Enhanced Data Collection, Student Progress Modeling, and Intelligent Hinting in SAGE

Sambhav Anand and Allison Sawyer

Final Report

COMS W3998, Section 14

May 10, 2017

Table of Contents

1 Introduction **3**

2 Related Work **4**

2.1 "Modeling How Students Learn to Program" 4

2.2 "A Framework for Capturing Distinguishing User Interaction Behaviours in Novel Interfaces" 4

2.3 DT Tutor 5

2.4 Andes Physics Tutor 5

2.5 "Applying Web usage mining for personalizing hyperlinks in Web-based adaptive educational systems" 5

3 Planning **6**

3.1 Motivation 6

3.2 Project Vision 7

3.2.1 Behavioral Data Collection Vision 7

3.2.2 Hint Generation Vision 7

4 Implementation **8**

4.1 Code Repositories 8

4.2 New Technologies Used 8

4.2.1 RapidMiner Studio 8

4.2.2 scikit-learn 11

4.3 Behavioral Data Processing 11

4.3.1 Unique Block IDs in Scratch Editor 11

4.3.2 Scratch Analyzer Enhancements 12

4.4 Hint Generation 15

4.4.1 Machine Learning Algorithms Used 15

4.4.2 Dashboard Integration 23

4.4.3 Scratch Editor User Interface Modifications 26

5 Limitations and Assumptions **29**

5.1 Automatically Generated Training Data 29

5.2 Assumptions Made for Dashboard Integration 29

5.3 Limitations of Hinting Strategy 30

5.4 RapidMiner Issues 30

6 Future Work **31**

6.1 Improving and Expanding the Hinting System 31

6.1.1 Personalizing Hint Frequency and Granularity 31

6.1.2 Integrating External Information on Student Knowledge/Ability 32

6.1.3 Supporting On-Demand Hinting 32

6.2 Collecting Mouse Data 32

6.3 Affect Detection System 33

6.4 Multi-Puzzle Models 33

6.5 Additional k-Means Features 34

6.6 Studies With Real Student Users 34

7 Conclusion **34**

8 References **35**

1 Introduction

This report describes our contributions to the existing Social Addictive Gameful Engineering (SAGE) project [**1**] through the development of an intelligent hinting system powered by machine learning and the expansion of SAGE's user data collection and processing capabilities. We modified the Scratch Analyzer [**2**] and Scratch Editor codebases to automate the collection of additional user behavioral data; these data were then distilled into features that could be leveraged by future machine learning algorithms to analyze students' behavior as they use SAGE. Additionally, we introduced new machine learning scripts into the SAGE architecture with the aim of automatically determining hints to issue based on a student's project state, his/her current problem-solving approach, and the prior, now-completed work of other students on the same project. This machine learning system was integrated into the Dashboard codebase.

SAGE is designed to immerse students in an enjoyable, game-like learning experience. It offers adaptable features and feedback in order to maximize engagement and minimize the risk of negative emotions including boredom, frustration, and confusion. Research attests to the usefulness of maintaining a sense of "flow" for students playing a learning game [**3**] [**4**], which entails a high level of focus, a sense that time is passing more quickly than it is, and immediate feedback about one's performance. This project enhances the "Addictive" component of SAGE, which includes the concept of flow, by improving the quality of real-time feedback in order to help ward off frustration for struggling students. Additionally, this project contributes to the "Social" dimension of SAGE by developing student models and by providing automated assistance, thereby enabling teachers to devote their attention to students who need the most help. One can imagine limitless potential applications of the data collected and analyzed in this project; we envision some of them in the Future Work section of this report.

2 Related Work

This section discusses previous research that inspired and guided our own efforts over the course of this project.

2.1 "Modeling How Students Learn to Program" [5]

Piech et al. developed a graphical model of how university students progressed through a programming assignment by taking snapshots of their projects at various points and then modeling their progress with a Hidden Markov Model (HMM). The researchers used the k-means clustering algorithm to find clusters of paths taken through the HMM. Furthermore, they showed that these clusters had implications for student success, as they were predictive of students' midterm grades.

Our project was strongly inspired by this approach to student progress modeling. We also used clustering algorithms to group students according to the general paths they took through an assignment. Instead of using the clusters to predict student outcomes, we used them to increase the relevance of hints generated for each cluster.

2.2 "A Framework for Capturing Distinguishing User Interaction Behaviours in Novel Interfaces" [6]

Kardan and Conati developed a user modeling framework based on two steps: behavior discovery and user classification. Behavior discovery involved extracting user log data, transforming them into feature vectors, and running k-means clustering on the feature vectors in order to separate users into two clusters. They then performed association rule mining to identify the behaviors that characterized each cluster. In the user classification phase, their classifier was used on-line to classify new users based on their interactions with the program, an artificial intelligence algorithm visualization tool. By comparing tests taken by users both before and after the study, the researchers found a significant difference in learning gains between clusters.

Like Kardan and Conati, we clustered users with the hope of identifying students with similar problem-solving traits. However, we then ran sequential pattern mining, rather than association rule mining, within the clusters, as our ultimate objective was generating hints rather than identifying characteristic behaviors.

2.3 DT Tutor [7]

DT Tutor is an intelligent approach to hinting that uses a probabilistic model to assesses how students might react to a variety of possible hints that could be issued by an intelligent tutoring system (ITS). It then evaluates the utility of the possible resulting student states and takes the action with the highest expected utility. This approach allows the ITS to tune its hinting strategy to fit student needs at a particular time.

This advanced approach should be taken as inspiration for further work on this project. Our hinting strategy, as it is currently implemented, is not probabilistic and does not take external information about student knowledge or ability into account. See section 6.1.2 for further discussion of this topic.

2.4 Andes Physics Tutor [8]

Andes is an ITS designed to help students learn college-level physics by working through problems in an environment with helpful visualizations and immediate feedback. Its granular hinting capability allows it to give feedback to the user at any step in his/her problem-solving process. When Andes identifies errors likely resulting from specific, common misconceptions, it provides error-specific feedback that aims to help the learner avoid making the same mistake in the future.

2.5 "Applying Web usage mining for personalizing hyperlinks in Web-based adaptive educational systems" [9]

Romero et al. used data mining techniques including sequential pattern mining to recommend links for a student to visit within an online educational system based on the sequences of webpages s/he, and other students in his/her cluster, had previously visited. Similarly, we clustered students and then applied sequential pattern mining. In a manner analogous to their searching for patterns of webpages in order to predict which page should be visited next, we searched for patterns of Scratch blocks in project solutions in order to predict which block should be used next.

3 Planning

This section discusses the motivation for our particular enhancements of SAGE, as well as the initial planning with which the project began.

3.1 Motivation

The motivation for this project was to leverage machine learning algorithms in order to enhance SAGE's ability to automatically, intelligently interact with students without requiring additional manual intervention by teachers. To this end, the project split into two major components: (1) the exploratory extraction of student behavioral data to be distilled into machine learning features, and (2) an automated hinting system.

The aim of capturing additional user behavioral data was to obtain useful input for future machine learning algorithms. Such algorithms could be used to identify undesirable traits such as frustration or boredom in students, or to cluster students based on their learning styles. They could also be used in conjunction with hinting to modulate hinting frequency and/or detail based on the student's perceived affective state.

The motivation for automated hinting was to enhance the gameful nature of SAGE, especially its "addictive" quality. Research into educational games, such as that of Kickmeier-Rust & Albert, supports the notion that hinting, as part of a system of "adaptive problem-solving support," is an important means of maintaining the flow of a game, keeping the learner motivated, and guiding him/her in the correct direction if necessary [**4**]; a sense of "flow" is the key to maintaining a balance between the undesirable poles of boredom and anxiety as the student plays [**3**]. Since manually writing several specific hints for each unique project is a time-consuming task, our aim was to generate hints in a way that does not require teacher intervention. The clusters of students generated as an intermediary step in the hinting process, however, may prove useful to teachers who are interested (and enhance the "Social" component of SAGE): students are clustered by their problem-solving approach, which could guide teachers in matching up similarly-thinking students for group work, or, alternately, in pairing up differently-thinking students so that they can explain their approaches to one another.

3.2 Project Vision

3.2.1 Behavioral Data Collection Vision

This component of the project aimed to capture additional user behavioral data in Scratch Editor and analyze these data in Scratch Analyzer [**2**]. Specifically, it focused on the frequency with which users manipulate blocks, based on the assumption that this could serve as a reasonable proxy for their affective state as they work in Scratch. This part of the project would require developing an algorithm to parse the multiple JSON files associated with each project in order to extract useful information from them.

3.2.2 Hint Generation Vision

This component of the project aimed to leverage the knowledge contained in existing complete, correct submissions for a particular project in order to intelligently generate hints for students currently working on the same project. On a high level, the idea behind our hinting approach is as follows: if a particular sequence of two blocks appears in many correct submissions, and if a student currently working on the project has just placed the first block in this sequence into his/her script, the student will likely soon want to use the second block in the sequence. Before these sequential hints are generated, students are separated into groups based on their high-level approach to solving the assignment, to minimize the chance that an unrelated use of a particular block sequence will trick the system into issuing a hint based on that sequence for a student solving the assignment in a wholly different way.

The hinting system will be discussed in lower-level detail in section 4.4.

4 Implementation

This section relays the locations of our finalized source code and provides an in-depth discussion of how the project was implemented.

4.1 Code Repositories

Project aspect	Repository location	Branch name	Party responsible
Behavioral data processing (4.3)	**https://github.com/cu-sage/scratch-analyzer**	master	Sambhav
Machine learning scripts (4.4.1) and Dashboard integration (4.4.2)	**https://github.com/cu-sage/sage-frontend** (scripts discussed in 4.4.1 are in **machine_learning** subdirectory)	machine-learning	Allison
Scratch Editor block ID and UI modifications (4.4.3)	**https://github.com/cu-sage/sage-scratch**	block-ids	Allison

4.2 New Technologies Used

This section discusses new technologies introduced into the SAGE system for the purposes of our project.

4.2.1 RapidMiner Studio [10]

Deciding which tool(s) to use to implement machine learning algorithms, we mainly considered RapidMiner and Weka, both of which are powerful machine learning software with support for the necessary algorithms. In the end, we chose RapidMiner because its simple-to-use graphic user interface increases efficiency when designing processes, and these processes can still be run from the command line as Java executables.

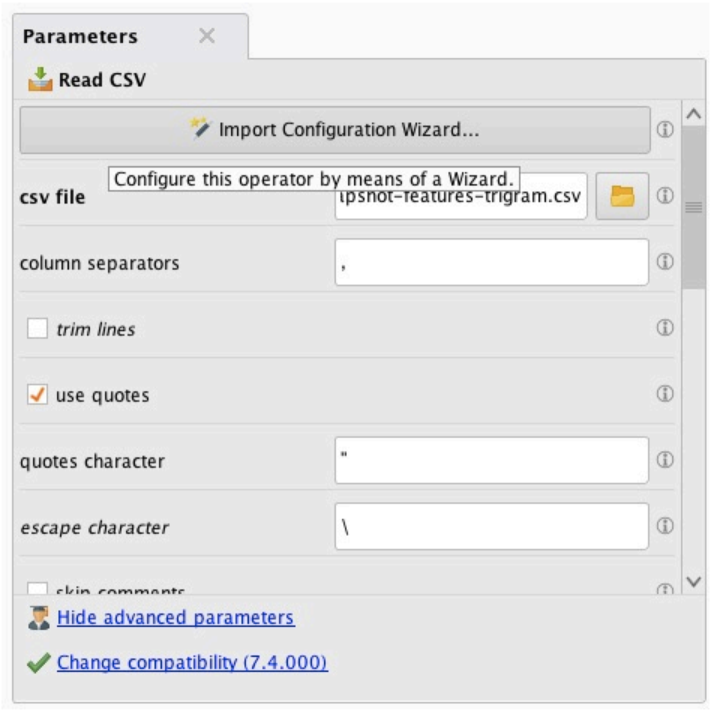
4.2.1.1 RapidMiner Setup Details

The RapidMiner Education Program offers free, temporary licenses to students and educators. Below are the instructions for setting up RapidMiner Studio for use with SAGE (the applications of RapidMiner in SAGE are discussed in section 4.4.1).

1. Apply for a RapidMiner Studio license under the RapidMiner Education Program at https://rapidminer.com/educational-program.
2. Install Java 8 if necessary.
3. Install RapidMiner Studio by following the instructions here: http://docs.rapidminer.com/studio/installation.
4. Download the SAGE RapidMiner processes from the sage-frontend Git repository at https://github.com/cu-sage/sage-frontend. They are located in the subdirectory `machine_learning/RapidMiner-repo`.
5. Copy the processes into your local RapidMiner repository. They must be in your local repository in order to be found by the RapidMiner shell script called from the Dashboard. On a Mac, the default location of the local RapidMiner repository is: `~/.RapidMiner/repositories/Local\ Repository/`.
6. If using a Mac, verify that the RapidMiner shell script is located at the following location on your machine:
   ```
   /Applications/RapidMiner\
   Studio.app/Contents/Resources/RapidMiner-
   Studio/scripts/rapidminer-batch.sh
   ```
   This path is hardcoded into `sage-frontend/routes/researcherRoutes.js`. If the shell script is at a different location on your machine, which will certainly be the case if you are not using a Mac, change the variable assignment in `researcherRoutes.js`.
7. For each of the RapidMiner processes now in your local repository, edit the .rmp file to change hardcoded pathnames. To quickly locate these pathnames, search for "csv_file". In the latter portion of the pathnames, you will see the familiar `sage-frontend/machine_learning/…` directory structure from the sage-frontend Git repository. Change the beginning portion of the pathnames to point to the location of this repository on your machine.

4.2.1.2 Editing RapidMiner Processes

Future researchers who want to modify the existing RapidMiner processes can do so either by editing them in the RapidMiner Studio application GUI, or by editing the processes' .rmp files (of which the original versions can be found in `sage-frontend/machine_learning/RapidMiner-repo`). Many parameter values, such as *k* for the clustering processes and output pathnames, can easily be changed by editing the .rmp file, which is faster than opening the GUI. However, editing the .rmp file is complicated when it comes to changing the input, because each feature is saved as a separate parameter when a new input file is selected from the GUI. Therefore, simply changing the input pathname may break the process, since it will still expect the same number and type of input features (e.g., if the process is configured to have 140 features and one switches the input to a new file with 200 features, only the first 140 of those features will be used, and if any of the features types are different, the process may break). The easiest way to change process input, therefore, is to open RapidMiner Studio and use the "Import Configuration Wizard…" feature of the "Read CSV" operator.


Parameters
Read CSV
Import Configuration Wizard...
Hide advanced parameters
Change compatibility (7.4.000)

Configure this operator by means of a Wizard.

csv file: ipshot-features-trigram.csv
column separators: ,
☐ trim lines
✓ use quotes
quotes character: "
escape character: \
☐ skip comments

Figure 1: The "Import Configuration Wizard…" button in RapidMiner Studio allows for easy input configuration.

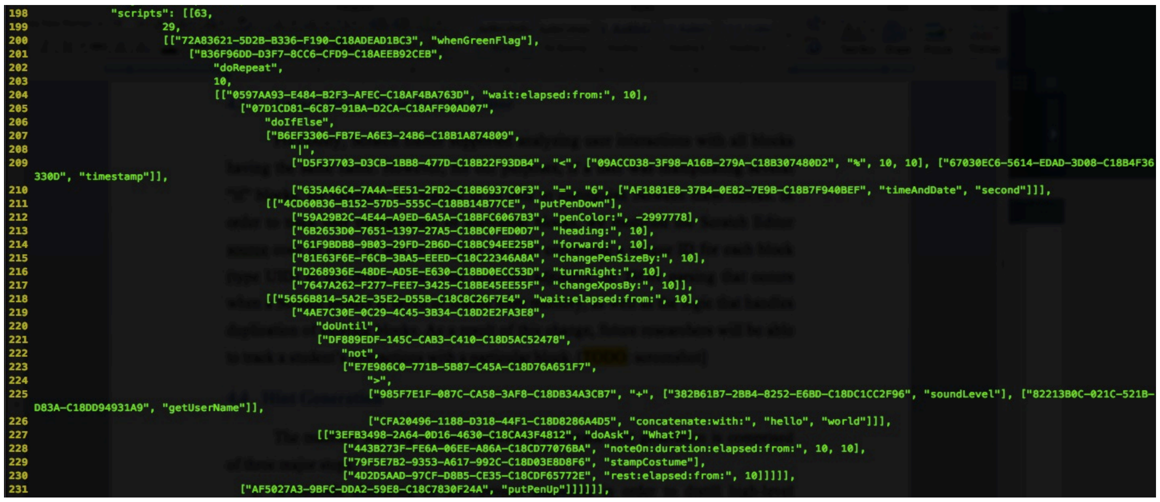
4.2.2 scikit-learn [11]

scikit-learn is Python's open-source library dedicated to data mining. It was leveraged to perform additional feature preprocessing and transformation in between RapidMiner processes.

4.3 Behavioral Data Processing

4.3.1 Unique Block IDs in Scratch Editor

Previously, Scratch Editor did not distinguish between different blocks with the same name. For our purposes, we wanted to change this: if a user manipulated several different "if" blocks, for example, we wanted to be able to distinguish between them. In order to track user manipulation of individual blocks, we added an instance variable to Scratch Editor's Block class that stores a unique block ID of type String, generated through ActionScript's UIDUtil class. This involved adjusting the JSON parsing that occurs when a Scratch project is saved or loaded from memory, as well as the logic that handles duplication of existing blocks. As a result of this change, future researchers will be able to track a student's interactions with a particular block.

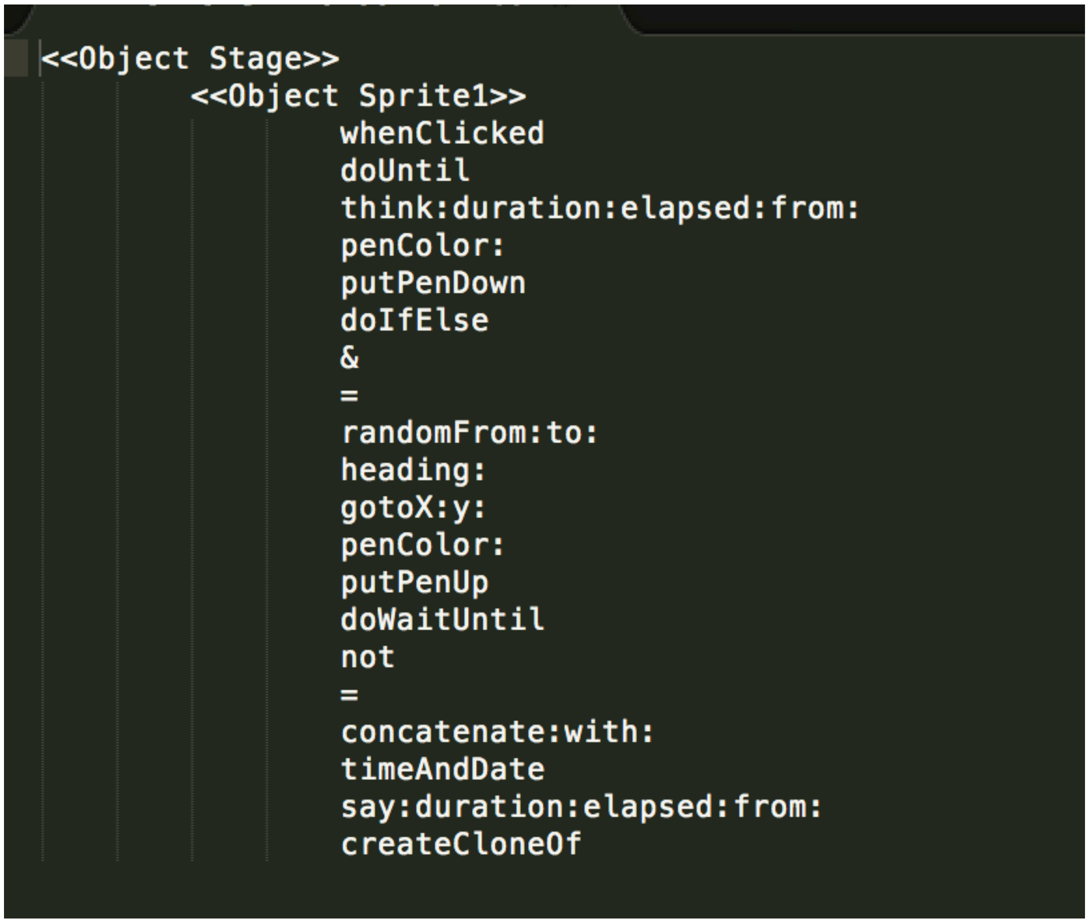


```
"scripts": [[63,
        29,
        [["72A83621-5D2B-B336-F190-C18ADEAD1BC3", "whenGreenFlag"],
            ["B36F96DD-D3F7-8CC6-CFD9-C18AEEB92CEB",
                "doRepeat",
                10,
                [["0597AA93-E484-B2F3-AFEC-C18AF4BA763D", "wait:elapsed:from:", 10],
                    ["07D1CD81-6C87-91BA-D2CA-C18AFF90AD07",
                        "doIfElse",
                        ["B6EF3306-FB7E-A6E3-24B6-C18B1A874809",
                            "|",
                            ["D5F37703-D3CB-1BB8-477D-C18B22F93DB4", "<", ["09ACCD38-3F98-A16B-279A-C18B307480D2", "%", 10, 10], ["67030EC6-5614-EDAD-3D08-C18B4F36330D", "timestamp"]],
                            ["635A46C4-7A4A-EE51-2FD2-C18B6937C0F3", "=", "6", ["AF1881E8-37B4-0E82-7E9B-C18B7F940BEF", "timeAndDate", "second"]]],
                        [["4CD60B36-B152-57D5-555C-C18BB14B77CE", "putPenDown"],
                            ["59A29B2C-4E44-A9ED-6A5A-C18BFC6067B3", "penColor:", -2997778],
                            ["6B2653D0-7651-1397-27A5-C18BC0FED0D7", "heading:", 10],
                            ["61F9BDB8-9B03-29FD-2B6D-C18BC94EE25B", "forward:", 10],
                            ["81E63F6E-F6CB-3BA5-EEED-C18C22346A8A", "changePenSizeBy:", 10],
                            ["D268936E-48DE-AD5E-E630-C18BD0ECC53D", "turnRight:", 10],
                            ["7647A262-F277-FEE7-3425-C18BE45EE55F", "changeXposBy:", 10]],
                        [["5656B814-5A2E-35E2-D55B-C18C8C26F7E4", "wait:elapsed:from:", 10],
                            ["4AE7C30E-0C29-4C45-3B34-C18D2E2FA3E8",
                                "doUntil",
                                ["DF889EDF-145C-CAB3-C410-C18D5AC52478",
                                    "not",
                                    ["E7E986C0-771B-5B87-C45A-C18D76A651F7",
                                        ">",
                                        ["985F7E1F-087C-CA58-3AF8-C18DB34A3CB7", "+", ["382B61B7-2BB4-8252-E6BD-C18DC1CC2F96", "soundLevel"], ["82213B0C-021C-521B-D83A-C18DD94931A9", "getUserName"]],
                                        ["CFA20496-1188-D318-44F1-C18D8286A4D5", "concatenate:with:", "hello", "world"]]],
                                [["3EFB3498-2A64-0D16-4630-C18CA43F4812", "doAsk", "What?"],
                                    ["443B273F-FE6A-06EE-A86A-C18CD77076BA", "noteOn:duration:elapsed:from:", 10, 10],
                                    ["79F5E7B2-9353-A617-992C-C18D03E8D8F6", "stampCostume"],
                                    ["4D2D5AAD-97CF-D8B5-CE35-C18CDF65772E", "rest:elapsed:from:", 10]]]]],
                    ["AF5027A3-9BFC-DDA2-59E8-C18C7830F24A", "putPenUp"]]]]],
```

Figure 2: In this fragment of a Scratch project JSON snapshot, one can see the unique block IDs as the leftmost element in each list that represents a block. Blocks are represented as lists containing their ID, name, and parameters (if applicable).

4.3.2 Scratch Analyzer Enhancements

Scratch Analyzer [**2**] was originally developed to read in .sb2 files associated with a project and then output its tree and block structure to a .se file. This component of our project built upon the functionality of Scratch Analyzer by adding the ability to parse timestamped JSON files (files representing the state of a project at a particular time) and to compare successive files to identify any changes made in the relative structure of the project. As was explained in section 4.3.1, each block in a project is associated with a unique ID that allows the program to distinguish between blocks with the same name.

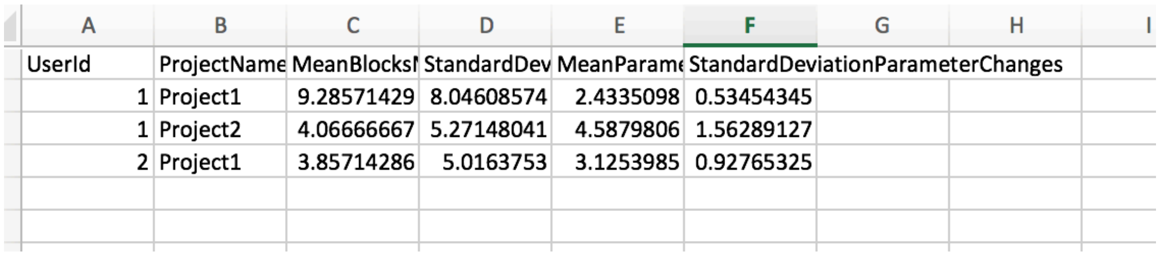


```
<<Object Stage>>
        <<Object Sprite1>>
                whenClicked
                doUntil
                think:duration:elapsed:from:
                penColor:
                putPenDown
                doIfElse
                &
                =
                randomFrom:to:
                heading:
                gotoX:y:
                penColor:
                putPenUp
                doWaitUntil
                not
                =
                concatenate:with:
                timeAndDate
                say:duration:elapsed:from:
                createCloneOf
```

Figure 3: The above screenshot shows the tree structure output in the .se files that is preserved from Bender's original Scratch Analyzer implementation. [2]

The current Scratch Analyzer implementation gives the user the option of parsing either a single .sb2 file associated with a project, or the hundreds of timestamped JSON files associated with that project. If the user chooses to parse data in the form of timestamped JSON files, then Scratch Analyzer maintains the original input hierarchy while outputting the files. Each JSON file is associated with a corresponding .se file that contains data about the tree structure of the blocks it contains.

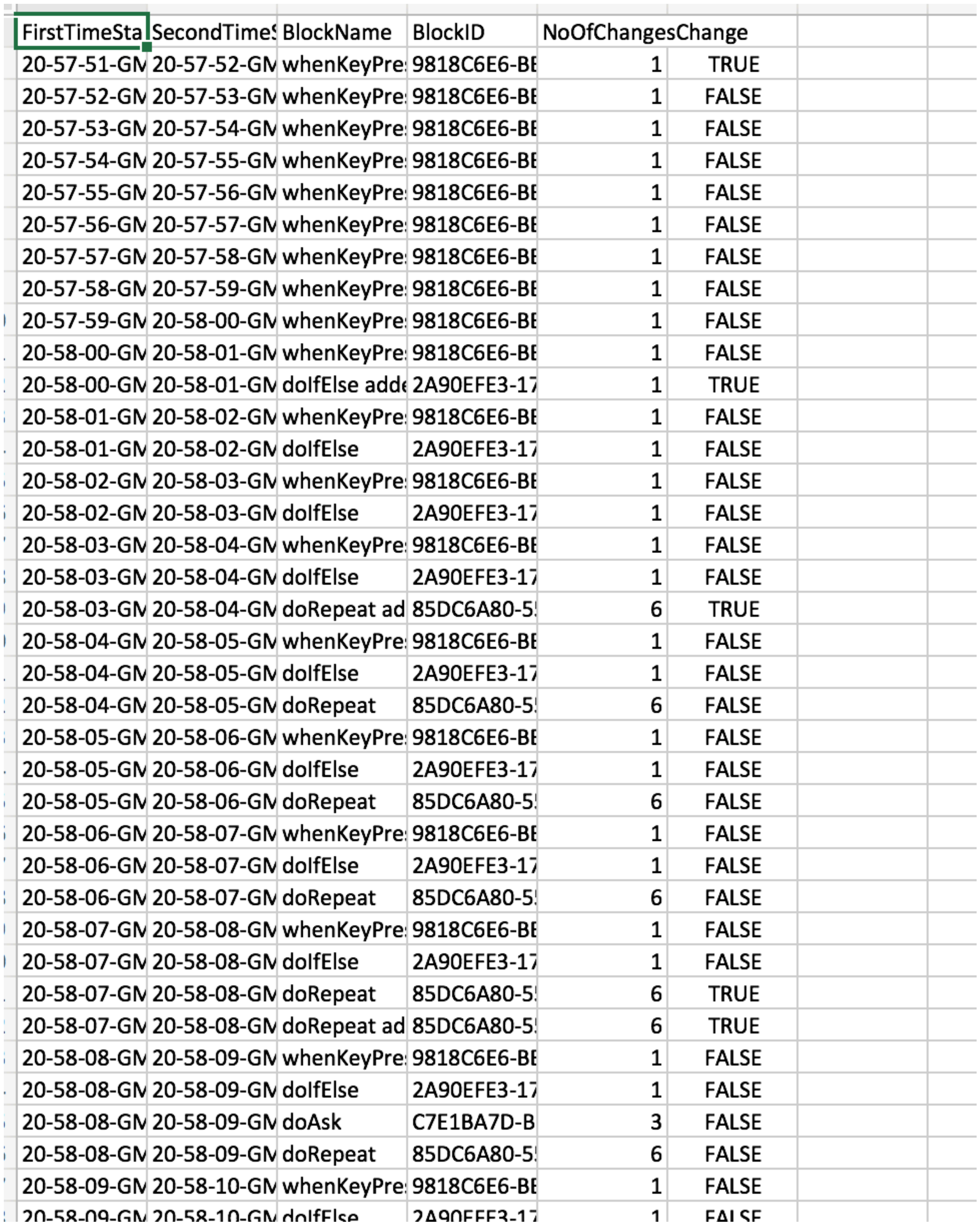
The most important output of the modified Scratch Analyzer is the `StatisticalAnalysis.csv` file outputted in the `OutputCSV` directory. This file contains statistical data about the number of times a block's state is changed in each project associated with each user. It also includes data about block parameter changes. By looking at the output files, like the one shown in Figure 4, one can observe that block movements are much more frequent than parameter changes; this indicates that users are more certain about the information they input into blocks than about block positions. The format of the output will be useful for machine learning algorithms when handling the data of hundreds of users, each with multiple projects under his/her name.



A	B	C	D	E	F	G	H	I
UserId	ProjectName	MeanBlocksI	StandardDev	MeanParame	StandardDeviationParameterChanges			
1	Project1	9.28571429	8.04608574	2.4335098	0.53454345			
1	Project2	4.06666667	5.27148041	4.5879806	1.56289127			
2	Project1	3.85714286	5.0163753	3.1253985	0.92765325			

Figure 4: The above screenshot shows the format of the `StatisticalAnalysis.csv` file.

There is also a per-project .csv output file that gives information about per-block movements and changes between consecutive JSON files. This allows the researcher to retrieve information about when a block was moved and when/if it was dragged back into the palette. Since the output files contain data about the point in a project at which a block was added or removed, they allow researchers to pinpoint details of individual students' behavior.



FirstTimeSta	SecondTimeS	BlockName	BlockID	NoOfChanges	Change
20-57-51-GM	20-57-52-GM	whenKeyPre	9818C6E6-BE	1	TRUE
20-57-52-GM	20-57-53-GM	whenKeyPre	9818C6E6-BE	1	FALSE
20-57-53-GM	20-57-54-GM	whenKeyPre	9818C6E6-BE	1	FALSE
20-57-54-GM	20-57-55-GM	whenKeyPre	9818C6E6-BE	1	FALSE
20-57-55-GM	20-57-56-GM	whenKeyPre	9818C6E6-BE	1	FALSE
20-57-56-GM	20-57-57-GM	whenKeyPre	9818C6E6-BE	1	FALSE
20-57-57-GM	20-57-58-GM	whenKeyPre	9818C6E6-BE	1	FALSE
20-57-58-GM	20-57-59-GM	whenKeyPre	9818C6E6-BE	1	FALSE
20-57-59-GM	20-58-00-GM	whenKeyPre	9818C6E6-BE	1	FALSE
20-58-00-GM	20-58-01-GM	whenKeyPre	9818C6E6-BE	1	FALSE
20-58-00-GM	20-58-01-GM	doIfElse adde	2A90EFE3-17	1	TRUE
20-58-01-GM	20-58-02-GM	whenKeyPre	9818C6E6-BE	1	FALSE
20-58-01-GM	20-58-02-GM	doIfElse	2A90EFE3-17	1	FALSE
20-58-02-GM	20-58-03-GM	whenKeyPre	9818C6E6-BE	1	FALSE
20-58-02-GM	20-58-03-GM	doIfElse	2A90EFE3-17	1	FALSE
20-58-03-GM	20-58-04-GM	whenKeyPre	9818C6E6-BE	1	FALSE
20-58-03-GM	20-58-04-GM	doIfElse	2A90EFE3-17	1	FALSE
20-58-03-GM	20-58-04-GM	doRepeat ad	85DC6A80-5	6	TRUE
20-58-04-GM	20-58-05-GM	whenKeyPre	9818C6E6-BE	1	FALSE
20-58-04-GM	20-58-05-GM	doIfElse	2A90EFE3-17	1	FALSE
20-58-04-GM	20-58-05-GM	doRepeat	85DC6A80-5	6	FALSE
20-58-05-GM	20-58-06-GM	whenKeyPre	9818C6E6-BE	1	FALSE
20-58-05-GM	20-58-06-GM	doIfElse	2A90EFE3-17	1	FALSE
20-58-05-GM	20-58-06-GM	doRepeat	85DC6A80-5	6	FALSE
20-58-06-GM	20-58-07-GM	whenKeyPre	9818C6E6-BE	1	FALSE
20-58-06-GM	20-58-07-GM	doIfElse	2A90EFE3-17	1	FALSE
20-58-06-GM	20-58-07-GM	doRepeat	85DC6A80-5	6	FALSE
20-58-07-GM	20-58-08-GM	whenKeyPre	9818C6E6-BE	1	FALSE
20-58-07-GM	20-58-08-GM	doIfElse	2A90EFE3-17	1	FALSE
20-58-07-GM	20-58-08-GM	doRepeat	85DC6A80-5	6	TRUE
20-58-07-GM	20-58-08-GM	doRepeat ad	85DC6A80-5	6	TRUE
20-58-08-GM	20-58-09-GM	whenKeyPre	9818C6E6-BE	1	FALSE
20-58-08-GM	20-58-09-GM	doIfElse	2A90EFE3-17	1	FALSE
20-58-08-GM	20-58-09-GM	doAsk	C7E1BA7D-B	3	FALSE
20-58-08-GM	20-58-09-GM	doRepeat	85DC6A80-5	6	FALSE
20-58-09-GM	20-58-10-GM	whenKeyPre	9818C6E6-BE	1	FALSE
20-58-09-GM	20-58-10-GM	doIfElse	2A90EFE3-17	1	FALSE

Figure 5: The above screenshot shows how the per-project output is formatted.

4.4 Hint Generation

The machine learning system supporting automatic hint generation is comprised of three major steps:

1. K-medoids clustering of user project snapshots in order to distill high-level project "milestones" through which users may pass
2. K-means clustering of users based on their paths through the milestones identified in step 1
3. Sequential pattern mining (using the Generalized Sequential Pattern (GSP) algorithm) within each cluster generated by step 2 in order to extract hints

The use of clustering algorithms to model student progress as a path through discrete states was influenced by the work of Piech et al. [**5**], and the idea to use GSP was inspired by Romero et al. [**9**]. For our purposes, in the absence of a substantial corpus of intermediary snapshots generated by many users working on the same project, we used mock data generated by a Python script that produced .se files of varying lengths with randomized omissions and repetitions on the basis of a complete, user-created .se file.

Each component of this system is discussed in greater detail below.

4.4.1 Machine Learning Algorithms Used

The three machine learning algorithms described below, as implemented in RapidMiner Studio, comprise our hint generation strategy.

4.4.1.1 K-Medoids Clustering

K-medoids clustering is used to condense a large set of project snapshots, in the form of .se files representing students' partially complete projects, into a set of k_1 discrete milestones through which students may pass as an intermediary step on their way to completing the assignment (k_1 is used here to distinguish the number of clusters generated in this step from the number k_2 of clusters generated in the following step). A corpus of .se snapshots serves as training data for the algorithm. The script `1-k-medoids-features.py` obtains input examples for clustering, using a Bag-of-Words approach for feature extraction. First, it strips each .se snapshot of lines that do not represent Scratch blocks (namely, lines that contain declarations of Sprite objects). Then, treating each block name as a “word” in the context of natural language processing, it uses Python’s scikit-learn library [**11**] in order to extract trigrams of blocks from the snapshots, where a trigram is a sequence of three block names. (The use of bigrams, or sequences of two “words,” instead of trigrams would improve computational speed and may be necessary with the use of a larger corpus. Implementing this change involves simply changing the second parameter to the function `ngram()` to 2 instead of 3 in `1-k-medoids-features.py`.) A set of vectors (one per example) containing the count of occurrences of each trigram per example serves as input to the k-medoids algorithm.

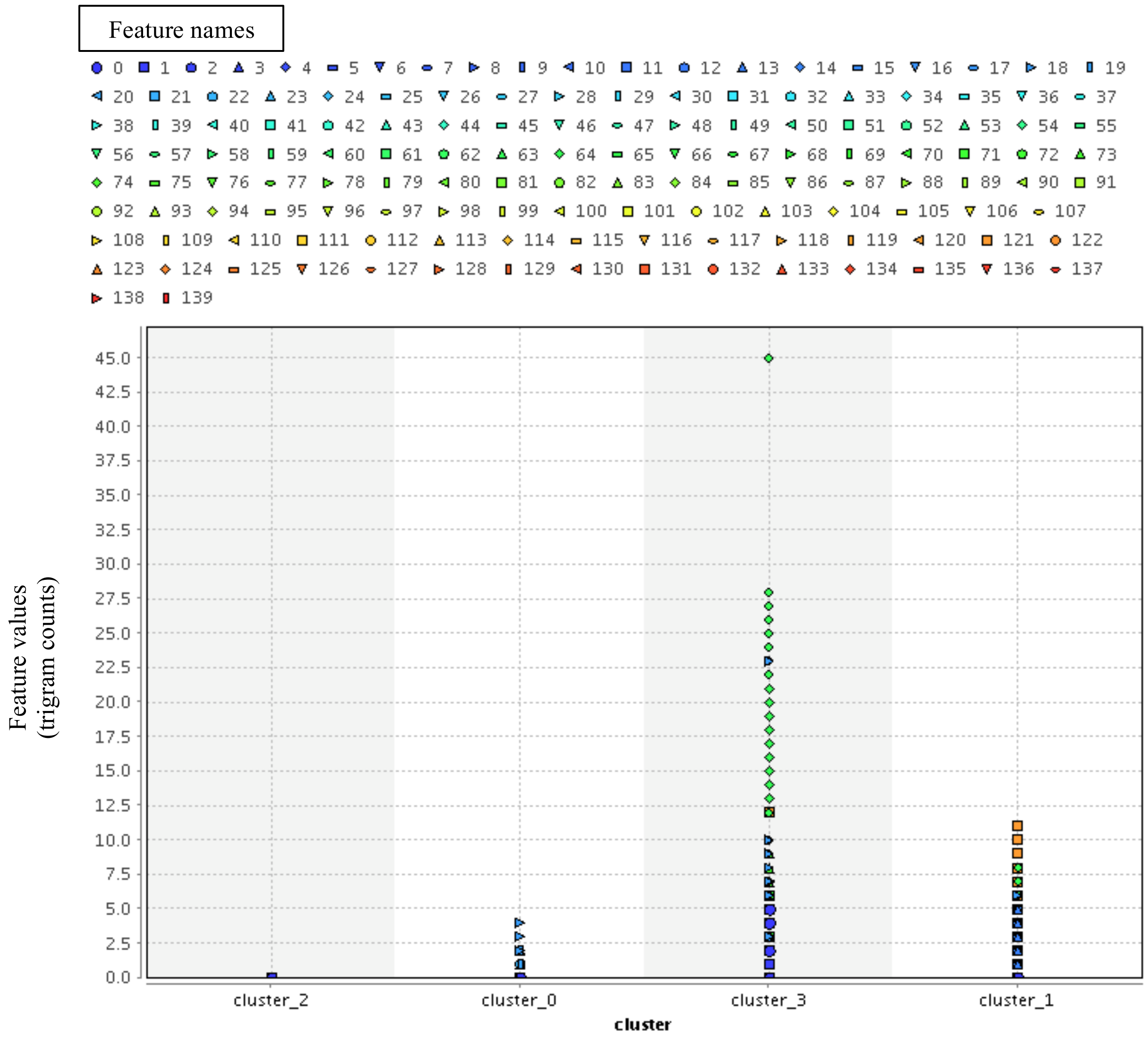


Figure 6: The above scatter plot shows the counts for each of the 140 trigram features in the 569 training examples distributed across the four clusters. One can see that each cluster has a distinct composition of trigram counts.

K-medoids is a clustering algorithm that separates the input data into some fixed number *k* groups. Using a specified distance metric, it assigns each input data point to the cluster centroid closest to it and then minimizes the distance between each point and its centroid as much as possible. Unlike k-means centroids, which are the average of the points in their cluster, k-medoids centroids are the median of their cluster's points; in other words, each k-medoids centroid is itself a member of its cluster, not a fabricated additional data point. This was important for our purposes, since an actual, existing snapshot is the most logical way to represent each discrete milestone.

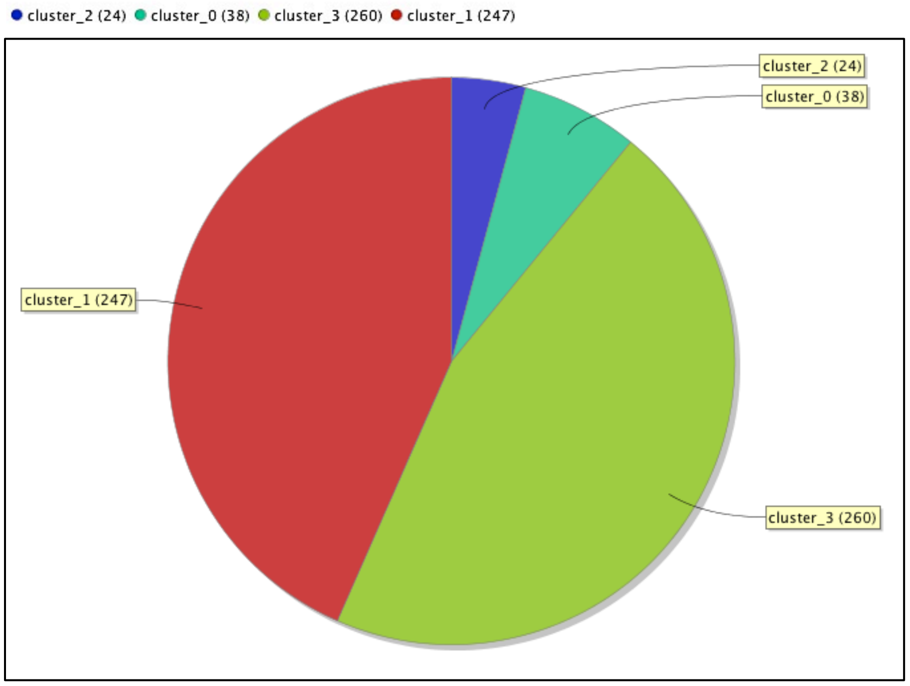
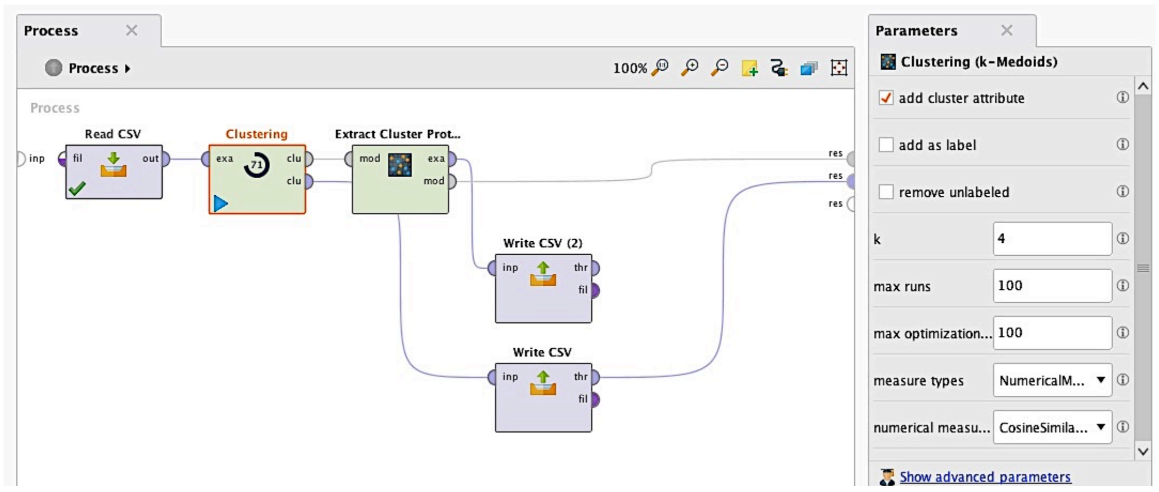


Figure 7: The above pie chart, based on the same input data as the previous scatter plot, shows the fraction of the 569 input examples assigned to each cluster.

After `1-k-medoids-features.py` performs feature preprocessing in Python, the input is fed into the RapidMiner process `milestones-clustering.rmp`. It is currently set to cluster the data into $k_1 = 4$ clusters, meaning four discrete milestones will be identified. Setting *k* is a notorious challenge with clustering algorithms, since there is no universally accepted process for finding the "best" value for *k*; instead, "[c]hoosing *k* is often an *ad hoc* decision based on prior knowledge, assumptions, and practical experience" [**12**]. Since the "best" value for k_1 is contingent on the dataset used, its value ought to be reassessed as future researchers start to work with larger corpuses of real, user-generated data. According to the RapidMiner Documentation, lower values of *k* tend to work better [**13**], but future researchers ought to experiment with slightly larger values of k_1 (i.e., $k_1 = 5, 6, 7$, or 8), especially for more complex projects. To automatically and more objectively set the value of k_1, researchers can also consider the G-means algorithm, which iteratively increases the value of *k* until the data in each cluster appear to follow a Gaussian distribution [**12**], or the X-means algorithm, which selectively adjusts clusters to optimize the Bayesian Information Criterion (BIC) or Akaike Information Criterion (AIC) [**14**].

Cosine Similarity is used as the similarity metric, as it is considered to be among the best similarity measures for text-based clustering [**15**]. It also worked best in practice based on its ability to appropriately separate mock data created on the basis of two different projects.



Clustering (k-Medoids)

- [✓] add cluster attribute
- [ ] add as label
- [ ] remove unlabeled

k	4
max runs	100
max optimization...	100
measure types	NumericalM...
numerical measu...	CosineSimila...

Figure 8: This image shows the k-medoids clustering process running in RapidMiner Studio.

Since the output centroids take the same form as the input examples (i.e., vectors of trigram counts), another step is necessary to match the centroids to the appropriate .se snapshots. After the RapidMiner process is complete, the Python script `1-get-snapshot-centroid.py` finds the .se files that have trigram counts identical to those of each of the cluster centroids.

4.4.1.2 K-Means Clustering

K-means clustering is used to arrange students using SAGE into k_2 groups based on their progression through the project relative to the k_1 milestones identified in the previous step (k_2 is used here to distinguish the number of clusters generated in this step from the number of milestone clusters generated in the previous step). The input to k-means is a vector of integers representing the similarity measures between each of n evenly spaced snapshots from each student's project and each milestone, giving a total of nk_1 features for this step. n is currently set to 10, but it could be changed by modifying `researcherRoutes.js`, where this parameter is currently set, or by hardcoding the desired value into the Python script `2-k-means-features.py`, which handles the generation of the k-means input.

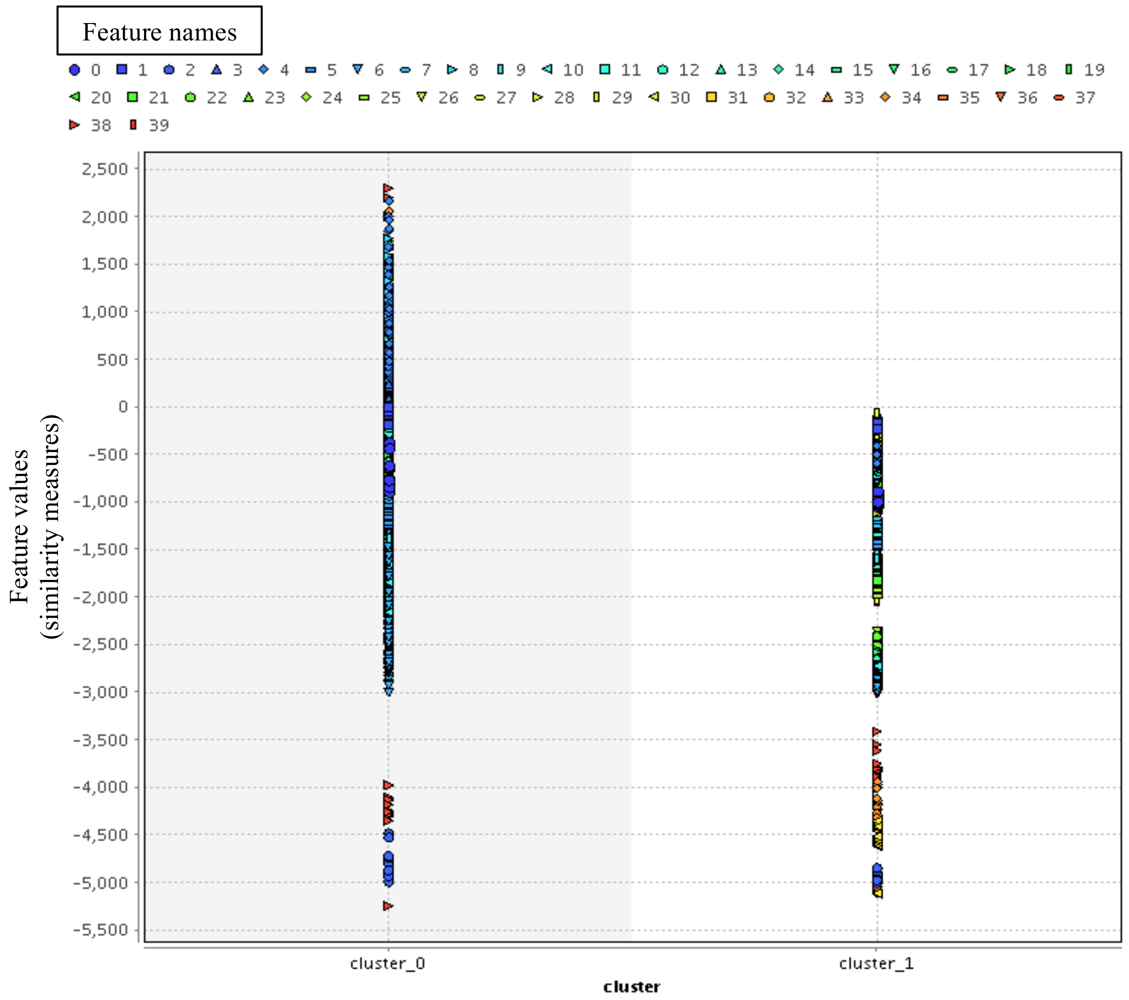
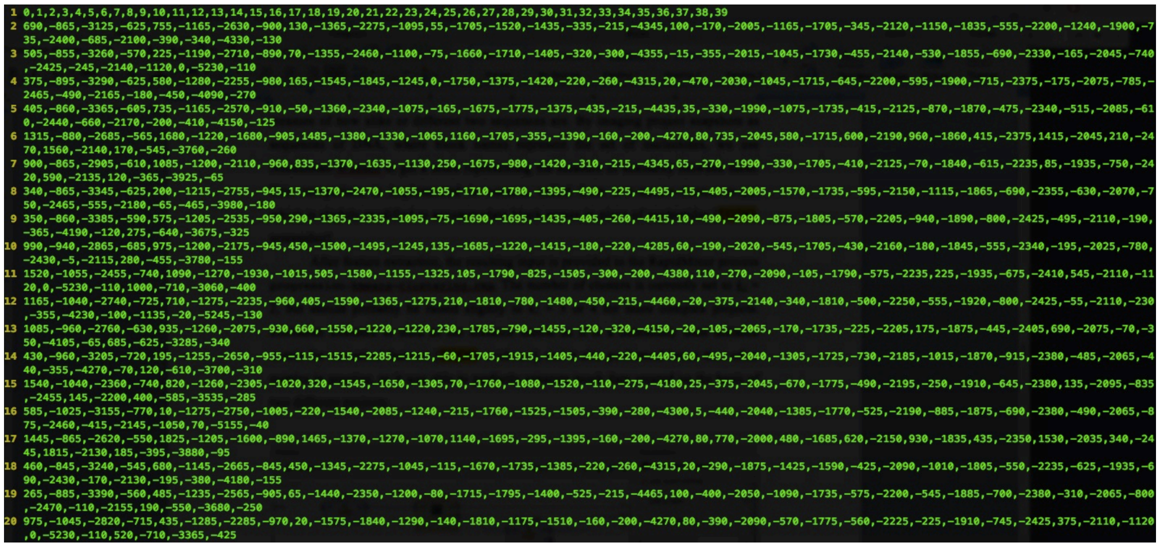


Figure 9: The above scatter plot shows the values for each of the 40 similarity metric features for the training examples within the two clusters. On the y-axis, one can see that the similarity metric values range from less than –5000 to more than +2000, according to the scores calculated by the Needleman-Wunsch algorithm.

The similarity metric used to compare snapshots is a free implementation of the Needleman-Wunsch algorithm available under the GNU General Public License [**16**]. The approach of Piech et al. [**5**] inspired the use of this algorithm, which is generally used for aligning the sequences of nucleotides that comprise DNA. In doing so, it calculates a "score" representing how much change is required to transform one sequence of DNA to match the other, which serves as a good measure of how alike or different two sequences are. By imagining project snapshots as sequences of DNA, where block names represent nucleotides, we use Needleman-Wunsch to get a score representing the measure of similarity between them. This comparison is implemented in `needle`, called from `2-k-means-features.py`, which is slightly modified to accommodate block names in place of nucleotides.

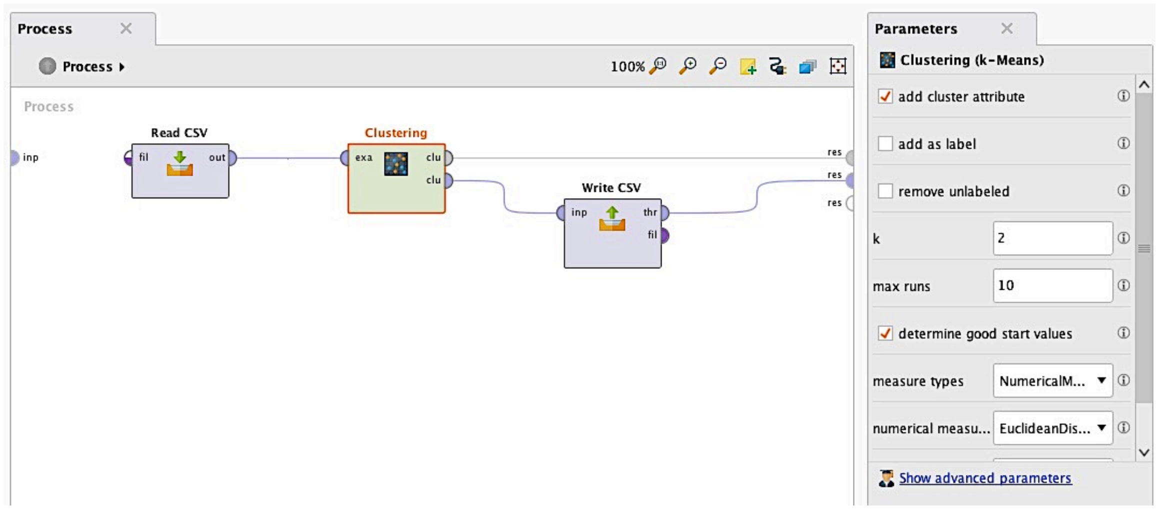


```
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39
690,-865,-3125,-625,755,-1165,-2630,-900,130,-1365,-2275,-1095,55,-1705,-1520,-1435,-335,-215,-4345,100,-170,-2005,-1165,-1705,-345,-2120,-1150,-1835,-555,-2200,-1240,-1900,-7
35,-2400,-685,-2100,-390,-340,-4330,-130
505,-855,-3260,-570,225,-1190,-2710,-890,70,-1355,-2460,-1100,-75,-1660,-1710,-1405,-320,-300,-4355,-15,-355,-2015,-1045,-1730,-455,-2140,-530,-1855,-690,-2330,-165,-2045,-740
,-2425,-245,-2140,-1120,0,-5230,-110
375,-895,-3290,-625,580,-1280,-2255,-980,165,-1545,-1845,-1245,0,-1750,-1375,-1420,-220,-260,-4315,20,-470,-2030,-1045,-1715,-645,-2200,-595,-1900,-715,-2375,-175,-2075,-785,-
2465,-490,-2165,-180,-450,-4090,-270
405,-860,-3365,-605,735,-1165,-2570,-910,-50,-1360,-2340,-1075,-165,-1675,-1775,-1375,-435,-215,-4435,35,-330,-1990,-1075,-1735,-415,-2125,-870,-1870,-475,-2340,-515,-2085,-61
0,-2440,-660,-2170,-200,-410,-4150,-125
1315,-880,-2685,-565,1680,-1220,-1680,-905,1485,-1380,-1330,-1065,1160,-1705,-355,-1390,-160,-200,-4270,80,735,-2045,580,-1715,600,-2190,960,-1860,415,-2375,1415,-2045,210,-24
70,1560,-2140,170,-545,-3760,-260
900,-865,-2905,-610,1085,-1200,-2110,-960,835,-1370,-1635,-1130,250,-1675,-980,-1420,-310,-215,-4345,65,-270,-1990,-330,-1705,-410,-2125,-70,-1840,-615,-2235,85,-1935,-750,-24
20,590,-2135,120,-365,-3925,-65
340,-865,-3345,-625,200,-1215,-2755,-945,15,-1370,-2470,-1055,-195,-1710,-1780,-1395,-490,-225,-4495,-15,-405,-2005,-1570,-1735,-595,-2150,-1115,-1865,-690,-2355,-630,-2070,-7
50,-2465,-555,-2180,-65,-465,-3980,-180
350,-860,-3385,-590,575,-1205,-2535,-950,290,-1365,-2335,-1095,-75,-1690,-1695,-1435,-405,-260,-4415,10,-490,-2090,-875,-1805,-570,-2205,-940,-1890,-800,-2425,-495,-2110,-190,
-365,-4190,-120,275,-640,-3675,-325
990,-940,-2865,-685,975,-1200,-2175,-945,450,-1500,-1495,-1245,135,-1685,-1220,-1415,-180,-220,-4285,60,-190,-2020,-545,-1705,-430,-2160,-180,-1845,-555,-2340,-195,-2025,-780,
-2430,-5,-2115,280,-455,-3780,-155
1520,-1055,-2455,-740,1090,-1270,-1930,-1015,505,-1580,-1155,-1325,105,-1790,-825,-1505,-300,-200,-4380,110,-270,-2090,-105,-1790,-575,-2235,225,-1935,-675,-2410,545,-2110,-11
20,0,-5230,-110,1000,-710,-3060,-400
1165,-1040,-2740,-725,710,-1275,-2235,-960,405,-1590,-1365,-1275,210,-1810,-780,-1480,-450,-215,-4460,-20,-375,-2140,-340,-1810,-500,-2250,-555,-1920,-800,-2425,-55,-2110,-230
,-355,-4230,-100,-1135,-20,-5245,-130
1085,-960,-2760,-630,935,-1260,-2075,-930,660,-1550,-1220,-1220,230,-1785,-790,-1455,-120,-320,-4150,-20,-105,-2065,-170,-1735,-225,-2205,175,-1875,-445,-2405,690,-2075,-70,-3
50,-4105,-65,685,-625,-3285,-340
430,-960,-3205,-720,195,-1255,-2650,-955,-115,-1515,-2285,-1215,-60,-1705,-1915,-1405,-440,-220,-4405,60,-495,-2040,-1305,-1725,-730,-2185,-1015,-1870,-915,-2380,-485,-2065,-4
40,-355,-4270,-70,120,-610,-3700,-310
1540,-1040,-2360,-740,820,-1260,-2305,-1020,320,-1545,-1650,-1305,70,-1760,-1080,-1520,-110,-275,-4180,25,-375,-2045,-670,-1775,-490,-2195,-250,-1910,-645,-2380,135,-2095,-835
,-2455,145,-2200,400,-585,-3535,-285
585,-1025,-3155,-770,10,-1275,-2750,-1005,-220,-1540,-2085,-1240,-215,-1760,-1525,-1505,-390,-280,-4300,5,-440,-2040,-1385,-1770,-525,-2190,-885,-1875,-690,-2380,-490,-2065,-8
75,-2460,-415,-2145,-1050,70,-5155,-40
1445,-865,-2620,-550,1825,-1205,-1600,-890,1465,-1370,-1270,-1070,1140,-1695,-295,-1395,-160,-200,-4270,80,770,-2000,480,-1685,620,-2150,930,-1835,435,-2350,1530,-2035,340,-24
45,1815,-2130,185,-395,-3880,-95
460,-845,-3240,-545,680,-1145,-2665,-845,450,-1345,-2275,-1045,-115,-1670,-1735,-1385,-220,-260,-4315,20,-290,-1875,-1425,-1590,-425,-2090,-1010,-1805,-550,-2235,-625,-1935,-6
90,-2430,-170,-2130,-195,-380,-4180,-155
265,-885,-3390,-560,485,-1235,-2565,-905,65,-1440,-2350,-1200,-80,-1715,-1795,-1400,-525,-215,-4465,100,-400,-2050,-1090,-1735,-575,-2200,-545,-1885,-700,-2380,-310,-2065,-800
,-2470,-110,-2155,190,-550,-3680,-250
975,-1045,-2820,-715,435,-1285,-2285,-970,20,-1575,-1840,-1290,-140,-1810,-1175,-1510,-160,-200,-4270,80,-390,-2090,-570,-1775,-560,-2225,-225,-1910,-745,-2425,375,-2110,-1120
,0,-5230,-110,520,-710,-3365,-425
```

Figure 10: This image shows a fragment of a file containing Needleman-Wunsch scores for a dataset, where there are 40 scores per training example. These are the features provided to the k-means algorithm.

After feature extraction, the resulting input is provided to the RapidMiner process `progression-kmeans-clustering.rmp`. The number of clusters is currently set to $k_2 = 2$, but future researchers should reassess this value when working with student-generated data and experiment with slightly greater values of $k_2 = 3$ or 4 for more complex projects. As with k-medoids clustering, they can also consider using the G-means [**12**] and X-means [**14**] algorithms. Euclidean Distance is used as the distance metric because it is an easy-to-conceptualize, well-established metric for features that are purely numerical [**17**].

It also performed better than other metrics in practice, as it was able to perfectly separate mock data created on the basis of two different projects.



Clustering (k-Means)

- [✓] add cluster attribute
- [ ] add as label
- [ ] remove unlabeled

k	2
max runs	10

- [✓] determine good start values

measure types	NumericalM...
numerical measu...	EuclideanDis...

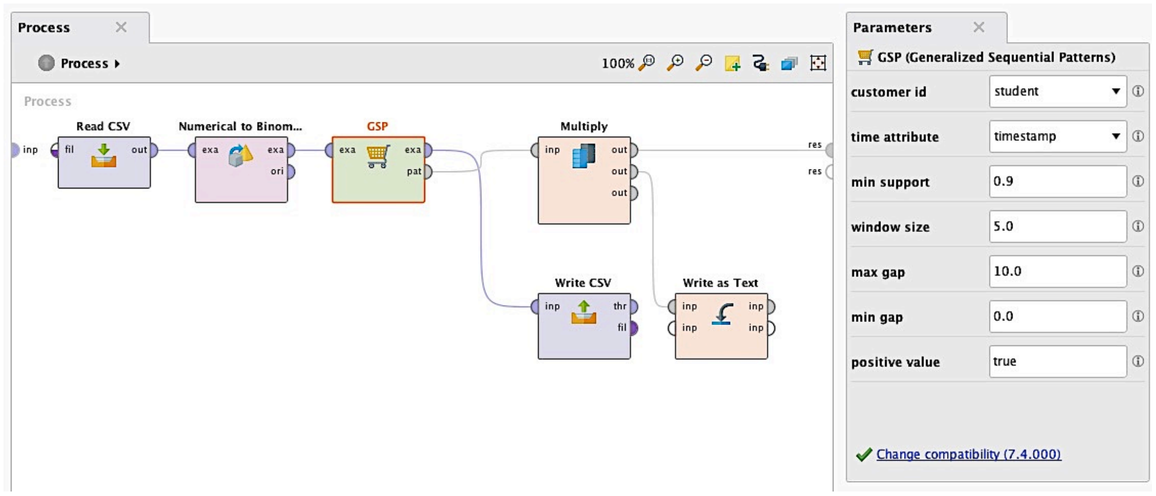
Show advanced parameters

Figure 11: This image shows the k-means clustering process in RapidMiner Studio.

4.4.1.3 Generalized Sequential Pattern (GSP)

We used GSP, a sequential pattern mining algorithm reported to work at least as well as alternative options by Romero et al. [**9**], to mine the data within each cluster for frequently occurring two-block sequences that could be used as rules for hint generation. The features used as input to this algorithm must be in a particular, binomial format. Each file in the solution corpus must therefore be transformed from a .se file to a set of binomial feature vectors, one feature vector per block in the file. Each unique block name is represented by a feature, or column. There is precisely one column per feature vector (row) that has value 1, indicating the block represented by that line of the .se file; all other columns have value 0. The script `transform-kmeans-output-for-gsp.py` achieves this transformation from raw .se files to input examples. The GSP algorithm should be run separately on projects in each of the clusters identified in the previous step to improve the relevance of hints. Only completed, correct solutions that belong to the cluster in question should be used in the input corpus in order to generate useful, high-quality hints. Since hints are generated based on the ensemble of available solution files, the larger the input corpus, the more reliable hints will be. Hints will be least dependable for the first few students working on a new project, as unconventional patterns found in the few available solution files may lead to unreliable hints in their case.

Sequential pattern mining is a technique also used in domains like e-commerce [**18**] where the goal is to seek patterns in a set of "transactions" (or block placements, in our case). Unlike association rule mining in general, sequential pattern mining maintains the importance of the order in which transactions occur by requiring a timestamp feature. For our purposes, the actual times at which blocks are added is not important; we are only concerned with the order they are in. Therefore, mock timestamps are generated in `transform-kmeans-output-for-gsp.py` by simply incrementing the timestamp value for each successive line within the same file. GSP also requires a "customer ID," likewise generated in `transform-kmeans-output-for-gsp.py` as a different ID for each solution file, which is important for our purposes so that only sequences within the same file are considered as potential rules.



customer id: student
time attribute: timestamp
min support: 0.9
window size: 5.0
max gap: 10.0
min gap: 0.0
positive value: true

Figure 12: This image shows the GSP process in RapidMiner Studio. "min support" is set to 0.9.

The RapidMiner process `sequential-pattern-mining.rmp` executes the GSP algorithm. First, the "Numerical to Binomial" RapidMiner operator ensures that the appropriate features are in binomial form. To produce rules, the "GSP" operator only uses itemsets (pairs of blocks) with a support value of at least 0.9, meaning that at least 90% of the input solution files must contain a particular two-block sequence in order for it to be used as a rule. This filters out uncommon patterns, thereby increasing the likelihood that rules will be based on broad strategies and will generalize well to other projects in their cluster. If, in practice, rules turn out to be too unreliable or too sparse, the value of “min support” can be raised or lowered, respectively.

4.4.2 Dashboard Integration

It was a priority of this project to integrate our work into the existing SAGE architecture in a reasonably cohesive manner, rather than tacking it on as an isolated system. Therefore, once the machine learning processes had been designed, they were placed in the new `machine_learning` subdirectory of the Dashboard project (the sage-frontend Git repository). The Dashboard UI was modified to support a third user type: Researcher (in addition to Student and Instructor). This involved changing the toggle switch to a drop-down menu and adjusting the backend code supporting user registration and login.

The Researcher Overview page is closely based on the Overview page for an Instructor. The Scratch Editor ActionScript executable is embedded in the page for easier experimentation with how rules affect the Editor UI. A researcher can simply click on a green button labeled “Enable clustering & hint rule mining” in order to run the sequence of machine learning algorithms described in section 4.4.1 in the background. The scripts are launched by the Dashboard server using the Node.js `exec()` command, which spawns a child process that runs asynchronously, so it does not interrupt the other functions of the server. Through the use of callback functions, all the scripts are run sequentially, waiting for the current one to terminate before the next begins. The commands being run, as well as any errors that may arise, are written to the console from which the Dashboard server was started. When enabled, the full machine learning sequence is re-launched every 15 minutes in order to ensure that the hinting rules are continually updated as new data become available. The user can click the now-red button, labeled “Disable clustering & hint rule mining,” in order to quit running the machine learning scripts in the background.

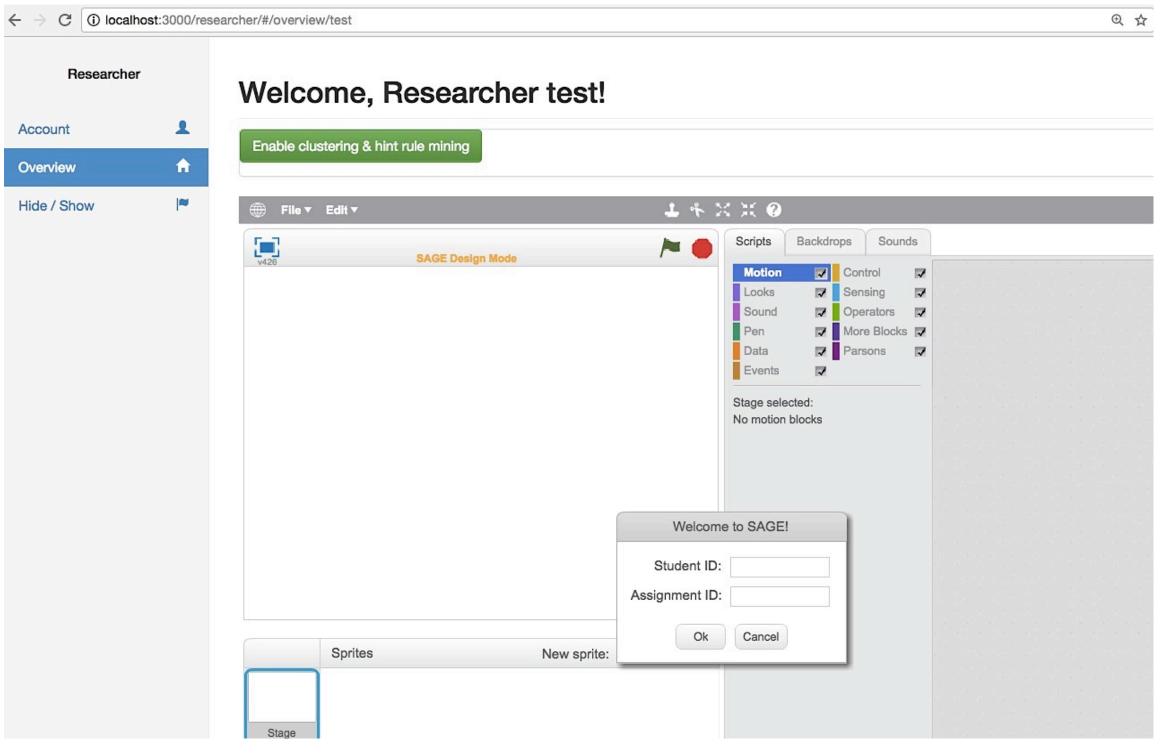


Figure 13: This is the researcher Overview page when machine learning is disabled (default state).

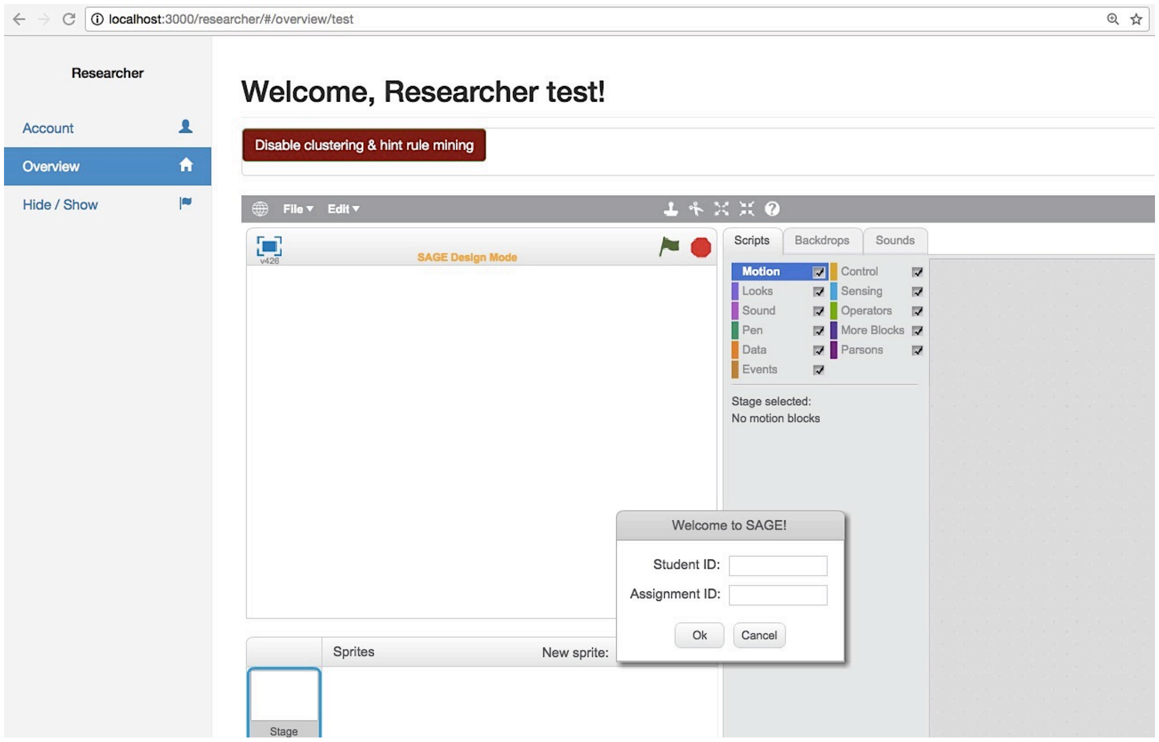


Figure 14: This is the researcher Overview page when machine learning is enabled.

4.4.2.1 Machine Learning Output Reference

As a reference, this section explains the contents and purpose of all files outputted by the machine learning processes to the `machine_learning/ml-output` (for Python script output) and `machine_learning/RapidMiner-output` (for RapidMiner output) directories. The names of these output files can be changed as desired by editing the relevant Python scripts, .rmp files, and `routes/researcherRoutes.js`, as necessary.

- `ml-output/output-k-medioids-features-with-labels-trigram.csv`
  - Outputted by: `1-k-medoids-features.py`
  - Contains: trigram feature vectors to be used by the RapidMiner k-medoids clustering process
- `RapidMiner-output/clustered-set-cosine-4clusters.csv`
  - Outputted by: `milestones-clustering.rmp`
  - Contains: input examples to `milestones-clustering.rmp` followed by the cluster they were classified into, in the rightmost column of each row
- `RapidMiner-output/centroids-cosine-4clusters.csv`
  - Outputted by: `milestones-clustering.rmp`
  - Contains: centroid (i.e., feature vector representing the median file) for each k-medoids cluster, in the format of trigram features (because this was the format of the input examples); `1-get-snapshot-centroid.py` uses the centroid trigram features to identify the names of the actual centroid files
- `ml-output/centroid-output.txt`
  - Outputted by: `1-get-snapshot-centroid.py`
  - Contains: list of filenames of k-medoids centroids (i.e., the files that represent the high-level project milestones)
- `ml-output/output-k-means-features-with-labels.csv`
  - Outputted by: `2-k-means-features.py`
  - Contains: similarity metric feature vectors to be used by the RapidMiner k-means clustering process

- `RapidMiner-output/kmeans-progresssions-2clusters.csv`
    - Outputted by: `progression-kmeans-clustering.rmp`
    - Contains: input examples to `progression-kmeans-clustering.rmp` followed by the cluster they were classified into, in the rightmost column of each row
- `ml-output/gsp-features-0.csv, ml-output/gsp-features-1.csv`
    - Outputted by: `transform-kmeans-output-for-gsp.py`
    - Contain: binomial feature vectors converted from .se files to the necessary format for input to `sequential-pattern-mining.rmp`; there is one for each k-means cluster
- `RapidMiner-output/gsp-data.csv`
    - Outputted by: `sequential-pattern-mining.rmp`
    - Contains: the converted binomial feature vectors used by `sequential-pattern-mining.rmp`, with the input value 1 converted to "true" and the value 0 converted to "false"; this file is not necessary, but is simply outputted for reference
- `RapidMiner-output/gsp-rules-cluster0.res`
    - Outputted by: `sequential-pattern-mining.rmp`
    - Contains: the set of GSP-issued sequential pattern mining rules upon which hints will be based

4.4.3 Scratch Editor User Interface Modifications

In order to determine if a hint is currently available, it is necessary to identify the latest block manipulated by the user: if the latest block falls on the left side of one or more of the GSP rules, the corresponding right block(s) should be hinted; if it does not, there is no hint available. In order to keep track of the latest block manipulated, we added two static variables to Scratch Editor: *latestBlock* (a Block) and *latestBlockList* (an array of Blocks). *latestBlockList* keeps track of all blocks that have filled the role of *latestBlock*, in order. *lastestBlock* is updated if a new block is dragged from the block palette to the scripts pane, if a block is moved around within the scripts pane, if the parameters of a block within the scripts pane are modified, and if one or more blocks are removed from the scripts pane. In the case that blocks are removed, *latestBlock* reverts to the most recent block in *latestBlockList* that is not part of the removed stack. Blocks are deleted from *latestBlockList* as they are removed from the scripts pane, so that if there are no blocks in the scripts pane, *latestBlock* is null and there is thus no possibility of a hint.

Regarding the question of how to visually implement hinting in Scratch Editor, we determined that shaking blocks was a gameful and clean way to issue hints (as opposed to displaying a text hint). Shaking blocks allows for the possibility of shaking multiple blocks at once (which typically only occurs when the hinted blocks are closely related). It also constitutes a less definitive hinting method than text, which is beneficial because even though we take precautions like clustering and requiring a fairly high minimum support of 0.9, there is no guarantee that a hint will be relevant to a particular student's current project state. However, in many cases, there are multiple ways to solve a problem, and even if the shaken block does not represent the only way to complete the assignment, it may nonetheless help point the student in the correct direction. For example, if the "less than" block is hinted, this indicates that the user has just manipulated a control block that needs to be filled with a condition; this is useful for the student to think about, even if an operator other than "less than" is preferable.

If the student is already in the same category as the block to be hinted, so that this block is already visible to him/her, the block is simply shaken. If s/he is not in the same category as the block to be hinted, the category is first shaken and highlighted in two flashes of yellow. Then, if s/he switches to the correct category, the precise block will be shaken.

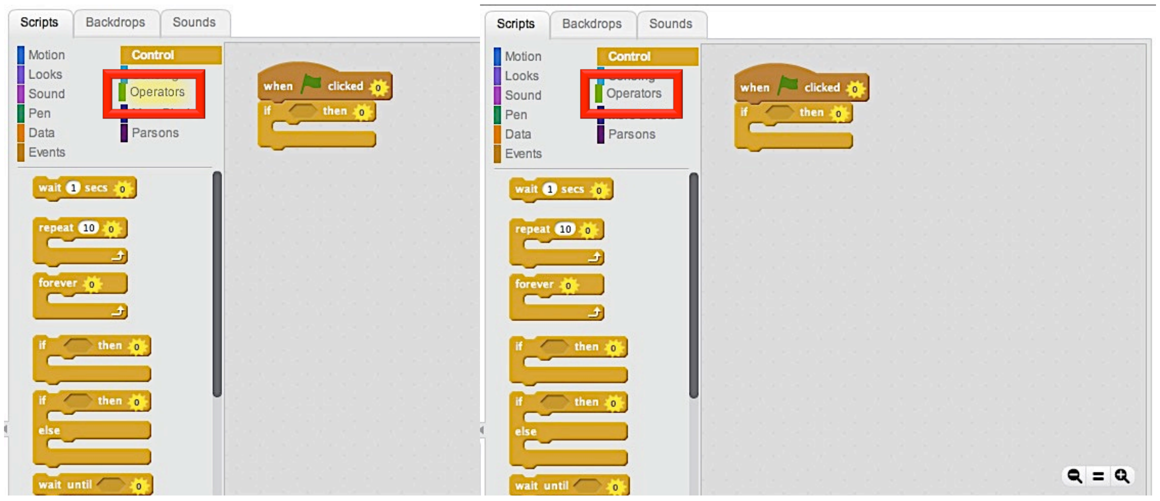


Figure 15: The "Operators" class is hinted when the latest block is "if-then," so it flashes a yellow highlight and shakes by moving slightly back and forth.

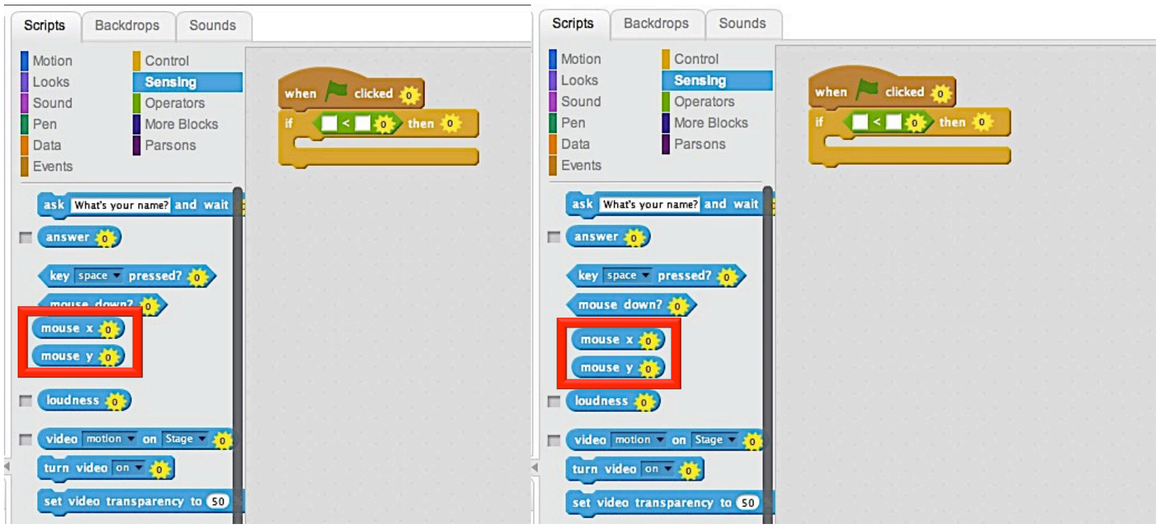


Figure 16: When the latest block manipulated is "less than," both "mouse x" and "mouse y" are hinted by shaking them.

Scratch Editor currently waits five seconds after the user's most recent move before issuing a hint, if one is available. This wait period may need to be augmented if, in practice, it encourages "gaming the system," or waiting idly until the next hint is issued. However, we expect the issue of "gaming the system" will be mitigated by the fact that hints will not be available for every latest move.

5 Limitations and Assumptions

This section describes some of the assumptions made that future researchers working on this project should be aware of. It also discusses the practical limitations of the current hinting approach.

5.1 Automatically Generated Training Data

The data upon which our machine learning algorithms were trained and tested were generated by a Python script that reads in a complete, user-created .se file and uses this as the basis for creating "progressions" of .se files of varying lengths with randomized omissions and repetitions. The resulting body of data consists of varied, unique files that nonetheless contain closely related patterns of blocks, with the hope that this would serve as a reasonable approximation for the work of numerous students on the same project. However, there is no guarantee that this approach sufficiently mimics the variations to be expected between .se snapshots manually created by many different students.

5.2 Assumptions Made for Dashboard Integration

Since the RapidMiner processes run from the Dashboard using the shell script must be stored in the user's local RapidMiner repository, they must contain absolute paths to the files they use. These paths are currently hardcoded and must be changed for future use by other users, as described in step 7 of the RapidMiner Setup Details. The Dashboard, particularly `sage-frontend/routes/researcherRoutes.js`, contains pathnames based on the directory structure of the sage-frontend Git repository, so this structure should be maintained.

In the script `transform-kmeans-output-for-gsp.py`, it is necessary to know the ID of each unique student solution file, so that only patterns occurring within the same file will be mined by GSP. Currently, the script assumes that each student's unique solution is located in a directory named after his/her ID (e.g., `0/example.se` is assigned the ID '0'). This assumption is marked in `transform-kmeans-output-for-gsp.py` with a "NOTE" tag.

The retrieval of complete, correct student solution files will also need to be handled when real submissions are being used instead of mock data. Currently, the *data_dir* variable in `researcherRoutes.js` is hardcoded to the value "machine_learning/sample_data", where mock data is stored. This will eventually need to be modified to the location of actual student submissions after they have been assessed and deemed correct.

5.3 Limitations of Hinting Strategy

The hinting strategy implemented in this project is based on two-block sequences. It does not, therefore, account for patterns spanning across more than two blocks. Hints are issued automatically, and there is currently no support for hinting on demand (i.e., a "Give me a hint" button).

Additionally, hinting does not currently take advantage of project state information. This is because state information is lost in the files used to generate the hints: namely, completed solution files. Therefore, our current approach ignores the student's current state and treats all portions of the solution files equally when determining hints to issue. In order to issue hints more relevant to a student's particular project state at a given moment, hinting could be based not just on completed solution files, but on the full progression of .se files leading up to each solution. In this case, the system would need to continually analyze each student's project state, match it with the most similar state(s) in the solution progressions, and then generate hinting rules based only on these particular, intermediary snapshots from the solution progressions.

5.4 RapidMiner Issues

Porting RapidMiner processes from one environment to another can be tedious because RapidMiner processes must be stored in the local repository of each system on which they are run. Refer to the RapidMiner Setup Details for more information.

RapidMiner is also quite demanding regarding input configuration. If the input file is changed, RapidMiner expects the number and types of input features to remain the same. See section 4.2.1.2 for more details on how to handle this.

6 Future Work

This section suggests additional lines of research that future SAGE researchers interested in machine learning could pursue to expand and ameliorate our project. Much room for improvement remains in the pursuit of behavioral analysis and comprehensive, accurate, and situationally appropriate hinting in SAGE.

6.1 Improving and Expanding the Hinting System

6.1.1 Personalizing Hint Frequency and Granularity

Hints are currently issued, when available, to all students in the same manner, regardless of a student's past experience or competence with the topic in question. It would be a vast improvement to implement modulation of hint frequency based on the user's interactions with the current project (e.g., starting to hint only after *x* incorrect moves). Hinting could also be adapted based on external information about the student (discussed in section 6.1.2 below) or his/her affective state, which would require constructing an affect detection system (discussed in section 6.3).

The block-shaking method of hinting poses challenges for the modulation of hint specificity, since the shaking of a particular block necessarily sends the specific message to use that block. However, a few modified shaking methods come to mind: for more advanced learners, future researchers could consider only providing category-level hints (not specific block hints), shaking several incorrect "red-herring" blocks in addition to the correct one, or personalizing the length of the delay between the student's latest move and the issue of a hint. Future researchers could also use the existing hinting system to auto-generate more general text hints that do not mention a specific block to use next (e.g., "How will your 'while' loop know when to end?")

Another potential future research area is teaching the system to recognize specific errors commonly made in Scratch in order to generate more granular error-specific feedback, as described by VanLehn [**19**]. VanLehn's approach involves determining precisely which misunderstanding led the student to take an incorrect action and correcting that misunderstanding itself, instead of just telling the student what to do, with the hope that the student will independently avoid making the same mistake in the future.

6.1.2 Integrating External Information on Student Knowledge/Ability

In support of the above hint improvement proposals, future researchers could explore the possibility of integrating information about students' knowledge and competence from other areas of SAGE into the hinting system. This information includes Dashboard data about the other courses students have taken (potential past exposure to the topics at hand), their performance in past courses that address the topics at hand (comfort level with these topics, if applicable), and their performance on past projects in general (comfort level with programming in SAGE). Since the existing machine learning system is integrated with the Dashboard, it will hopefully be straightforward to relay additional information from the Dashboard to the machine learning algorithms.

6.1.3 Supporting On-Demand Hinting

Many well-constructed hinting systems, such as Andes [**8**], give the user the option of clicking on a hint button to receive on-demand feedback in addition to issuing unprompted hints where appropriate. Our system currently only supports unprompted hinting. One obstacle to providing on-demand hints with the current system is that a hint is not available in every project state, especially for students who are among the first to work on a new project. However, one way to work around this would be to revert to the *latestBlockList* variable to see if any hints are available based on the previous two, three, or four moves. If the list also does not lead to hinting opportunities, clicking the button could simply return, "Try moving some blocks around first." Given the well-established concern about students "gaming the system" by abusing the hint button [**20**], some penalty for using the hint button should be put in place to discourage this behavior.

6.2 Collecting Mouse Data

Further work can include gathering data about students' mouse activity and parsing it in Scratch Analyzer, in addition to the block data currently being collected. To implement this feature, Scratch Editor would need to be modified to output student mouse activity in the project JSON files.

6.3 Affect Detection System

Future researchers could expand upon the progress made in processing student behavioral data in Scratch Analyzer by attempting to link these behaviors to affective states such as frustration, excitement, and boredom. Some combination of the following features could be used in an affect detection clustering algorithm:

- Mean and standard deviation (over all blocks used) of the number of times the student has moved each individual block
- Mean and standard deviation (over all blocks used) of the number of times the student has modified the parameters of each individual block
- Mean and standard deviation of time elapsed since the student's last action
- Number of actions a student has taken in the last *x* seconds, for any number of fixed intervals
- Number of times a block is removed from the scripts area

The resulting clusters would have to be assessed empirically to see if the students sorted into each group did indeed share behavioral traits. If researchers were able to develop reliable detectors of particular affective states, these data could be relayed to the hinting system in order to change the frequency or specificity of hints with the hope of maintaining a state of flow for the student.

6.4 Multi-Puzzle Models

Our k-means clustering of student progressions through a single puzzle at a time could be expanded to create broader, aggregate models of student progressions through multiple puzzles (i.e., "courses" or "quests"). Aggregate models could reveal to teachers which concepts students in general understand best and which will require more attention. A model accounting for courses/quests, rather than just one puzzle at a time, would also enable analysis of student performance on one assignment in order to generate an outer-loop recommendation of which puzzles s/he should work on next and which concepts s/he should practice further.

6.5 Additional k-Means Features

The current implementation of k-means, applied in the second step of the hinting system, only uses features obtained by measuring the similarity between project milestones and student snapshots using the Needleman-Wunsch algorithm. In order to experiment with clustering students for purposes other than the issue of hints (e.g., to compare behavioral characteristics), future researchers could add features to k-means or run the algorithm with an entirely different feature set. If researchers prefer to add features to the existing ones, we would recommend either reducing the number of or assigning a lower weight to the similarity metric features so that they do not overpower the effect of the newly added ones.

6.6 Studies With Real Student Users

Obtaining true, student-generated training data to input into our hinting system will require having a group of students from the target sixth- to eighth-grade range try out SAGE. Student-created data will likely provide insights that can be used to strength the current hinting system. Feedback from these students will also be necessary to empirically assess the usefulness and validity of our hinting approach.

7 Conclusion

From the research phase to initial project planning to the details of our implementation, this report has discussed the process by which we added a machine learning-based hinting system to SAGE and enhanced its user data collection functionality. There is vast opportunity for future work to expand upon this project and construct a more comprehensive and advanced hinting system, and to apply the additional user data collected to the analysis of student behavior. Machine learning approaches carry great advantages for game-based educational platforms in that they allow researchers to leverage existing data in order to gain new pedagogical insights, and they can be used to develop systems that aid students without increasing the demands on teachers. We therefore hope that our work has laid a foundation that will inspire further machine learning applications within SAGE.

8 References

[**1**] J. Bender, "Tooling Scratch: Designing a Collaborative Game-Based Learning System to Infuse Computational Thinking within Grade 6-8 Curricula," Columbia University, New York, NY, 2015.

[**2**] J. Bender, "Scratch Analyzer: Transforming Scratch Projects into Inputs Fit for Educational Data Mining and Learning Analytics," Columbia University, New York, NY, 2014.

[**3**] K.H. Koh, A. Basawapatna, H. Nickerson, and A. Repenning, "Real Time Assessment of Computational Thinking," in *2014 IEEE Symposium on Visual Languages and Human-Centric Computing*, Melbourne, Australia, 2014, pp. 49-52.

[**4**] M.D. Kickmeier-Rust and D. Albert, "Micro-adaptivity: protecting immersion in didactically adaptive digital educational games," in *Journal of Computer Assisted Learning*, 26. University of Graz, Graz, Austria: Blackwell Publishing Ltd., 2010, pp. 95-105.

[**5**] C. Piech, M. Sahami, D. Koller, S. Cooper, and P. Blikstein, "Modeling How Students Learn to Program," Stanford University, Stanford, CA, ACM, 2012.

[**6**] S. Kardan and C. Conati, "A Framework for Capturing Distinguishing User Interaction Behaviours in Novel Interfaces," in *Proceedings of the 4th International Conference on Educational Data Mining*, Eindhoven, the Netherlands, 2011, pp. 159-168.

[**7**] R.C. Murray, K. Vanlehn, and J. Mostow, "Looking Ahead to Select Tutorial Actions: A Decision-Theoretic Approach," in *International Journal of Artificial Intelligence in Education*, 14. 2004, pp. 235-278.

[**8**] K. VanLehn, C. Lynch, K. Schulze, J.A. Shapiro, R. Shelby, L. Taylor, D. Treacy, A. Weinstein, and M. Wintersgill, "The Andes Physics Tutoring System: Lessons Learned," in *International Journal of Artificial Intelligence in Education*, 15(3). 2005.

[**9**] C. Romero, S. Ventura, A. Zafra, and P. de Bra, "Applying Web usage mining for personalizing hyperlinks in Web-based adaptive educational systems," in *Computers & Education*, 53. Elsevier Ltd., 2009, pp. 828-840.

[**10**] "RapidMiner Studio," 2017. https://rapidminer.com/products/studio.

[**11**] "scikit-learn," 2016. http://scikit-learn.org.

[**12**] G. Hamerly and C. Elkan, "Learning the *k* in *k*-means," in *Advances in Neural Information Processing Systems 16 (NIPS 2003)*, Vancouver, British Columbia, Canada, 2003.

[**13**] RapidMiner Documentation, “K-Medoids (RapidMiner Studio Core),” 2017. http://docs.rapidminer.com/studio/operators/modeling/segmentation/k_medoids.html.

[**14**] D. Pelleg and A. Moore, “*X*-means: Extending *K*-means with Efficient Estimation of the Number of Clusters,” Carnegie Mellon University, Pittsburgh, PA, 2000.

[**15**] A. Strehl, J. Ghosh, and R. Mooney, “Impact of Similarity Measures on Web-page Clustering,” in *AAAI Technical Report WS-00-01*, The University of Texas at Austin, Austin, TX, 2000.

[**16**] A. Levchuk, “Pairwise string alignment in Python (Needleman-Wunsch and Smith-Waterman algorithms),” 2011. GitHub repository, https://github.com/alevchuk/pairwise-alignment-in-python.

[**17**] A.K. Jain and R.C. Dubes, “Data Representation,” in *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice-Hall, 1988, pp. 15.

[**18**] S.C. Hsueh, M.Y. Lin, and C.L. Chen, “Mining Negative Sequential Patterns for E-commerce Recommendations,” *2008 IEEE Asia-Pacific Services Computing Conference*, Yilan, 2008, pp. 1213-1218.

[**19**] K. VanLehn, “The Behavior of Tutoring Systems,” University of Pittsburgh, Pittsburgh, PA, 2006.

[**20**] R.S.J.d. Baker, A.T. Corbett, I. Roll, and K.R. Koedinger, “Developing a Generalizable Detector of When Students Game the System,” Human-Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, PA, 2008.