

Intelligent Hinting and Behavior Detection in SAGE

Final Report

Chengwei Bian and Mengqiao Zhang

COMS E6901, Section 14

December 23rd, 2017

Table of Contents

1 Introduction	2
2 Related Work	3
2.1 “Conditions of Learning in Novice Programmers”	3
2.2 “Early Prediction of Student Frustration”	3
2.3 “Coarse-Grained Detection of Student Frustration in an Introductory Programming Course”	4
2.4 “Advances in Intelligent Tutoring Systems: Problem-solving Modes and Model of Hints”	5
2.5 “Developing a Generalizable Detector of When Students Game the System”	6
3 Motivation	7
4 Implementation	9
4.1 Code Repositories	9
4.2 Generate Mock Data	10
4.3 Feature Extraction	13
4.4 Build Behavior Classification Models	15
4.5 Adjust the Hinting System	16
5 Future work	18
5.1 Obtain Real Student Data	18
5.2 Improve the Behavior Detection Integration	18
5.3 On-demand Hinting	19
5.4 Multi-layer Hints	19
5.5 Avoid Gaming the System	20
6 References	21

1 Introduction

The education for students is a widely studied and controversial topic. Researchers around the world keep trying to improve and perfect the teaching approach, which not only allow students to learn knowledge, but also feel enjoyable when they are learning. One of the main goals of educational research is the development of appropriate teaching formats for computer science and computational thinking concepts, since these basic concepts are often challenging for students to learn in general classrooms.

SAGE is a system designed to immerse students in an enjoyable, game-like learning experience [1]. It offers adaptable features and feedback in order to maximize engagement and minimize the risk of negative emotions including boredom, frustration, and confusion. The intelligent tutoring system in SAGE is designed to provide immediate and customized instruction or feedback to learners, usually without requiring intervention from a human teacher.

Our primary motivation for this part of the project is to improve the effectiveness of the existing hinting system in Social Addictive Gameful Engineering (SAGE) by adding more functionalities. So in general, we built a pipeline to create mock data and 24 features we need to train our machine learning model. And then built the machine learning model which would be able to classify the students into 4 different behaviors (mover, extremely mover, stopper and tinkerer). And finally, based on what kind of the user behaviour is, we adjust different hinting frequency for each user behaviour in order to prevent frustration and improve the user experience.

2 Related Work

In this section we will briefly discuss some of the papers that help our project and also the ones that might be useful for the future work.

2.1 “Conditions of Learning in Novice Programmers” [2]

Perkins and other educators conducted a series of clinical studies of novice programmers at Harvard Graduate School of Education, and reflected on the why some youngsters learned much better than others.

They suggested that underlying reason for their difference is their different patterns of learning. They further classified students to different categories based on their learning patterns: stoppers who easily give up the hope of solving the problem on their own and just stop, movers who consistently try different approaches and never seem to get stuck, and extreme movers who move too fast without reflection on their previous ideas. And they pointed out that students often tend to take the approach of tinkering, which is solving a programming problem by first writing some skeleton code and then making successive small edits.

To help students better learn programming skills, they suggest that students should closely track their written code and some instructions may be useful to lead them to better learning practices and improve their learning efficiency.

2.2 “Early Prediction of Student Frustration” [3]

Scott et al. collected training data by closely monitoring features including temporal features, locational features, intentional feature and physiological response when students interact with the interactive task-oriented learning environment Crystal Island. Besides, they designed a “self-report emotion dialog” box which will show up

periodically to ask participants to choose their affective state, which serves as class labels to induce frustration model.

They developed frustration model by both sequential model as n-grams, as well as non-sequential modeling techniques like naïve Bayes, decision trees, and support vector machines. And they utilized convergence point information gained by n-grams model to enable non-sequential models to make early predictions, approximately 35 seconds prior to the self-reported affective state. And test results showed that the induced frustration recognition model was both efficient and accurate, with decision tree model achieving the highest accuracy of 88.8% and precision of 88.7%.

2.3 “Coarse-Grained Detection of Student Frustration in an Introductory Programming Course” [4]

The researchers attempted to automatically detect student frustration when students are learning introductory java programming under BlueJ environment.

They collected student interaction data including students’ average number of consecutive compilations with the same edit location, average number of consecutive pairs with the same error, the average time between compilations, and average number of errors. And during lab sessions, they continuously observed students and classified their behaviour into different categories.

Based on the collected data and labels, they built a linear regression model to detect frustration by using Weka. The result indicates that the frustration detection model was able to predict average frustration during each lab session better than would be expected by chance, whereas the detector performance to detect frustration on a per-lab basis was unstable. It is suggested that more data such as keystroke and mouse movement data could also be utilized as well as the coarse-grained semantic data to develop a finer-grained detector for frustration.

2.4 “Advances in Intelligent Tutoring Systems: Problem-solving Modes and Model of Hints” [5]

This paper focuses on the issues of providing an adaptive support for learners in intelligent tutoring systems when solving practical problems. The results of the analysis shows that the hinting model used in this paper, which divides the hint into three general hint categories, provides greater adaptive abilities to the intelligent tutoring system and helps improve student learning efficacy.

This model divides hints into two layers, namely the layer of the general hint categories and the layer of hints within the general categories. Among the layer of general hints, there are three categories: general hint, hints of average informativeness, and specific hints. When the user first request a hint, the most general hint will be presented. If further hinting request is made, a subsequent (more specific) hint will be given. This process continues until the most specific hint is reached. The mechanism is shown in the picture below.

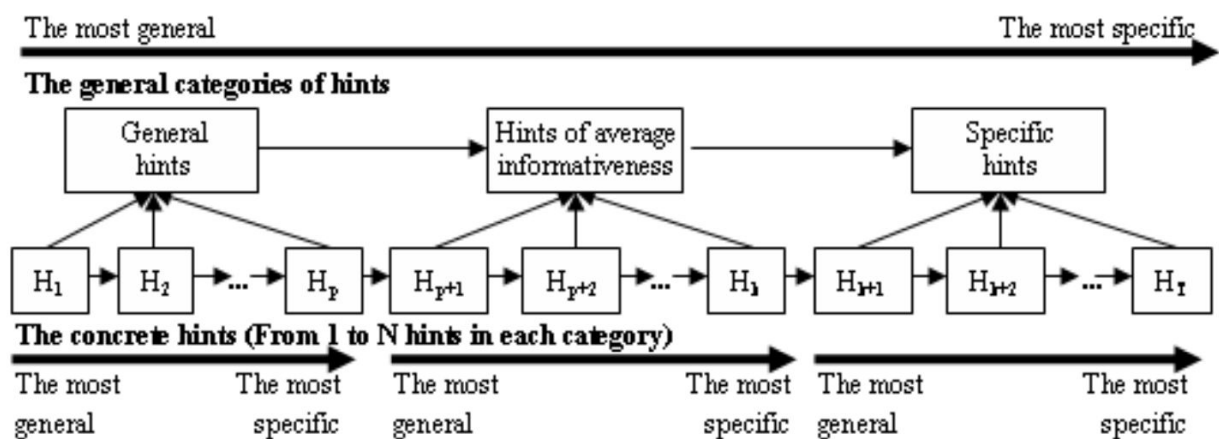


Figure 1: Multi-layer hinting mechanism

2.5 “Developing a Generalizable Detector of When Students Game the System” [6]

In this paper, the researchers point out that some students, when working in an intelligent tutoring system, attempt to succeed in the system by exploiting properties of the system rather than by learning the material itself. And the author presents a detector that can accurately detect whether a student is gaming the system when solving mathematical problems under the environment of Cognitive Tutor. Furthermore, the detector is also capable of predicting when students are gaming the system, which may be utilized to suggest appropriate interventions by the intelligent tutoring system in order to facilitate student learning.

After comparing results from other intelligent tutoring systems which have the functionality of detecting gaming, the researchers concluded that the gaming phenomenon is fairly robust and exists among many intelligent tutoring systems. However, it is still unclear to what degree the two distinct types of gaming - harmful gaming, which is associated with poor learning outcomes, and the other gaming behaviours, which do not significantly affect learning - exist in these systems.

3 Motivation

Researchers found that frustration will likely to lead to student disengagement from learning. And hence effectively detecting and predicting frustration and intervene appropriately either from teacher or intelligent tutor can hopefully keep student engaged and motivated in the task [7].

Previously SAGE researchers designed an automatic five-second hinting functionality in hope of maintaining a sense of flow for the students and keep them from feeling frustrated. However, the fixed five-second hinting intervene interval may not be suitable for all students, and better mechanism to decide appropriate hinting frequency tailored for different students is a necessity.

In order to adjust the hinting frequency according to different student behaviors and to improve the effectiveness of the intelligent hinting system, we divided students into four categories based on the theory proposed by Perkins, et al [2]. These behavior categories are extreme movers, movers, stoppers and tinkerers, which are classified by different learning patterns. Stoppers are the one who tend to stop and give up when they encounter some obstacles, who will need some external help otherwise they will get stuck. In the contrary, movers tend to consistently try different methods and never seem to get stuck, which is the most ideal learning practice. Extreme movers tend to try consistently like movers, however, they try new approaches too quickly without reflecting on the past failure. And tinkerers are the ones who first write some skeleton work and then make many successive small edits to complete their work, which may result in low learning efficiency if they choose the wrong path in the first place.

Our aim is to classify students into these four behavior types based on their interaction with SAGE, and then adjust the hinting frequency for them based on their behavior type accordingly. The most challenging part to build the behavior detection models is

obtaining enough data. Since the SAGE platform is still under construction and cannot be used in real experiments, we are unable to obtain real students interaction data and hence only used the mock data to train the models. And we also modified the SAGE analyzer so that we can extract more features to build machine learning models. At last, we adjust the hinting frequency for different behaviors based on their characteristics in order to meet the needs for different types of students.

4 Implementation

In this section we will introduce the work we have done on behavior detection and how we implemented the project. There are basically four phases including mock data generation, feature extraction, building models and adjusting the hinting frequency. We will discuss each phase in detail and the code repositories are also included as below.

4.1 Code Repositories

Project aspect	Repository location	Branch name
Generate mock data (4.2)	https://github.com/cu-sage/scratch-analyzer/tree/main/machineLearning	machineLearning
Feature extraction (4.3)		
Build models (4.4)		
Dashboard integration (4.5)	https://github.com/cu-sage/sage-scratch.git	block-ids

4.2 Generate Mock Data

In order to build machine learning models to classify students into different behavior types, a large amount of students interaction data will be needed, which are not available at this stage of SAGE. Therefore although it is not an ideal approach, we have to generate mock data and use them to train the model instead of real data.

The mock data are generated using the 22 complete .se files in the previously obtained sample input file, each of which represent the block structures of a project in the final complete stage. Since each behavior is characterized by the learning pattern, hence to produce mock data representing each behavior type, the crucial part is to model different learning patterns and generate corresponding mock .se files which represent how the student make his way to the complete stage.

To stimulate the behavior of each type of students, we make some assumptions about the probability models of action intervals and action types for each behavior based on the definitions. The following code in figure 2 show how we generate mock data for stoppers and extreme movers. We first read the block structure from the complete .se file into an array *I*, then write the first *i* lines of *I* into timestamped .se files representing the student movement per second. As shown in the code, we assume the action interval of stopper to be 5 seconds on average and with standard deviation of 3. And for the extreme mover type, we assume the mean of action interval to be 1 seconds and the standard deviation to be 1 since they tend to move faster than stopper. And also for the extreme movers we model their likelihood to cancel their previous movement by decreasing the value of *i* by 1. The generation of timestamped .se files will be continued until the final stage is reached.

```

def stopper(url, l):
    if not os.path.isdir(url):
        os.makedirs(url)
    i = 1
    count = 0
    while i < len(l):
        interval = abs(int(random.normal(5,3)))
        t = 0
        while t < interval:
            ouf = open(url+str(count)+'.se', 'w')
            ouf.writelines(l[:i])
            ouf.close()
            t += 1
            count += 1
        i += 1

def extremeMover(url, l):
    if not os.path.isdir(url):
        os.makedirs(url)
    i = 1
    count = 0
    while i < len(l):
        interval = abs(int(random.normal(1,1)))
        t = 0
        while t < interval:
            ouf = open(url+str(count)+'.se', 'w')
            ouf.writelines(l[:i])
            ouf.close()
            t += 1
            count += 1
        if random.random() > 0.8:
            i -= 1
        else:
            i += 1

```

Figure 2: Functions from mockdataProducer.py

The output files are sequences of timestamped .se files as shown in the picture below. The first level of directories represent four behavior categories respectively and the second level of directories represent the mock projects each type of students completed. And for each project, it has a sequence of timestamped .se files associated

with it. Here we generated 20 projects for each behavior type and the number of timestamped .se files for each project depends on the project and behavior type.

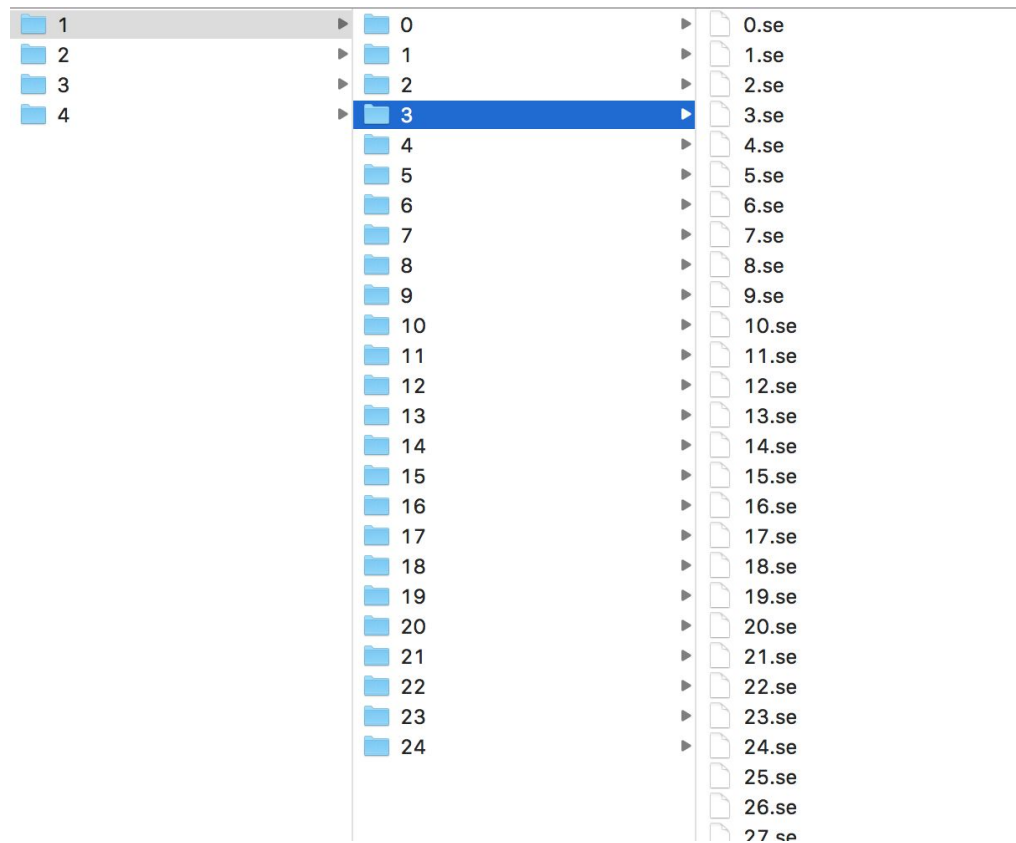
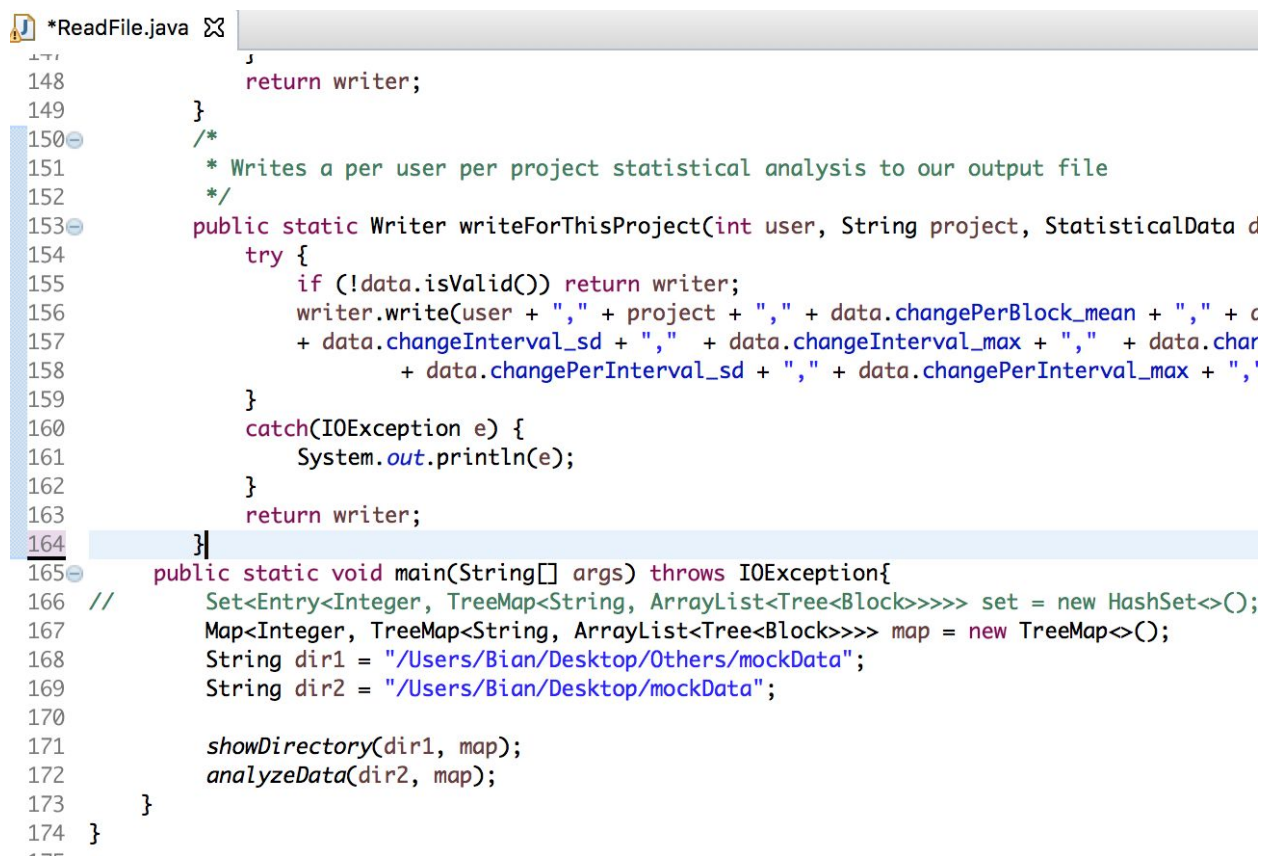


Figure 3: Output mock data files by the mockdataProducer.py

4.3 Feature Extraction

In this part, we used the time-stamped .se file we have created in the last section to calculate different features for our machine learning model.

So we wrote a java class called ReadFile.java in the scratch analyzer. First we try to read these time-series .se files into a tree object, and then calculate different features according to the tree, and output the features in a .CSV file.



```
148         return writer;
149     }
150     /*
151     * Writes a per user per project statistical analysis to our output file
152     */
153     public static Writer writeForThisProject(int user, String project, StatisticalData d
154     try {
155         if (!data.isValid()) return writer;
156         writer.write(user + "," + project + "," + data.changePerBlock_mean + "," + c
157         + data.changeInterval_sd + "," + data.changeInterval_max + "," + data.char
158         + data.changePerInterval_sd + "," + data.changePerInterval_max + ","
159     }
160     catch(IOException e) {
161         System.out.println(e);
162     }
163     return writer;
164 }
165 public static void main(String[] args) throws IOException{
166 // Set<Entry<Integer, TreeMap<String, ArrayList<Tree<Block>>>> set = new HashSet<>();
167 Map<Integer, TreeMap<String, ArrayList<Tree<Block>>>> map = new TreeMap<>();
168 String dir1 = "/Users/Bian/Desktop/Others/mockData";
169 String dir2 = "/Users/Bian/Desktop/mockData";
170
171 showDirectory(dir1, map);
172 analyzeData(dir2, map);
173 }
174 }
```

Figure 4: ReadFile.java

As you can see, we have 4 main features in general, and they are Time interval between two actions, Interval between two block removal actions, Number of actions taken in every 10s and Changes of each block in one project. And based on these 4 main features, we calculated 24 features in total, which includes the mean, standard

deviation, range, max_value, min_value, and the skewness of each of these main feature, and this would be very useful in training our machine learning models.

Userld	ProjectName	changePerBlock_mean	changePerBlock_sd	changePerBlock_max	changePerBlock_min	changePerBlock_range	changePerBlock_skewness	changeInterval_mean
1	0	48.57142857142858	94.02583231674762	330	2	328	2.691814861997231	2.2531645569620258
1	1	50.28571428571429	48.576890449207276	166	2	164	1.0838399129311116	1.8783783783783785
1	10	15.80952380952381	18.16485355740323	80	2	78	2.715707596395749	1.5526315789473684
1	11	445.8846153846155	842.7378736476968	4722	2	4720	3.4450694473474366	1.7239583333333337
1	12	510.054054054054	1286.5204136642274	5388	2	5386	3.2081708523091788	1.9673024523160763
1	13	50.52631578947368	145.82637445075848	648	2	646	4.266059927427594	2.3368421052631576
1	14	21.2	29.353740907474773	118	2	116	2.4131427393915246	2.120689655172414
1	15	59.388888888888886	71.44199556394494	332	2	330	2.1283477530251673	1.9357798165137614
1	17	85.33333333333334	94.68766754950146	318	2	316	1.2648002488646999	2.3975903614457836
1	2	43.36363636363636	87.23779513738813	308	2	306	2.73803199273866	2.1136363636363638
1	20	11.666666666666668	15.343134122658329	60	2	58	2.381405949557531	1.9444444444444442
1	21	55.8	44.518003454545756	160	2	158	0.7158272963302862	1.9420289855072466
1	22	19.7	24.762237803813928	102	2	100	2.24583860046676	2.125
1	23	455.2432432432432	1151.2290390275518	4784	2	4782	3.2433519756054654	1.893732970027248
1	24	81.81818181818183	187.3544225623715	902	2	900	4.369843310230596	1.7999999999999996
1	3	40.36363636363636	81.84919085299526	294	2	292	2.8187695931585073	2.246376811594203
1	4	410.0769230769231	765.5973406984017	4238	2	4236	3.3976422283336847	1.7656675749318802
1	5	7.4444444444444445	9.375953110592338	38	2	36	2.544030827937867	1.88

changeInterval_sd	changeInterval_max	changeInterval_min	changeInterval_range	changeInterval_skewness	blockInterval_mean	blockInterval_sd	blockInterval_max	blockInterval_min
1.9772316468725544	10	1	9	2.1085441563855283	2.2531645569620258	1.9772316468725544	10	1
1.671294328232133	8	1	7	2.278609266610831	1.8783783783783785	1.671294328232133	8	1
0.9211440718682281	5	1	4	1.9176318381921558	1.5526315789473684	0.9211440718682281	5	1
1.5025996474807384	10	1	9	3.2726462893187045	1.7239583333333337	1.5025996474807384	10	1
1.675610670369221	10	1	9	2.3012227208413987	1.9673024523160763	1.675610670369221	10	1
1.8827917609840874	10	1	9	1.4517630834235606	2.3368421052631576	1.8827917609840874	10	1
2.1445777092749934	10	1	9	2.656433328442773	2.120689655172414	2.1445777092749934	10	1
1.7011718692854392	8	1	7	2.1252101035369133	1.9357798165137614	1.7011718692854392	8	1
2.2959239721133464	10	1	9	1.9202696689919655	2.3975903614457836	2.2959239721133464	10	1
2.019626790274833	10	1	9	2.462715076224234	2.1136363636363638	2.019626790274833	10	1
1.308094458023239	7	1	6	2.1353121604803995	1.9444444444444442	1.308094458023239	7	1
1.9620222008923711	10	1	9	2.7060688890999174	1.9420289855072466	1.9620222008923711	10	1
2.089741916265199	10	1	9	2.8340676427151417	2.125	2.089741916265199	10	1
1.3639139135124552	8	1	7	1.7329512618568832	1.893732970027248	1.3639139135124552	8	1
1.6848470783484637	10	1	9	3.158451487329315	1.7999999999999996	1.6848470783484637	10	1
1.8818466923668467	9	1	8	1.64981578536487	2.246376811594203	1.8818466923668467	9	1
1.6676342094130723	10	1	9	3.2972639359092595	1.7656675749318802	1.6676342094130723	10	1
1.16619037896906	5	1	4	1.6217800986302477	1.88	1.16619037896906	5	1

Figure 5: StatisticalAnalysis.csv

4.4 Build Behavior Classification Models

Using the data of features extracted from mock data, we are able to build machine learning models using Python Scikit Learn library.

First we perform some simple preprocessing procedures to transform the data, which include splitting data into training and testing sets, with 80% for training and 20% for testing, and then use Linear Discriminant Analysis method to reduce the dimensionality of data.

Next we build supervised classification models including Decision Tree, K Nearest Neighbours, Logistic Regression and Support Vector Machine models. After tuning the parameters of each model, we use cross-validation to assess the accuracy of each model. The result is shown in the picture below.

```
MacBook-Pro:Desktop zhangmengqiao$ python -W ignore Models.py
Decision tree accuracy: 0.8309115927072582
KNN accuracy: 0.8222772617819057
Logistic regression accuracy: 0.8209700722394221
SVM accuracy: 0.8085517715858274
```

Figure 6: Model accuracy output by Models.py

We can see that the accuracy for these four models are all around 80%. And it is worth mentioning that the dimension reduction in the preprocessing part plays an important role in boosting the accuracy scores. Previously we have very unstable performance for different models without Linear Discriminant Analysis, some have an accuracy score as low as 60%. This may be caused by the redundant or correlated features we extracted from .se files, since there are indeed several features representing similar meanings in StatisalAnalysis.csv. A larger size of real data may be considered to further boost the model performance in the future, here we only use limited mock data due to save processing time.

4.5 Adjust the Hinting System

The past hint system in our platform issues Hints every 5 seconds, when available, to all students in the same manner, by shaking the right block, regardless of a student's type. In order to integrate our behaviour detection model to the current hint system and provide student with personalized hinting frequency, which could improve the user experience, we modified the code in the .swf file, which is the main file to control the sage-editor.

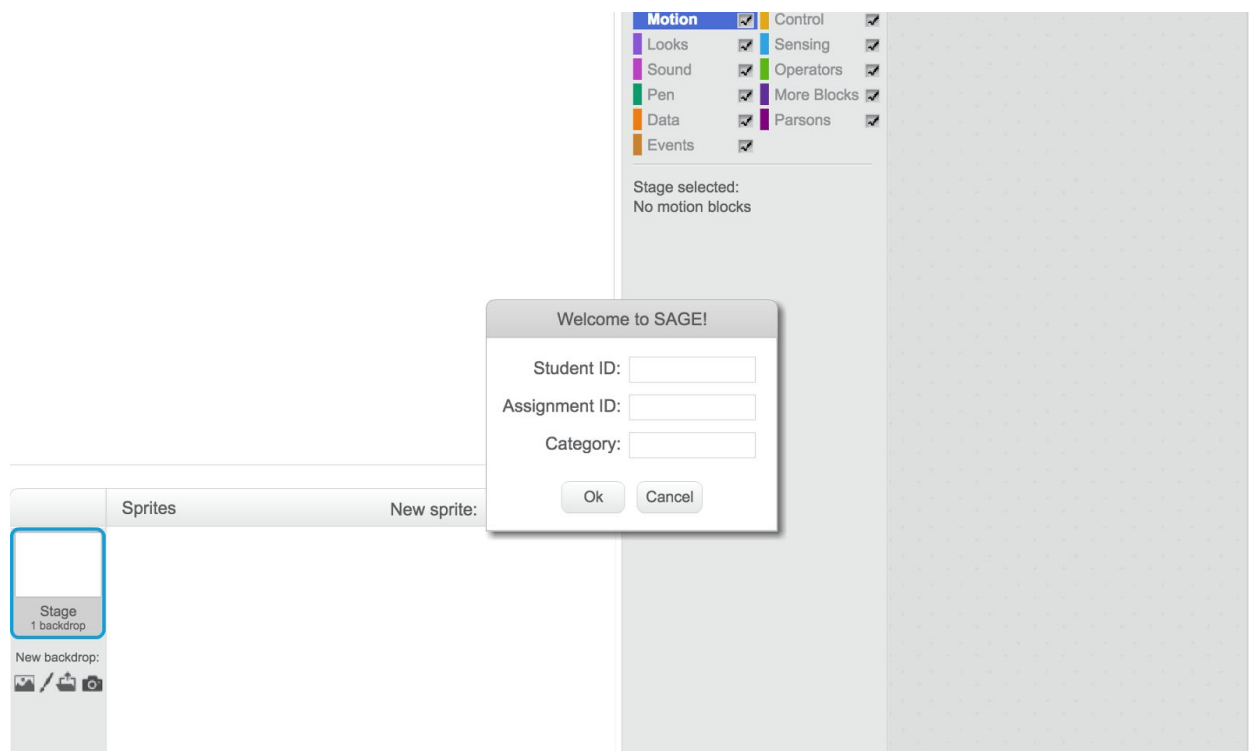


Figure 7: Dashboard integration

We added a category type to the hint system, so now when user start a new project in the platform, he/she will be asked to input a category type manually (currently we haven't integrate our affect detection model to the hinting system, so when the behaviour detection model is integrated to the system, this platform could automatically

classify the user to different behavior based on the previous projects he/she has done, and allocate different frequency to each behaviour). Here we use 1, 2, 3, 4 to represent the 4 different behavior (1= extreme mover, 2 = mover, 3 = stopper, 4 = tinkerer). The initial frequency we allocate to each behaviour are as follows, this is only for test purpose, and we could easily change the frequency for each behaviour by modifying the code in the .swf file.

```
if (behaviorType == 1) {  
    //extreme mover  
    Hints.categoryTimer = new Timer(3000);  
    Hints.blockTimer = new Timer(3000);  
} else if (behaviorType == 2) {  
    //mover  
    Hints.categoryTimer = new Timer(15000);  
    Hints.blockTimer = new Timer(15000);  
} else if (behaviorType == 3) {  
    //stopper  
    Hints.categoryTimer = new Timer(4000);  
    Hints.blockTimer = new Timer(4000);  
} else if (behaviorType == 4) {  
    //tinkerer  
    Hints.categoryTimer = new Timer(5000);  
    Hints.blockTimer = new Timer(10000);  
}
```

Figure 8: Hinting frequency adjustment

5 Future work

In this section we will discuss some possible improvements that can be done to improve the current intelligent hinting system.

5.1 Obtain Real Student Data

Since we use mock data to train the behavior detection model, the resulting model is very unreliable and need improvement. However, the pipeline we built to process the log data and predict behavior types can be used when real data become available.

To obtain information about behavior type for training models, self-reporting mechanism can be considered. Participants in the experiment could be informed the definitions of different behavior types and asked to choose their behavior types after they have solved the task. Then models can be built with their log data and selected behavior types to classify behavior types. And hinting frequency for each behavior types may also be modified based on student feedback.

Furthermore, since current hints are also generated using mock data, real student log data and feedback are also important to improve the hinting generation model.

5.2 Improve the Behavior Detection Integration

Currently we ask students to input the behavior type to adjust to corresponding hinting frequency. When real data can be obtained to train the models, we could integrate the model to classify behavior types after a student has finished a project, and store his behavior type into the databases.

The predicted behavior type when he conduct a new project can be decided by considering his previous stored behavior types with some simple model, for example, calculate a weighted average of all previous behavior types, where more recent

behavior types are assigned more weight. This predicted behavior type can then be used to determine the appropriate automatic hinting frequency for this project.

5.3 On-demand Hinting

Currently, our intelligent tutoring system only supports automatic hinting, so in order to provide users with better user experience, like many well-established intelligent tutoring systems do, it is important to add the on-demand hinting function to the current hinting system.

This can be done by simply adding an on-demand hinting button to the system, which will give students the option to click on a hint button to receive on-demand feedback when needed. One challenge for this part is that now the intelligent tutoring system does not provide hints at every project state, especially at the very beginning of a new project. So we will try to revert to the latest BlockList variable to check if any hints are available based on the previous moves. If there is still no hint available, clicking the button could simply return “Try moving some blocks around first”.

5.4 Multi-layer Hints

The current method that a hint is presented to the students is suggesting which block to be taken in the next step by shaking the specific block. In order to better encourage students to think and learn, future researchers can provide users with a more complete hinting system. The hints can be divided into three categories, general text hints, category-level block hint, and the specific hint, which are ranged from less informative to more informative.

At first, the intelligent system allows the learner to receive a more general text hints that do not mention which blocks to use next. Further requesting help during problem-solving, the learner will receive category-level hint like shaking several correct blocks or incorrect blocks. For example, we could infer the right block type from the right

block and shake several blocks of that type. If after receiving of a category-level hint, the learner is still not capable to execute a correct action, he will be presented with a specific hint, i.e. the right block will shake.

Such approach starts to present the learner with less informative hints, and as the learner timely receives a more informative hint providing help, they will be less likely to get frustrated and thus gain more pleasant and gameful learning experience [5].

5.5 Avoid Gaming the System

A well-established concern about the on-demand hinting function pointed out that some students tend to “game the system” when they work in an interactive learning environment, so that they could figure out the answer easily from system help instead of putting much of their own effort into learning the material and applying their knowledge [6]. So in order to stop students abusing the hint button when on-demand hinting is available, future researchers can add some punishment mechanism (like setting an on-demand hinting time limits) to the current system in order to discourage students from gaming the system.

6 References

- [1] Bender, J. (2015). Tooling Scratch: Designing a Collaborative Game-Based Learning System to Infuse Computational Thinking within Grade 6-8 Curricula.
- [2] Perkins, D. N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1986). Conditions of learning in novice programmers. *Journal of Educational Computing Research*, 2(1), 37-55.
- [3] Mcquiggan, S. W., Lee, S., & Lester, J. C. (2007, September). Early prediction of student frustration. In *International Conference on Affective Computing and Intelligent Interaction* (pp. 698-709). Springer, Berlin, Heidelberg.
- [4] Rodrigo, M. M. T., & Baker, R. S. (2009, August). Coarse-grained detection of student frustration in an introductory programming course. In *Proceedings of the fifth international workshop on Computing education research workshop* (pp. 75-80). ACM.
- [5] Anohina, A. (2007). Advances in intelligent tutoring systems: problem-solving modes and model of hints. *International Journal of Computers Communications & Control*, 2(1), 48-55.
- [6] d Baker, R. S., Corbett, A. T., Roll, I., & Koedinger, K. R. (2008). Developing a generalizable detector of when students game the system. *User Modeling and User-Adapted Interaction*, 18(3), 287-314.
- [7] Dweck, C. S. (2000). Self-theories: Their role in motivation, personality, and development. Psychology Press.