Programming Behavior Detection 1.1

Final Report

Fall 2018

Wei Han

Joe Huang

Table of Contents

1. Abstract

This report describes our contributions to the programming behavior detection to assist the Intelligent Tutoring System in the Social Additive Gameful Engineering (SAGE) project this semester. We have been concentrating on the Programming Behavior Detection 1.1, #290 sage-scratch integration, in addition to the new parameters added to the behavior detection model. Currently, the programming behavior detection API allows SAGE Node to pass in student data as inputs and receive the student behavior type as output.

2. Introduction

The Social Addictive Gameful Engineering (SAGE) research project is a system utilizing Scratch, a visual programming language, to infuse computational thinking in 6-8 grade students through puzzles and exercises. And one of the important components in SAGE is Intelligent Hinting, which generates hints based on student actions automatically and/or on-demand. Other than generating hints based on student learning paths [6], Intelligent Hinting also takes the student behavior types into consideration. Therefore, providing a functional API that analyzes and outputs the student behavior types would be helpful to improve the current system.

From previous semesters, many works had been done in Behavior Detection. However, most of the works are separated and not fully integrated together. To make Behavior Detection an easy-to-use API, we adopt the methodologies from previous works and develop the API under one module. Moreover, we have reduced the redundant data transformation and allow simplified data flow in the system to eliminate the use of local storage. We also take the previous student behaviors into factors when deciding the current student behavior. Finally, we leverage the RESFful API in Intelligent Hinting system [6] to allow SAGE Node to make use of our functionality.

In this report, we will be mainly discussing about the architecture of Behavior Detection and the details of our implementation. In the end, we will explore the current limitation and possible future works to improve the Behavior Detection and SAGE system.

3. Related Works

3.1. "Conditions of Learning in Novice Programmers"[4]

Perkins et al. discusses its clinical investigation on different learning performance of novice programmers and classifies their behaviors into mover, stopper, extreme mover, and tinker. The main difference between those behavior types lies in students' different patterns of learning in the programming context. Specifically, stoppers generally have difficulty coming to an answer and are unwilling to proceed; movers tend to consistently testing new ideas and do not appear stuck for too long; extreme movers incline to make moves too quickly with little consideration on the correctness of their answers; tinkers are ones who write down some codes first and slowly making changes. In addition, this paper suggests that students could benefit from the type of instructions which corresponds to their behavior type, for better learning practices.

3.2. "A Framework for Capturing Distinguishing User Interaction Behaviors in Novel Interfaces"[5]

S. Kardan and C. Conati implement a user modeling framework that analyzes learner behaviors and the relationship between learning and different types of behaviors. They use unsupervised clustering and 'Class Associating Mining' to recognize different interaction patterns between student and educational software and successfully cluster students into meaningful groups. In addition, their framework provides descriptions of common behavior shared within the same group and allows education software to help the students to improve given their group.

4. Architecture

In this section, we will demonstrate the changes in the data flow of Behavior Detection API. From previous semesters, behavior detection was implemented by a combination of files from different modules. For example, the ReadFile.java, which outputs the statistical analysis of the snapshots, is located under Scratch Analyzer written in Java, while the machine learning model is implemented in Python. Moreover, the data formats varied; from Figure 1, the readFile.java requires snapshots to be transformed from json to .se files and saves locally before processing them. This complicates the data flow of API and requires the use of local storage, which limits the scalability of the program.

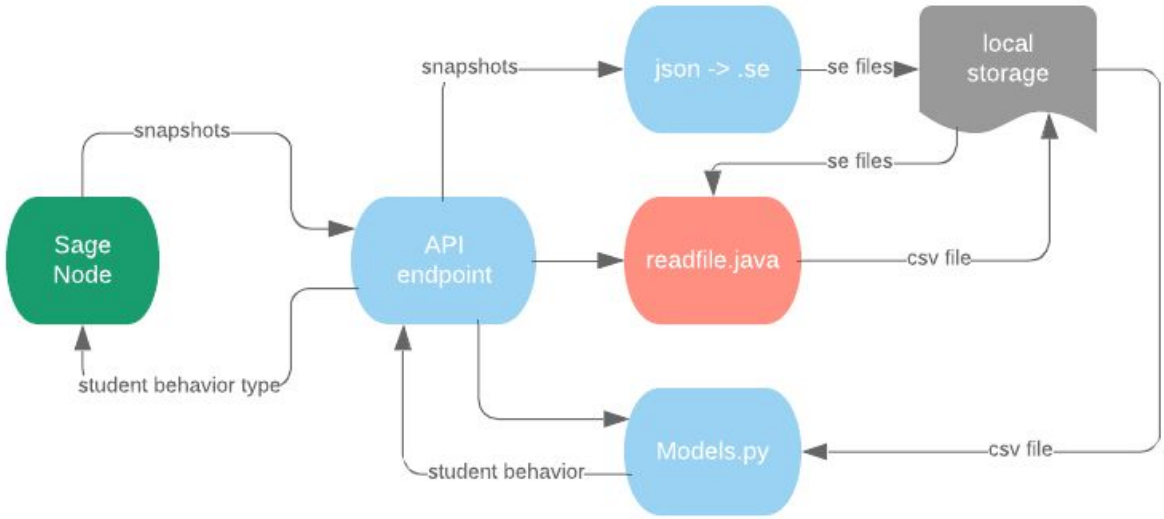

Figure 1: the API flow utilized in previous semesters

To simplify the data flow and increase scalability of the application, we have integrated all the parts of behavior detection under the same module and minimized local data storage.

Specifically, as snapshots in json format are passed from the SageNode API endpoints, the machine learning model utilizes the helper function to get the statistical analysis on snapshots and predicts the student behavior type using the trained models, in addition to the student's previous data.

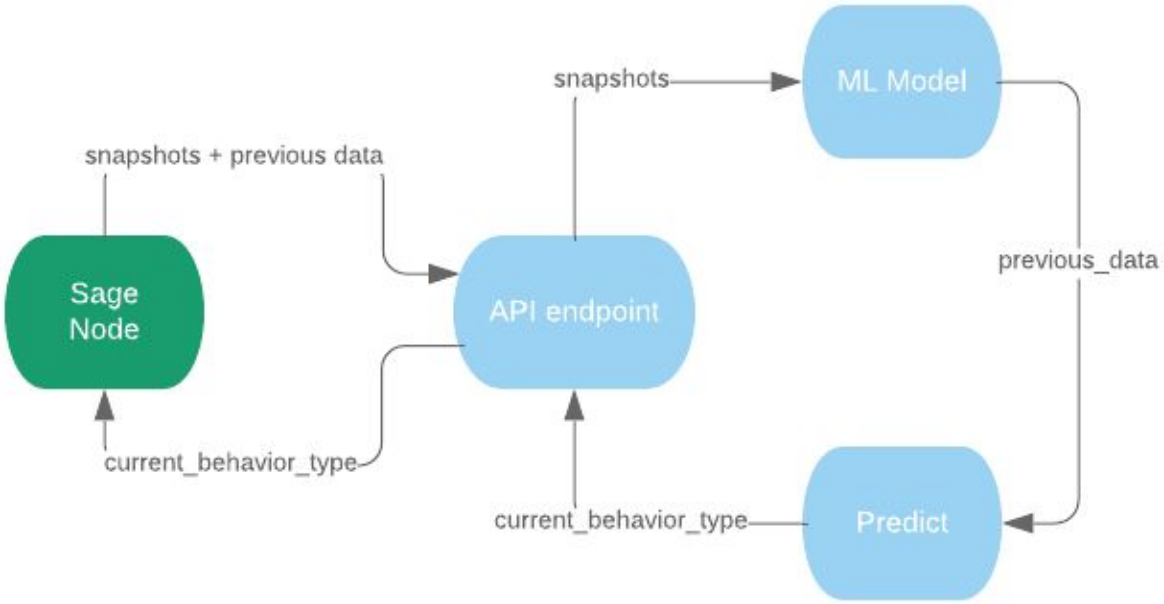

Figure 2: the simplified API flow

5. Implementation

In this section, we will discuss the details of mathematical formulation and code implementation in each project aspect.

5.1. Code Repositories

Project Aspect	**Repository**	**Branch name**
Mock Data Generation	/cu-sage/scratch-analyzer/intelligentHint/behaviorDetection	bd_test
Input Data Format		
Classification Models		
Previous Student Data		
API Endpoints	/cu-sage/scratch-analyzer/intelligentHint/server_interface	

5.2. Mock Data Generation

We inherited the mock data generation method from the previous semester[2]. Specifically, we separate the student behavior types into mover, stopper, extreme mover, and tinker. We generated 50 mock data projects for every student behavior types to train the machine learning models. The generating file is located in mockdataBD.py.

- **Movers**: students who consistently make attempts without pausing for too long. We set the time interval between actions to a Gaussian distribution with a mean of 3 seconds and a standard deviation of 1 second, and only allow the student to add blocks.

- **Stoppers**: students who spend more time at each step compared to other students. We set the time interval between actions to a Gaussian distribution with a mean of 10 seconds and a standard deviation of 3 seconds, and only allow the student to add blocks.
- **Extreme Movers**: students who make attempt quickly without many thoughts. We set the time interval between actions to a Gaussian distribution with a mean of 1 second and standard deviation of 1 second, and allow the student to add and delete blocks.
- **Tinkers**: students who attempt to change their works by making small edits. We set the time interval between actions to a Gaussian distribution with a mean of 3 seconds and a standard deviation of 1 second, and allow the student to add and delete blocks.

5.3. Input Data Format

From the previous behavior detection API, the student snapshots were captured at a fixed time frequency even when no change is made, which created multiple identical snapshots that do not provide us with more useful information. Therefore, to decrease the number of unnecessary snapshots, the midterm report [3] suggests recording the snapshot only when a student action is taken. By utilizing the timestamps, we could calculate the time differences between snapshots and utilize them as features in machine learning training. As shown in Figure 3, every snapshot contains the timestamp and the content.

```
"snapshots": [
    {
        "timestamp": "18-40-28-GMT-1204-2018",
        "content": "<<Object Stage>>\n"
    },
    {
        "timestamp": "18-40-30-GMT-1204-2018",
        "content": "<<Object Stage>>\n\t\twhenGreenFlag\n"
    },
    {
        "timestamp": "18-40-31-GMT-1204-2018",
        "content": "<<Object Stage>>\n\t\twhenGreenFlag\n\t\tstartScene\n"
    },
```


Figure 3: an example of snapshots

In addition to the timestamped snapshots, we also included the previously analyzed student data into consideration in determining the current student behavior. For example, if a student had been a mover in the past 10 classifications and the recent classification shows that he is a stopper, this behavior might be temporary and we should take the previous data into consideration. Also, we give more weights to the behavior type of games that are the same type as the current game since they are more relevant in terms of game content. Finally, for each previous student data, we collect both the game type and the behavior type at each timestamp, shown in Figure 4.


```
"19-55-11-GMT-1207-2018": [
    0,
    1
],
"19-55-11-GMT-1205-2018": [
    3,
    6
],
```

Figure 4: an example of previous student data. The first number represents the behavior type and the second represents the game type.

5.4. Classification Model

In this section, we will discuss the feature extraction and machine learning models used for student behavior classification based on snapshots.

5.4.1. Feature Extraction

Given the timestamps and current student snapshots, we built a tree for each snapshot based on the hierarchy of blocks. Then we compared consecutive snapshot trees and keep track of the number of changes as well as the time difference between snapshots. Finally, we calculated the mean and standard deviation of the time difference, number of change per block, and number of changes per interval, respectively, and use them as features for the machine learning model. The feature extraction is implemented in readFile.py and the tree structure class in Tree.py.

5.4.2. Model Training

By using the mean and standard deviation data from feature extraction, we build supervised machine learning models, including Decision Tree Classifier, K-nearest Neighbor Classifier, Support Vector Machine, Multilayer perceptron, Random Forest, and Linear regression, to correctly predict student behavior type. To ensemble all the models, we use a voting classifier to take all the outputs from each model and get the average probability for each behavior type. However, having more models does not guarantee a better prediction. According to the testing results of 10-fold cross-validations with each model, the better performing models are Multilayer Perceptron, Random Forest, and Linear Regression, and rhe accuracy of Voting Classifier using those better performing models reaches around 93%.

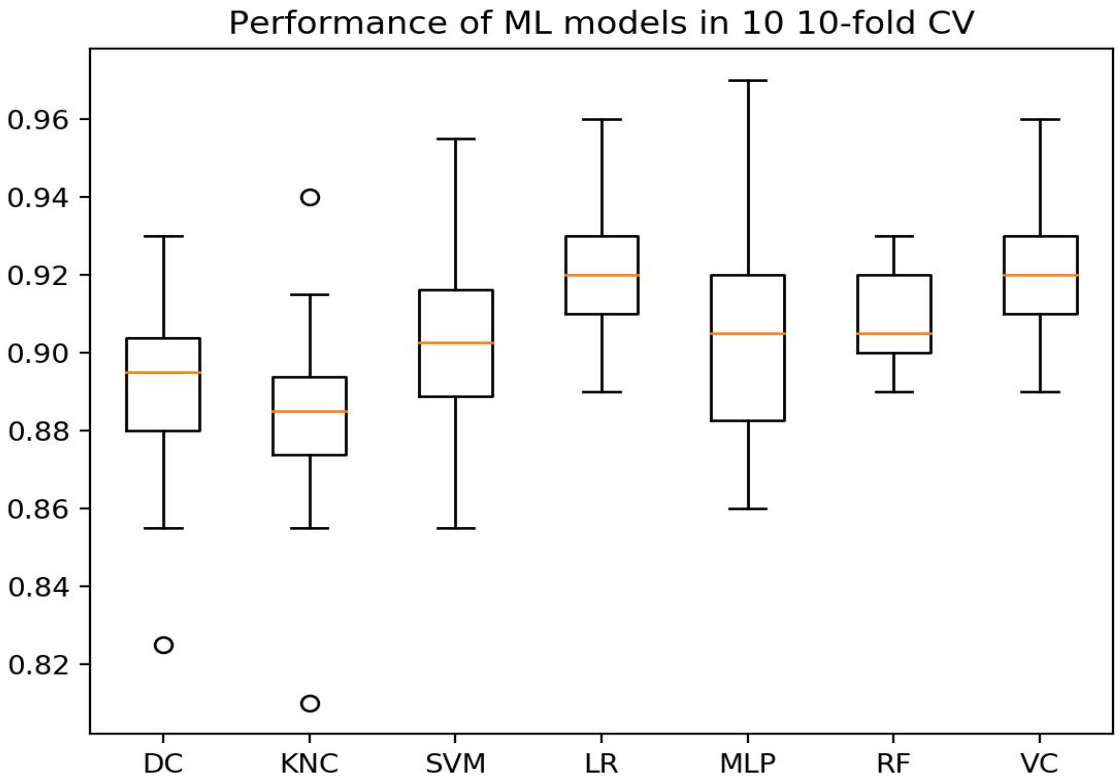

Figure 5: performance of ML models with 10 times 10-fold Cross-Validation.

Even though the accuracies for Decision Tree Classifier, K-nearest Neighbor Classifier, and Support Vector Machine are not as preferable, we think that they could possibly have better results once the real student data is available and should be kept for future training. Finally, the trained model is saved locally as 'save_model.pkl' in the Behavior Detection folder. The Linear Discriminant Analysis used for dimension reduction is saved in the pickle file as 'lda' and the Voting Classifier as 'vc'. An example of model prediction would be [0.7, 0.5, 0.4, 0.64], which number in each index represent the probability of each behavior type.

5.5. Integrating previous student data and behavior classification results

5.5.1. Previous student data

Previous data contains 1) the previous behavior type, and 2) the previous game type. Further, each contributes different weights according to the time difference between its timestamp and current timestamp, recent behavior type is considered more relevant to the current estimation and thus weighted more. In the current model, we have chosen to model weight distributions in an exponential decay function, in which -*ln*(0.01) serves as the maximum value for x range in which the exponential decay function value is considered effective / nonzero, since $e^{-x} = 0.01$. The cutoff in the denominator indicates the cutoff parameter, default to 20 days, such that too obsolete data is tossed out. The time difference simply denotes the time difference in unix time between the previous to current timestamps.

$$w_i = e^{\frac{ln(0.01) \times \text{time\_diff[i]}}{\text{cutoff}}}$$

In addition, a simple probability model of $P(behavior|game)$ is constructed based on the student's previous data. As the data might be overfitting the training model when the training sample is small, additive smoothing is added to the model; here, k is a constant (default set to 0.5) and in the denominator is multiplied by 4, which is the number of behavior types.

$$P(b|g) = \frac{P(b,g)+k}{P(g)+4k}$$

5.5.2. Integration

From the machine learning model, we obtain the probability of each student behavior types. In addition to the probability of behavior type given the game type and the weights from previous

student data, we then calculate a score for each student behavior types using the formula below. The API returns the one with the highest score.

$$P(b_x) = P_{model}(b_x)P(b_x|g)\sum_{b_i=b_x} w_i, x \in \text{Number of behavior types}$$

The integration is implemented in Models.py.

5.6. API Endpoint

This section shows the details of API usage and example of an API call. We integrate the behavior detection API within the Intelligent Hinting RESTFul API. To stimulate the API call, we use Postman with sample mock data, as shown in Figure 6.

API	**Input & Output**	**Notes**

POST /get_behavior_type	input:
	{
	“prev_data”:{
	“timestamp: string”:{
	behavior_type: int,
	game_type: int
	} … },
	current_game: int,
	“snapshots”:{{
	timestamp: string,
		The API takes in a list of previous student data, the current game type, and a list of snapshots, return the current student behavior type.

	content: string } … } } output: behavior type: int	

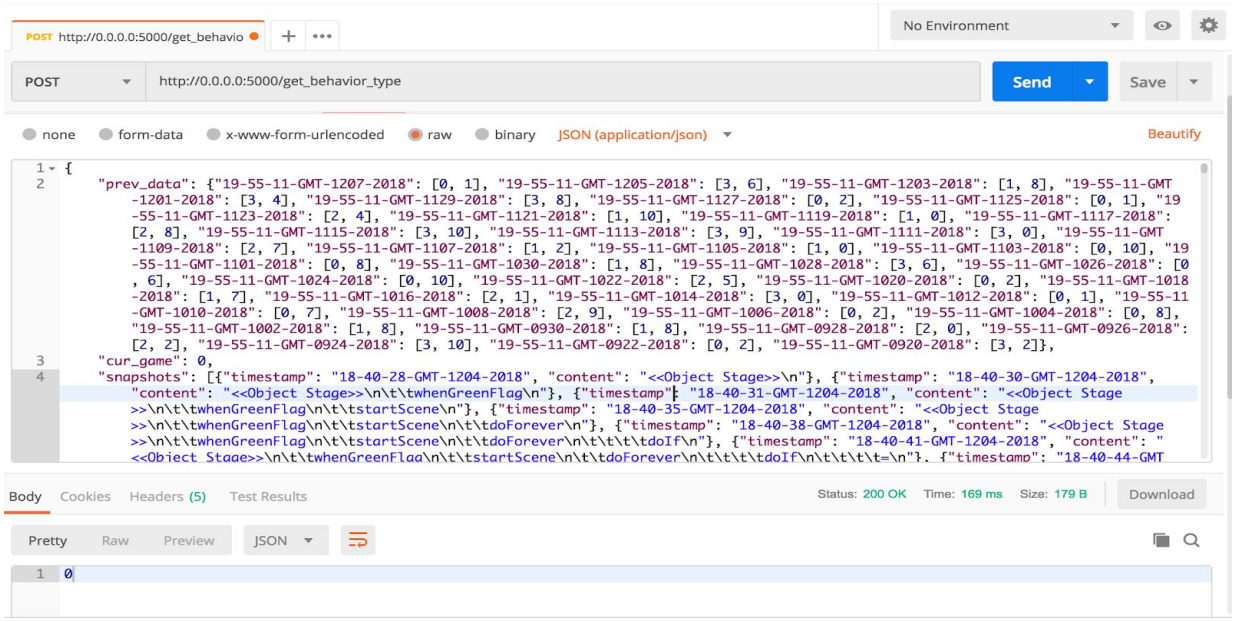

Figure 6: an example of an API call. The json texts represent the API input and the ‘0’ represents the behavior type, in this case, a mover.

6. Assumption and Limitation

6.1. Lack of Real Student Data

Currently, all the models are trained using the mock data, which is self-defined under our assumptions and does not necessarily reflect the real student data distribution. This could result in the model misclassifying the real student behaviors. However, given enough real data, the model could be re-trained to accurately make predictions. Also, the number of student behavior types is fixed to allow Intelligent Hinting to be trained with consistency. However, the number of possible student behavior types could be different from games to games and require further research.

6.2. Probability Model and function parameters

Our probability model for behavior type given game types assumes the cut off days to 20 and set their lower limits to around 0.01. However, this is our default function parameters and should be changed based on weights for game types. Also, more hyperparameter tuning could be done to improve the current machine learning models.

.

7. Future Works

7.1. More Variants in Mock Data Generation

The mock data is currently generated by adding and deleting blocks and cumulatively build to the solution. Since the behavior detection model focuses more on the time interval between actions and the number of blocks changed, this implementation is enough for us to train the machine learning model. However, more variance could be considered to generate more realistic data, such as adding blocks at different timestamps.

7.2. Unique ID for Each Block

When comparing two snapshots, we identify a block by its name and determine if a block has been removed or added. Blocks with the same name could be deleted and added again between snapshots, but our current method won't be able to detect such change. The report [1] proposed and implemented a unique ID for each block, which could be better identification of block.

7.3. Integration with SAGE Node

Currently, there is no connection between the SAGE Node and Behavior Detection API. Specifically, the SAGE Node should decide the frequency of the API calls, which could depend on how many snapshots not used by the API yet and update in mLab accordingly. For example, the SAGE Node could call the Behavior Detection API once there are 50 unused snapshots and update the mLab by marking those snapshots as used and adding the new behavior type.

7.4. Attempts in Supervised Learning

Since the four student behavior types have been defined, those parameters could perhaps be of use in behavior classification, other than simply generating mock data. Attempts from all

previous semesters have been focusing on clusterings, supervised learning using these parameters might produce better results

8. References

[1] Anand, Sambhav & Sawyer, Allison, "Enhanced Data Collections, Student Progress Modeling, and Intelligent Hint in SAGE" Spring, 2017 https://github.com/cu-sage/Documents/blob/master/2017_1_Spring/final_report_SAGE_ml.pdf

[2] Bian, Chengwei & Zhang, Mengqiao, "Intelligent Hinting and Behavior Detection in SAGE Final Report." Fall, 2017 https://github.com/cu-sage/Documents/blob/master/2017_2_Fall/final_report_SAGE_git.pdf

[3] Dziena, Alex & Li, Lily Yu, "SAGE DevOps, Gameful Intelligent Tutoring, and Publication midterm report" Fall, 2018 https://github.com/cu-sage/Documents/blob/master/2018_3_Fall/SAGE%20DevOps%2C%20Gameful%20Intelligent%20Tutoring%2C%20and%20Publication%20Midterm%20Report.pdf

[4] Perkins, David N., et al. "Conditions of learning in novice programmers." *Journal of Educational Computing Research* 2.1 (1986) 37-55.

[5] S. Kardan, C. Conati. "A Framework for Capturing distinguishing User Interaction Behaviours in Novel Interfaces" Department of Computer Science, University of British Columbia

[6] Ding, Y., Luo, W., & Zhang, J. "Intelligent Hinting 1.1 Final Report" Spring, 2018. https://github.com/cu-sage/Documents/blob/master/2018_1_Spring/Final_SAGE_git_intelligent_Hinting_1.1.pdf