearest-neighbor approach when extracting the random effects in predictions relies on the training set having seen a vast amount of types of students to ensure that the correct random effect is chosen in evaluation. Third, the Bayesian linear model with Lasso priors enables confidence intervals to be produced on the variables that are not close to 0. By doing this, we are able to infer which variables are important in this prediction. Ultimately, the model strongly relied on pre-test scores and applied very small coefficients to all other fixed variables.

The improvement in performance of the regression models using the negative binomial and Poisson distributions is not surprising, as the approximate distribution of the post-test score does not follow the normal distribution. Specifically, the standard errors in the standard normal regression are calculated under the assumptions that the residuals are independently and identically distributed. When evaluating these models, the prediction intervals are calculated assuming the residuals are normally distributed. If this is not the case, the prediction intervals may not be accurate.

A surprising result is that the activity-level random effects do not improve predictions over the student-level random effects. Activity-level random effects were hypothesized to be useful due to the varying level of difficulty in the activities, implying performance on easy activities is less informative than performance on harder programming activities. However, it is possible there is limited data for all activities to become of use. Specifically, fewer students attempt the later activities. The later activities, which are more difficult and perhaps more revealing about the student's computer science skill, are attempted far less frequently and thus the random effects for these activities cannot be distinguished from those of earlier activities. Incorporating additional data will likely help with this phenomenon.

7 Limitations

A key challenge was determining how to aggregate predictions for each student in evaluation. Since the model makes predictions per every student-activity pair, the predictions must be aggregated for every student. We chose to average these results, i.e., each activity that a student attempts contributes equally to the overall prediction. A non-uniform weighting of the activities could further improve accuracy.

A second limitation in this work is the way in which the nearest neighbor calculation is performed. For choosing the student-level random effect in evaluation, we found the closest vector in the training set to the test observation. This method assumes that each feature in the observation should be treated equally. However, we found that certain features were more predictive overall, which implies that weighting those features in this nearest neighbor calculation could improve the overall indexing of random effects. Another potential problem occurs if the test observation is unlike any observation in the training set. In this case, the random effect chosen will refer to a student that is perhaps extremely different from the students in the test observation. One approach would be to create a threshold distance between the test and training observations. If no training observation falls within this distance threshold, then the model could use a marginal random effect as a default.

Another limitation is the large number of student-level random effects. The model creates a random effect per student in the training set, which ultimately varies depending on how much data is available. If expert prior information were available to describe types of students, fewer random effects could be used, reducing the number of parameters in the final model. This could reduce overall model complexity and lead to better interpretability with respect to which types of students benefit from particular programming activities.

8 Conclusion

As computer science education continues to evolve and utilize block-based programming as a teaching tool for introductory concepts, learning environments that tailor the experiences of students based on student-level characteristics have significant potential. However, predictive models that do incorporate these student-level characteristics often make a number of simplifying assumptions that can negatively affect model performance.

To address these issues, we created a Bayesian linear regression framework that models the response variable distribution using more appropriate distributions, and we created random effects to account for both student-level and programming activity-level characteristics when making post-test score predictions. We found that models that used the negative binomial distribution outperformed both the Poisson and normal distribution-based models. Additionally, models that incorporated student-level characteristics outperformed models that did not. We also added activity-level random effects to the hierarchical model, but this did not significantly improve results, suggesting that the better model uses student-level characteristics only. This indicates that student models that treat types of students differently in making predictions can improve student modeling.

In future work it will be instructive to investigate different feature representations for student programs. Instead of comparing student solutions to expert-designed block-based programs, other encodings such as comparing abstract syntax trees, or distributed representations created from large data sets using deep learning [29] could be used. Another promising direction for future work is to investigate alternative representations for student-level characteristics, which may further increase the predictive accuracy of student models. Finally, it will be important to integrate these models into the learning environment, to better understand how they can be used to drive adaptive support and improve the effectiveness and efficiency of the learning experience for novice computer science students.

ACKNOWLEDGMENTS

This research was supported by the National Science Foundation under Grants DUE-1626235 and DUE-1625908. 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.

REFERENCES

[1] Akram, B., Min, W., Wiebe, E., Mott, B., Boyer, K.E. and Lester, J. 2019. Assessing Middle School Students' Computational Thinking Through Programming Trajectory Analysis. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 1269–1269.

[2] Akram, B., Min, W., Wiebe, E., Mott, B., Boyer, K.E. 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] Arthurs, N., Stenhaug, B., Karayev, S. and Piech, C. 2019. Grades are Not Normal: Improving Exam Score Models Using the Logit-Normal Distribution. In *Proceedings of the 12th International Conference on Educational Data Mining*, 252–257.

[4] Bakker, B. and Heskes, T. 2004. Task Clustering and Gating for Bayesian Multitask Learning. *Journal of Machine Learning Research*, 4(1), 83–99.

[5] Blikstein, P. 2011. Using Learning Analytics to Assess Students' Behavior in Open-Ended Programming Tasks. In *Proceedings of the International Conference on Learning Analytics and Knowledge*, 110–116.

[6] Corbett, A.T. and Anderson, J.R. 1994. Knowledge Tracing: Modeling the Acquisition of Procedural Knowledge. *User Modelling and User-Adapted Interaction*, 4, 253–278.

[7] Desmarais, M.C. and Baker, R.S.J.D. 2011. A Review of Recent Advances in Learner and Skill Modeling in Intelligent Learning Environments. *User Modeling and User-Adapted Interaction*, 22(1–2), 9–38.

[8] Effenberger, T. and Pelánek, R. 2018. Towards Making Block-Based Programming Activities Adaptive. In *Proceedings of the 5th Annual ACM Conference on Learning at Scale*, 6–9.

[9] Gelman, A., Hwang, J. and Vehtari, A. 2014. Understanding Predictive Information Criteria for Bayesian models. *Statistics and Computing*, 24(6), 997–1016.

[10] Gelman, A. and Rubin, D.B. 1992. Inference from Iterative Simulation using Multiple Sequences. *Statistical Science*, 7, 457–511.

[11] 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.

[12] Huang, Y., Xu, Y. and Brusilovsky, P. 2014. Doing More with Less: Student Modeling and Performance Prediction with Reduced Content Models. In *Proceedings of the 22nd International Conference on User Modeling, Adaptation and Personalization*, 338–349.

[13] Khajah, M., Lindsey, R. V. and Mozer, M.C. 2016. How Deep is Knowledge Tracing? In *Proceedings of the 9th International Conference on Educational Data Mining*, 94–101.

[14] Mao, Y., Zhi, R., Khoshnevisan, F., Price, T.W., Barnes, T. and Chi, M. 2019. One Minute is Enough: Early Prediction of Student Success and Event-level Difficulty during a Novice Programming Task. In *Proceedings of the International Conference on Educational Data Mining*, 119–128.

[15] Min, W., Frankosky, M., Mott, B.W., Rowe, J., Smith, P.A.M., Wiebe, E., Boyer, K. and Lester, J. 2019. DeepStealth: Game-Based Learning Stealth Assessment with Deep Neural Networks. *IEEE Transactions on Learning Technologies*.

[16] Mislevy, R.J., Behrens, J.T., Dicerbo, K.E. and Levy, R. 2012. Design and Discovery in Educational Assessment: Evidence-Centered Design, Psychometrics, and Educational Data Mining. *Journal of Educational Data Mining*, 4(1), 11–48.

[17] Piech, C., Bassen, J., Huang, J., Ganguli, S., Sahami, M., Guibas, L. and Sohl-Dickstein, J. 2015. Deep Knowledge Tracing. In *Advances in Neural Information Processing Systems*, 505–513.

[18] Plummer, M. 2003. DSC 2003 Working Papers JAGS: A Program for Analysis of Bayesian Graphical Models using Gibbs Sampling. In *Proceedings of the 3rd International Conference on Distributed Statistical Computing*, 1–10.

[19] Price, T.W. and Barnes, T. 2017. Position Paper: Block-Based Programming Should Offer Intelligent Support for Learners. In *Proceedings of the 2017 IEEE Blocks and Beyond Workshop*, 65–68.

[20] Price, T.W., Dong, Y. and Lipovac, D. 2017. iSnap: Towards Intelligent Tutoring in Novice Programming Environments. In *Proceedings of the Forty-Eighth ACM Symposium on Computer Science Education*, 483–488.

[21] Reich, B.J. and Ghosh, S.K. 2019. *Bayesian Statistical Methods*. CRC Press.

[22] Rivers, K., Harpstead, E. and Koedinger, K. 2016. Learning Curve Analysis for Programming. In *Proceedings of the International Conference on Computing Education Research*, 143–151.

[23] Rupp, A., Levy, R., Dicerbo, K.E., Sweet, S.J., Crawford, A. V., Calico, T., Benson, M., Fay, D., Kunze, K.L., Mislevy, R.J. and Behrens, 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.

[24] Sao Pedro, M.A., De Baker, R.S.J., Gobert, J.D., Montalvo, O. and Nakama, A. 2013. Leveraging Machine-learned Detectors of Systematic Inquiry Behavior to Estimate and Predict Transfer of Inquiry Skill. *User Modelling and User-Adapted Interaction*, 23(1), 1–39.

[25] Sawyer, R., Rowe, J., Azevedo, R. and Lester, J. 2018. Modeling Player Engagement with Bayesian Hierarchical Models. In *Proceedings of the 14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 215–221.

[26] Shute, V.J. 2011. Stealth Assessment in Computer-Based Games to Support Learning. *Computer Games and Instruction*, 503–524.

[27] Watanabe, S. 2010. Asymptotic Equivalence of Bayes Cross Validation and Widely Applicable Information Criterion in Singular Learning Theory. *Journal of Machine Learning Research*, 11, 3571–3594.

[28] Weintrop, D. and Wilensky, U. 2017. Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Transactions on Computing Education*, 18(1), 1–25.

[29] Wu, M., Mosse, M., Goodman, N. and Piech, C. 2018. Zero Shot Learning for Code Education: Rubric Sampling with Deep Learning Inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 782–790.

[30] Xie, B. and Abelson, H. 2016. Skill Progression in MIT App Inventor. In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing*, 213–217.

[31] Yudelson, M. V., Koedinger, K.R. and Gordon, G.J. 2013. Individualized Bayesian Knowledge Tracing Models. In *Proceedings of the International Conference on Artificial Intelligence in Education*, 171–180.

[32] Yudelson, M. V., Medvedeva, O.P. and Crowley, R.S. 2008. A Multifactor Approach to Student Model Evaluation. *User Modeling and User-Adapted Interaction*, 18(4), 349–382.

[33] Zhang, L., Xiong, X., Zhao, S., Botelho, A. and Heffernan, N.T. 2017. Incorporating Rich Features into Deep Knowledge Tracing. In *Proceedings of the 4th (2017) ACM Conference on Learning at Scale*, 169–172.