Mining Code Submissions to Elucidate Disengagement in a Computer Science MOOC

Efrat Vinker
Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv, Israel
efratvinker@gmail.com

Amir Rubinstein
Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv, Israel
amirr@tau.ac.il

ABSTRACT

Despite the growing prevalence of Massive Open Online Courses (MOOCs) in the last decade, using them effectively is still challenging. Particularly, when MOOCs involve teaching programming, learners often struggle with writing code without sufficient support, which may increase frustration, attrition, and eventually dropout. In this study, we assess the pedagogical design of a fresh introductory computer science MOOC. Keeping in mind MOOC "end-user" instructors, our analyses are based merely on features easily accessible from code submissions, and methods that are relatively simple to apply and interpret. Using visual data mining we discover common patterns of behavior, provide insights on content that may require reevaluation and detect critical points of attrition in the course timeline. Additionally, we extract students' code submission profiles that reflect various aspects of engagement and performance. Consequently, we predict disengagement towards programming using classic machine learning methods. To the best of our knowledge, our definition for attrition in terms of disengagement towards programming is novel as it suits the unique active hands-on nature of programming. To our perception, the results emphasize that more attention and further research should be aimed at the pedagogical design of hands-on experience, such as programming, in online learning systems.

CCS CONCEPTS

• **Social and professional topics** → Professional topics; Computing education; Computing education programs; Computer science education; CS1; • **Applied computing** → Education; E-learning; • **Human-centered computing** → Visualization; Visualization application domains; Visual analytics; • **Computing methodologies** → Machine learning; Learning paradigms; Supervised learning; Supervised learning by classification.

KEYWORDS

Introductory computer science education, code analysis, automated tutoring systems, massive open online courses (MOOCs), learning analytics, educational data mining, machine learning

ACM Reference Format:
Efrat Vinker and Amir Rubinstein. 2022. Mining Code Submissions to Elucidate Disengagement in a Computer Science MOOC. In *LAK22: 12th International Learning Analytics and Knowledge Conference (LAK22), March 21–25, 2022, Online, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3506860.3506877

1 INTRODUCTION AND RELATED WORK

1.1 Massive Open Online Courses

Distance learning in the form of Massive Open Online Courses (MOOCs) was introduced into our lives a decade ago, and the COVID-19 pandemic increased their popularity in an unprecedented way [1]. MOOCs provide free of charge, open access, high-scale online structured courses. The modern MOOCs conserve the conventional pedagogical model of a 'one-to-many' relationship [2, 3]. They typically include a combination of experts' short video lectures, reading material, exercises, quizzes, and forums. To enable large-scale audiences, exercises and varied activities must be automatically evaluated.

Numerous challenges surrounding MOOCs have been reported. From a learner's point of view, MOOCs seem to require a high degree of self-regulation and autonomy [4, 5]. Researchers have also identified that the feeling of isolation is a major source of disengagement and dropout [6]. Therefore, it is not surprising that MOOCs generally suffer from high dropout rates. Despite the high numbers of enrollees, the reported completion rate is low, e.g., below 13% [7]. Other reported reasons for dropout include lack of (1) ability to devote time to study, (2) real-time support while struggling with course material, and (3) motivation [8, 9]. Another challenge, from the education research point of view, is dishonesty, where learners use different unauthorized methods to improve their performance and chances of earning a certificate or academic credit [10]. For example, studies report that 3%-10% of certificate earners are suspected as cheaters [11].

As MOOC platforms record information on users' interaction with the courseware, they provide massive datasets that can be used to study the behavior of learners. Coupled with the development of "big data" related tools, the educational research community today faces novel opportunities in the form of learning analytics and educational data mining [12, 13], which allow to better understand, design, and optimize learning. Furthermore, educational data analyses are used for predicting learners' behavior, such as attrition [14], academic success [15, 16], and self-regulated learning [17]. Since the massive nature of MOOCs does not allow attention to individuals, these predictions enable targeted intervention (e.g., for those who need assistance).

1.2 Learning Analytics in CS (Computer Science) and Programming Education

Computers have become a core part of our lives. With their growing impact on society, CS-centered programs have become more abundant for different types of audiences, starting as early as kindergarten [18, 19]. It is often claimed that basic computational literacy is a skill everyone should acquire in the 21st century. Yet, there are intense research efforts and discussions in the CS education community about the structure and topics of CS programs in general and introductory (often termed "CS101") curricula in particular. The CS computing curricula report [20] describes different pedagogical models for their advantages and disadvantages, such as "programming-first" [21] and "breadth-first" [22]. Regardless, one of the most debated questions is about the role of programming in such courses.

The first steps in the field of CS and particularly in programming pose various challenges. When learning occurs in online systems, such as MOOCs, the students face further challenges, as the lack of communication and feedback from the teaching staff and other learners [23]. Some courses provide automated feedback and grading for programming assignments [24, 25]. While the feedback is delivered immediately, it does not always meet the students' needs for more detailed explanations [26]. These difficulties may increase students' attrition and disengagement toward an already challenging, and often highly frustrating topic such as programming. Thus, when developing CS101 MOOCs that include programming, attention should be given to unique pedagogical design issues, such as providing effective hands-on experience, interaction with the learning platform and IDE (integrated development environment), the timing and extent of feedback on code, and cognitive overload.

Methods from the domain of learning analytics have been harnessed to gain insights into such challenges, where the focus is often on the assessment of students' programs, using various approaches, such as static analysis [27], dynamic analysis [28], and natural language processing [29, 30].

One prominent approach is to study students' progress through code assignments using visual analyses. For example, the graph presented in [31] reflects the effectiveness of feedback by visualizing how submissions change over time in response to feedback. Other studies assess future performance by detecting visual patterns of behavior in code snapshots, for example by exploring size and frequency of code updates [32]. Our work uses a visual approach to provide a "bird's eye view" of common code submissions outcomes, which is easy to interpret, has the potential to reveal learning patterns at a glance, identify content that may require reevaluation and detect critical points of attrition.

When it comes to predicting performance and other learning outcomes in the context of CS education different kinds of data are used [16]. Some studies use clickstream activities information [33, 34]. While the clickstream data can be highly valuable if analyzed correctly, processing its unstructured format may be challenging for an "end-user" educator (e.g., a CS teacher). Other predictors are based on self-reported characteristics, such as gender or learning goals [35, 36]. However, the reliability of such data is not always sufficient, and it may not exist without delivering a tailor-made questionnaire.

Other studies base the prediction directly on students' submitted programs data. For example, compilation error sequences were used to predict students' frustration [37] and were later suggested as a possible proxy for traditional measures of performance [38]. Finer-grained data such as keystroke latencies were used to distinguish novice programmers from more experienced ones [39]. However, such code characteristics are often not available to teachers using existing learning environments. In this study we base our analyses on features easily extracted from code submitted by learners. The research reported in [40] has devised a programming feature profile based on early programming submissions in a course to predict performance. We adopt this approach but focus on attrition rather than on performance (our experience shows that these two are not necessarily correlated, e.g., when exercises are too straightforward or easy, or when learners do not care about their grade). The research reported in [41] indicates that perseverance in assignment submission can be used to predict persistence in a CS MOOC. Our work supports this finding and extends it by covering wider aspects in the submission history, such as performance, participation, persistence, and interaction timing.

1.3 Research Questions

In this study, we use a data-driven approach over students' code submissions to assess the pedagogical design of a fresh CS101 MOOC. Shedding light on how students interact with programming assignments in MOOCs and other tutoring systems may enable a more effective design of such educational units. More specifically, we aim to identify points in the course with a high risk of disengagement towards programming, and extract students' code submission profiles to predict attrition. As far as we know, there is no commonly used universal definition for attrition in terms of disengagement towards programming, in a way that suits the unique active hands-on nature of programming education. To that end, we pose the following research questions (RQs) in the context of a CS101 MOOC:

RQ1 What are the common patterns that characterize students' trajectories of code submission?

RQ2 Can disengagement towards programming be predicted based on an early code submission profile?

2 SETTING

2.1 'First Steps in Computer Science and Python Programming' MOOC

The MOOC 'First Steps in Computer Science and Python Programming' (1SCS) is an online course developed by Tel-Aviv University (TAU) [42]. It is offered free and publicly since 2018 on a national MOOC edX-based platform. The course is designed to make CS and programming basics accessible for the general public in a simple, friendly and non-formal manner. It aims at familiarizing learners with the computational "culture", by offering the first glimpse into several fundamentals CS themes, as detailed in Table 1. The theoretical facets introduced go side by side with programming in Python, to provide hands-on experience and a comprehensive, balanced view of this field. The first three units teach Python programming basics, such as data types, the use of variables, conditional computation, loops, and functions. The next 5 units touch on several different areas in CS, alongside programming hands-on practice

Table 1: 1SCS course content

Unit	Main themes	Main examples
1	Bits and bytes, Python, variables and basic types	The stable marriage problem and the Gale-Shapley algorithm
2	lists, strings, conditionals, functions	Maximum of a fixed number of arguments, Python packages (random, time)
3	Loops: for and while	Palindromes, naïve primality testing
4	Searching and sorting algorithms, complexity	Binary search, Selection sort
5	Error detection and correction	Check digit,parity bit, repetition codes
6	Encryption and decryption	Caesar, substitution ciphers
7	Digital images	Image representation, noise reduction
8	Complexity and computability, graph problems	Travelling salesperson, halting problem, Turing test

in the context of each topic. This MOOC assumes no prior background in programming or CS and starts from the 'bits and bytes' (literally). However, as is usually the case, some learners are more "computationally-oriented" than others and may even have prior programming experience. This poses a major educational challenge, as some learners experience the course as easy and fun, while others may find themselves struggling even with the simplest exercise. This gap is reflected in written posts in the course forums, as well as post-course questionnaires.

The programming examples and assignments are embedded within the course using the web-based IDE Codeboard [43]. The course consists of 10 graded assignment sets: one at the end of each unit, in addition to mid-course and final assignment sets. Each assignment set contains between 1-4 assignments, yielding a total of 27 graded programming assignments. In these assignments the students are required to write a short function from scratch, to edit prewritten code, or to identify and fix embedded bugs. Upon submission, Codeboard evaluates the submitted program using predefined unit tests and the student receives a grade in the range of 0 to 100. The code submissions are logged in Codeboard, and the instructional staff can inspect all learners' submissions history.

The course runs in a self-paced mode, where all course resources are accessible from the beginning of the learning process and students have the flexibility to learn and submit assignments at their own pace. Nevertheless, a teaching assistant supports the learning process in the course forums. Learners can record their achievements (referred to as credited learners) by paying a fee and register for an on-campus exam. The majority of these are TAU students from the humanities, social sciences, and other "non-exact" sciences who obtain academic credit for their participation, as well as undergraduate candidates to several university departments.

Over the years, some learners have reported in the end-of-course open surveys, that the first three programming units require much more time and learning effort and may even lead to frustration and negative view of CS as a field. Indeed, this was a major design issue considered during the course development - to what extent and at which pace should programming be delivered, and whether the course should start with concentrated units on programming (as is eventually the case) or whether spreading it gradually over the whole course can help students absorb programming more easily. Indeed, this was one of the motivating issues for the current study.

2.1.1 Population. Between October 2018 to July 2020, 4,064 learners participated in the MOOC's graded programming assignments, 491 of those are credited learners, 368 of which are TAU students. The other 3,573 are non-credited learners. We believe that the latter audience is where the MOOC has the most notable potential to have an educational impact and foster social change. These non-credited learners come from various geographical locations, ages, and backgrounds. Most notably, the course is used by innovative CS educators from several high schools, youth, and social non-profit organizations, including some from the geographic and socio-economic periphery, who wish to incorporate the MOOC in their activities. We note that a small number of these learners eventually feel competent enough to pursue accreditation in the on-campus final exam. Each year TAU hosts a seminar for those educators, in which they are exposed to pedagogical issues related to distant and hybrid learning using the MOOC.

In terms of gender distribution, 64% of the students identify themselves as male, and 34% as female. Of those who reported their education level, 30% have secondary education or less, and 20% acquired post-secondary education. The reported age varied (roughly) from 12 to 75, with 12% of the students under 20, 32% in the range of 20-40, and 10% above (all numbers are based on self-reports).

2.2 Data

2.2.1 Using merely code submissions. It is one of this study's main goals to facilitate the adoption of CS and programming MOOCs by educational agents and institutions who wish to bridge social gaps. Therefore, we consider data that is legally, freely, and easily available to any "end-user" educator, mostly excluding data that requires substantial effort to acquire or are less reliable. our dataset for this study contains 71,157 graded code submissions of the 4,064 unique learners who participated in the course and submitted their assignments. To identify the learners, we de-anonymized the dataset [44].

2.2.2 Data preprocessing. We use a dynamic analysis approach that evaluates source-code execution based on the input-output unit-tests. We transformed each code submission into a functional vector representing the submission outcome, with the following entries:

- **Assignment Number:** A value in the range of 1 to 27.

- **Inner Grade:** A value between 0 and 100, which is the percentage of tests passed when submitting the code. The inner grade is not visible to the learner.
- **Grade:** A value between 0 and 100 with the grade that the student received for his submission. This value is not necessarily identical to the inner grade since the tests are divided into groups. Each group contains tests that examine a particular aspect in the solution correctness, e.g., straightforward cases, edge cases, handling invalid inputs, and fulfilling constraints specified in the assignment. The automatic grading system evaluates the code by groups. A group is considered successful if all its tests pass successfully, otherwise the group completes with failure. The grade field is the percentage of successful groups. This way, it equally weighs the different aspects of solution correctness and compensates for the high correlation between succeeding in tests that examine the same aspect of correctness.
- **Execution Results Mapping:** A mapping of which tests successfully passed and which failed. The map is in the form of a vector of binary indicators. For example, for an assignment with three tests and submitted code that fails only the first test, the vector will be $(0, 1, 1)$.

After extracting this functional vector out of each submission, we generated, for each student, a sequence of her progress in time through the code assignments. Since we focus on the initial learning trajectories of code submissions, and since the programming exercises generally increase in difficulty along the course timeline, we ignored submissions that go back to earlier units (for example by students who wish to improve their grade before the final exam).

3 STUDY 1

3.1 Methodology

With the aim of shedding light on how students interact with programming assignments, we built the submission trajectory graph - a graphical visualization of the students' submission sequences. The graph provides an overview of how the students progress through the course timeline of graded assignments. It allows a high-level visual identification of common learning paths, as well as paths "less travelled by", problematic assignments where many fail, attrition points, etc. Next, we define the submission trajectory graph structure.

Nodes: The nodes are sorted into layers according to the course timeline and represent the observed submission outcomes, containing a 4-tuple of assignment number, grade, inner grade, and execution results mapping (as described in section 2.2.2). The nodes in layer k represent the various observed outcomes for assignment k and are labeled 'ass{k}_{m}' (e.g., 'ass1_1', 'ass1_2', 'ass1_3' in Figure 1 represent three different outcomes observed in assignment 1). Between layers k and k+1 we add two nodes, labeled 'k' and 'End_k' (these will improve the visual representation for attrition, as will be explained below). In addition, we have a node labeled 'Start'.

Edges: The edges reflect submission patterns. For a pair of consecutive submission outcomes in a sequence of student i, if the two submissions belong to the same assignment, we create an 'in-layer' edge between the two submission outcomes (e.g., edge (ass1_1,ass1_2) in Figure 1). If the submissions belong to different assignments, we represent the transition by two 'between-layer' edges (e.g., edges (ass1_2,1) and (1,ass2_1) in Figure 1). For each student, we start the trajectory in the 'Start' node and end it in an 'End_k' node, where k is her last assignment submitted. The edges to the sink nodes (such as 'End_1' in Figure 1) capture attrition distribution over the various programming assignments.

To illustrate the graph generation, let us consider a course with only two assignments and four participants who interacted with the assignments as follows: Student 1 failed in all tests of assignment 1. Then she fixed her code, resubmitted it and all tests passed successfully. In assignment 2 all her tests passed successfully. Student 2 partially succeeded in assignment 1. Then he drops out of the course. Students 3 and 4 submitted successfully both assignments in the first trial. Denote vertices 'ass1_1', 'ass1_2', 'ass1_3' as an absolute failure, an absolute success, and partial success in assignment 1, and 'ass2_1' as an absolute success in assignment 2. Figure 1 illustrates the submission trajectory graph of the described scenario. For instance, for student 1 we generated a trajectory with the following set of edges: {(Start,ass1_1), (ass1_1,ass1_2), (ass1_2,1), (1,ass2_1), (ass2_1,2), (2,End_2)}, and student 2 trajectory consists of the following set of edges: {(Start,ass1_3), (ass1_3,1), (1,End_1)}.

The submission outcome nodes are marked with a 'traffic-light' heatmap according to the quality of this outcome, as measured by its inner grade. At one end are green nodes that represent general success, at the other are red nodes that represent general failure (e.g., the outcome node 'ass1_2' in Figure 1 is green as it represents an absolute success). Furthermore, a node's size is correlated with its in-degree. For example, the outcome node 'ass1_2' in Figure 1 is larger than 'ass1_1' since it has been observed more times. Finally, we annotated edges with heatmaps as well - darker edges indicate larger weights, as shown in Figure 1 for the example described above. To avoid visual overload in the graph we reduce the number of vertices by removing vertices with a low in-degree. Specifically, we found that the threshold of 2% of the number of students significantly improves the visualization. This means that an outcome node appears only if its correlated outcome occurred for at least 2% of the students.

3.2 Results

Figure 2 shows the submission trajectory graph of the learners in the 1SCS course. From the graph, we can observe that the majority of learners make the same consecutive transitions and receive similar outcomes when submitting assignments, as reflected in the horizontal 'zig-zag' dark grey path. The most common path passes mainly through green vertices that represent an absolute success (all inner tests pass successfully). The exceptions are assignments 10, 21 and 22, where an absolute success is common but not the most common outcome (dark green node is not the largest). Those exceptions may indicate problematic unit tests, unclear or too difficult assignments, and point at assignments that require further investigation.

In assignment 10 (4th unit, searching algorithms) the students are required to write a short function from scratch that checks whether a given list is sorted or not by comparing consecutive elements. Although the given pseudo-code and the instructions specifically

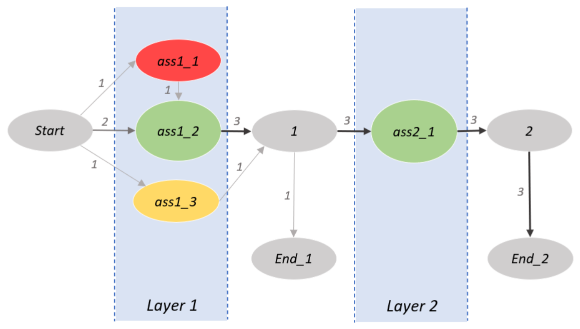


Figure 1: An illustration of the submission trajectory graph. The nodes are sorted into layers according to the assignments' timeline (there are 27 layers for our 1SCS MOOC). In-layer nodes represent different submission outcomes, their size and color are correlated to their frequency and quality, respectively. A darker edge indicates a more frequent transition.

mentioned and emphasized the edge case where the order is broken only by the last element in the list, the students tend to fail on this scenario test ('ass10_2' large light green node). Assignments 21 and 22 belong to the 7th unit on digital images, which is considered more challenging in terms of programming level relative to the preceding units. In those assignments, the students are required to perform basic image processing operations on a given input image. In both assignments, the common outcomes are an absolute failure (all inner tests failed), represented by a red node. The course staff confirms that in these two assignments the input-output examples provided to the students are indeed very partial.

Table 2 summarizes the attrition rate distribution over the assignment timeline. There are various dropout points in the course timeline, but some are more significant: Almost 33% of the students quitted solving programming assignments after assignment set 2, about 9% left in assignment set 3, and only 35.21% of them participate in the final assignment set. Overall, the attrition points are highly associated with specific stages in the course timeline: in assignment sets resolution, the first two assignment sets with code exercises (assignment sets 2-3) are the main attrition points. While in in-set resolution, attrition occurs mainly at the end of an assignment set (e.g., 19.38% dropout after completing the last assignment of assignment set 2).

4 STUDY 2

From the very beginning, MOOCs have suffered from a low percentage of completion, when compared to "standard" frontal academic courses [6]. Although the research community has been studying this phenomenon for years, it is still a challenging contemporary problem [45]. Student engagement is considered one of the major factors that determine the attrition rate from MOOCs [46]. There are numerous causes for lack of engagement: some of them are student-related, such as the lack of motivation, time, or insufficient prior skills. Others are related to the course design, such as too long educational units, lack of support, etc. In the context of our course, the findings of study 1 indicate that assignment sets 2-3 are the main points where learners have a high risk to stop participating in the programming assignments (referred to as "risk points"). In this study we check whether there are early signs in the students' programming submissions of assignment sets 2 and 3 that enable the prediction of attrition. Such early prediction can be used to design mechanisms for intervention, with the goal of improving students' perseverance.

4.1 Methodology

4.1.1 Attrition definition. To the best of our knowledge, there is no universally accepted definition for MOOC's attrition in the literature. Different studies describe attrition (also referred to as dropout or stopout) as the lack of interaction with the platform, in terms of viewing videos, submitting exercises or exams, posting in a forum, earning a certificate, etc. In our context, since learning to program comes with hands-on experience, we decided to focus on characteristics surrounding programming assignments. Therefore, we focus on the submission of graded exercises as an indication of perseverance. Specifically, we consider attrition as the submission of less than 70% of the remaining programming assignments. For instance, if our checkpoint is the end of assignment set 3, then the remaining assignments are from assignment set 4 to the final assignment set (including). Because our definition does not address students' interaction with other learning resources except the graded programming assignments, we prefer to use the term attrition instead of dropout/stopout, as students may skip any programming-related content, while still participating in other parts of the course.

4.1.2 Features and label extraction. Based on the selected checkpoint in the course timeline, we divided the students' submission sequences into two: The *historical interval* from which we extract the input features, and the *prediction interval* from which we extract the label of the binary classification problem (1 for attrition and -1 for perseverance), by checking the number of distinct submitted assignments and declaring attrition when this number is less than 70% of the remaining assignments.

We converted the historical interval into a code submission profile that represents various aspects of behavior on that interval. The features cover five major aspects: performance (e.g., unit test results); participation (e.g., number of assignments submitted); persistence (e.g., repeated submission of an assignment); interaction timing (e.g., the average time between submissions attempts); and population (TAU enrolled students). In total, for each student, we extracted 248 features representing 20 different attributes that are summarized in Table 3. Some attributes are associated with the entire submission sequence of a student, while others are assignment-related and thereby were extracted separately for each assignment. For some attributes, where relative measures may be more indicative than absolute ones, we also calculated the attribute with respect to the average and percentile values, to assess the learner in comparison to her peers. Finally, we applied L_2 normalization to all features.

4.1.3 Method. We consider three standard and simple to interpret supervised learning models for our binary classification problem: soft support vector machine (Soft-SVM) [47] , logistic regression

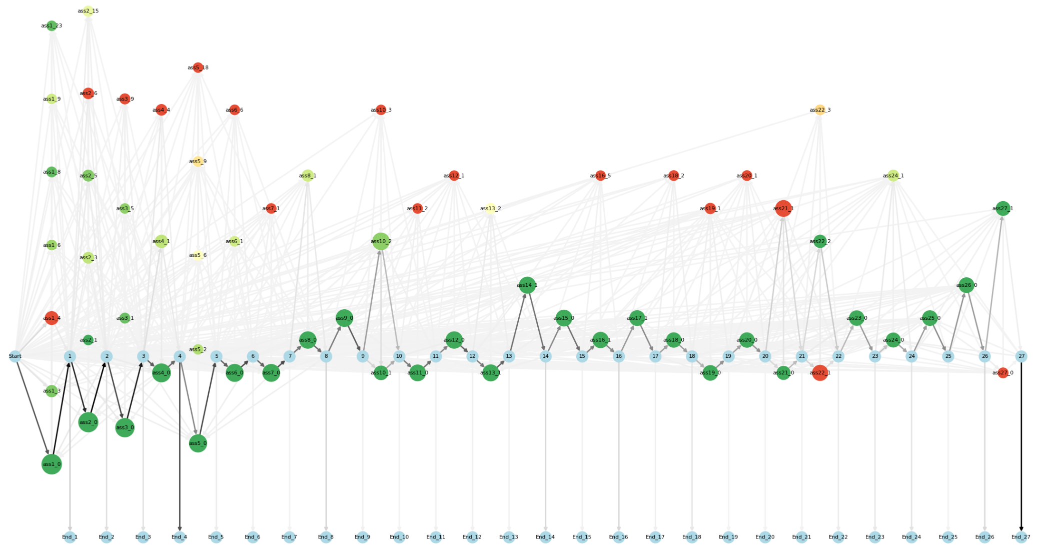


Figure 2: The submission trajectory graph in the 1SCS MOOC. A darker path reflects more common trajectories: the horizontal 'zig-zag' path visits mostly green nodes, indicating that the majority tend to succeed. The edges to the sink nodes at the bottom show attrition distribution over assignments: a major dropout point is assignment 4 (end of assignment set of assignment set 2).

[48], and decision tree [49, 50]. We split the dataset randomly into two fixed sets, 80% training set and 20% testing set. To improve models' performance, we fine-tuned all models' parameters during training to maximize accuracy via 5-cross-validation folders. To avoid uncertainty when concluding the features' predictive power, we validate features consistency by running the models multiple times with different seeds. Since our features' dimensional space is relatively small, we did not perform any feature selection.

We used the models' implementation offered by the sklearn open-source package [51]. To measure the effectiveness of the tuned models for the task of attrition prediction, the following metrics are reported: accuracy, precision, recall, and F1-score as defined in Table 4. We also report the precision, recall, and F1-score for the negative category (perseverance) as well. Moreover, we compared the models using the ROC (Receiver Operating Characteristic) curve.

4.2 Results

We selected the attrition risk point detected in study 1 (end of assignment set 3) as the checkpoint for our prediction model, meaning we predict attrition with the historical interval of assignments 1-8. With respect to this checkpoint, 2763 learners received the 'attrition' label and the rest 1301 received the 'perseverance' label. Table 5 and Figure 3 show the evaluation and the ROC curve of the three models. All models converge into a similar Area Under Curve (AUC) of 0.83%-0.84% (Figure 3). The SVM and the Logistic Regression achieved at least 90% precision on positive and recall of 74% on positive, while the Decision Tree reached lower positive precision (86%) and higher positive recall (77%). Regarding the measurements

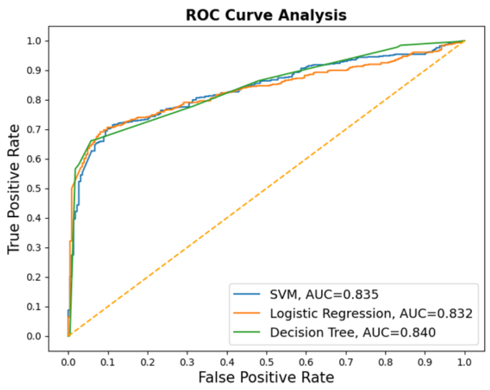


Figure 3: The models' ROC curve for the task of attrition prediction with two assignment sets history. The ROC curve shows the true-positive rate against the false-positive rate at various classification thresholds. The area under curve (AUC) provides an aggregate measure of the performance across all possible thresholds.

of the negative class, all models achieved around 53% Precision and recall of 68%-81%.

Figure 4 (A) shows the five features that had the most predictive power for each model. Figure 4 (B) shows the Pearson Correlation Coefficient (linear correlation) between those features. From Figure 4 (A) we learn that some of the top features affect prediction under

Table 2: Students' attrition distribution over programming assignments. The attrition rates are visualized by the top-down edges to sink nodes in Figure 2.

Assignment	Assignment set	Percentage of dropouts by assignment	Percentage of dropouts by unit
1	2	5.97%	32.75%
2		3.71%	
3		3.66%	
4		19.38%	
5	3	1.62%	8.66%
6		0.95%	
7		0.68%	
8		5.38%	
9	4	0.95%	2.55%
10		0.61%	
11		0.98%	
12	mid-course	0.86%	6.25%
13		0.54%	
14		4.84%	
15	5	0.86%	5.95%
16		5.09%	
17	6	0.95%	3.27%
18		2.31%	
19	7	0.46%	3.00%
20		0.24%	
21		0.56%	
22		1.72%	
23	8	2.33%	2.33%
24	Final	3.10%	35.21%
25		0.63%	
26		4.87%	
27		26.59%	

more than one model (e.g., Percentile Q - the percentile of the number of distinct assignments submitted is predictive to some extent in all 3 models). In other cases, features are uniquely associated only with a single model. For example, only the decision tree uses the number of submissions for assignment 8 (feature H_8), while only the SVM uses the maximal grade of assignment 7 (feature E_7). However, these two features are correlated, as shown in Figure 4 (B). It is interesting to note that the binary indicator of whether the student is TAU enrolled (feature T), is used by only 2 out of 3 models, which we find surprising. This feature represents an external incentive for engagement and was found to be weakly correlated with the rest of the features (Figure 4 (B)), therefore, we expected all 3 models to relate to it. This result may be explained by the relatively low size of TAU's students in the general course population.

5 DISCUSSION

In this paper we study students' interaction with auto-graded programming assignments in a CS101 MOOC that includes programming (in Python). We use a data-driven approach that involves a graphical visualization of students' submission trajectory through the course, and a prediction of disengagement towards programming using classic machine learning methods.

We generated a submission trajectory graph to track common patterns of behavior. This provides a "bird's eye view" of common code submissions outcomes, learning progress trajectories, and attrition distribution over the programming assignments in the course timeline. Our study conforms with previous reports on early high dropout rates in MOOCs. These valuable observations are easily extracted almost at a glance from such type of graph, providing insights into possible interventions and content that may require re-design. We believe that incorporating simple visual tools in online learning platforms is a substantial step towards data mining facilitation by educators using online learning environments.

Specifically, we concluded that the vast majority of students in our MOOC follow the same submission trajectory and tend to fully succeed in almost all the graded assignments they submit. This is surprising, given the variance in population types and motivation, and increases the suspicion that the unit tests are too simple and not inclusive enough, and may benefit from a further in-depth inspection. We therefore conclude that the overall success in assignments in our MOOC does not necessarily indicate effective learning. Moreover, we suspect that the high dropout rates in assignment sets 2 and 3 constitute the first evidence that programming fundamentals presented in this MOOC should be designed in a more approachable way, that reduces cognitive overload. To that end, we suggest that units 2 and 3, which introduce programming fundamentals

Table 3: The list of extracted features for assignments 1-8. Features with * marking were also calculated with respect to their average and percentile values

	Feature Description
A_k	A binary indicator of whether the student submitted assignment k
B_k	Grade of first submission of assignment k
C_k	Inner grade of first submission of assignment k
$D_{k,m}$	Binary indicator of whether the student successfully passed the internal test m of assignment k
E_k	*Maximal grade for assignment k
F_k	*Average grade for assignment k
G_k	*Maximal inner grade for assignment k
H_k	*Number of submissions for assignment k
I_k	*Average time between submissions of assignment k
J_k	*Time between first and last attempts of assignment k (0 if submitted only once)
K_k	Number of submissions of assignment k with grade >= 60
L_k	Percentage of submissions of assignment k with grade >= 60
M_k	Binary indicator of whether the student received grade >= 60 for any submission of assignment k
N	*Total number of submissions
O_q	*Average time between re-submissions of the same assignment set q
P	*Average H_k over all k
Q	*Number of distinct assignments submitted
R	*Number of distinct assignments submitted with grade >= 60
S	*Maximal H_k over all k
T	Binary indicator of whether the student is TAU enrolled

Table 4: Evaluation metrics, where TP = True Positive; FP = False Positive; TN = True Negative; FN = False Negative

Metric	Accuracy	Precision	Recall	F1-Score
Definition	$\frac{TP+TN}{TP+FP+TN+FN}$	$\frac{TP}{TP+FP}$	$\frac{TP}{TP+FN}$	$\frac{2*Percision*Recall}{Percision+Recall}$

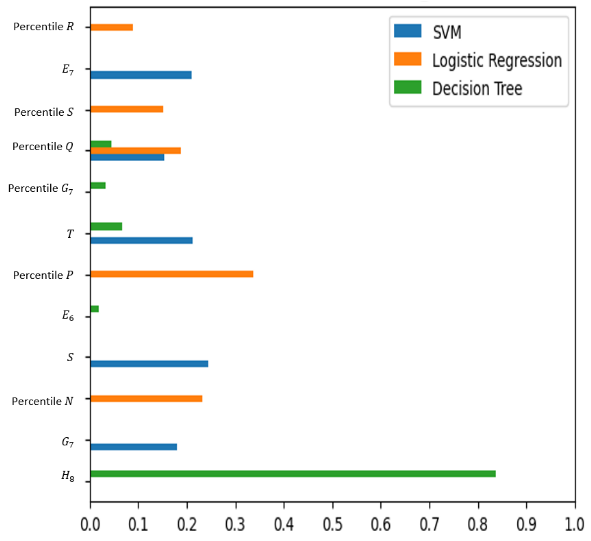
Table 5: Evaluation of 3 models for the task of attrition prediction with two assignment sets history. We report the precision, recall and F1-score for both attrition (+) and perseverance (-)

Model	Accuracy	Precision (+)	Recall (+)	F1 Score (+)	Precision (-)	Recall (-)	F1 Score (-)
SVM	0.75	0.90	0.74	0.81	0.53	0.78	0.63
Logistic Regression	0.76	0.91	0.74	0.81	0.54	0.81	0.65
Decision Tree	0.75	0.86	0.77	0.82	0.53	0.68	0.60

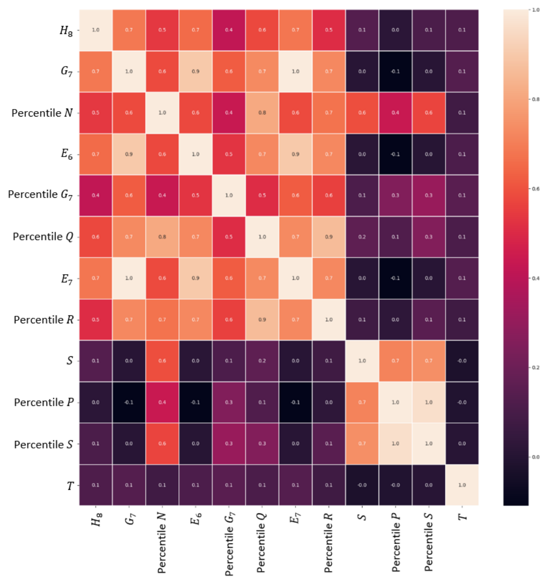
such as conditionals, functions, and loops, should be split into 3 or even 4 shorter units. This conclusion is consistent with learners' end-of-course open surveys, in which these units are reported to require much more time and learning effort, and often lead to frustration and disengagement. Such a split is mostly technical and easy to implement in the edX-based platform and does not involve any content modifications. Furthermore, for the teachers who use this MOOC in hybrid learning we suggest spending more time on these units in class, and where possible provide more assistance and guidance to their learners at these stages.

Our work further presented a tailor-made definition for attrition, that suits the active hands-on nature of programming education. Under our definition for attrition, the prediction problem can be set up independently for any strategic checkpoint. We have shown that an interval of submission data from two early assignment sets can be composed into a profile that enables classic machine learning models to predict attrition in a way that captures most students who are likely to suffer from attrition (precision of 90%), without scarifying the prediction of those who are likely to persist. Furthermore, the prediction relative success indicates that some of our proposed features indeed capture students' disengagement toward programming. Yet, to determine the reasons for attrition, further research is needed.

One limitation of our approach is that it is restricted to data related only to graded programming submissions recorded by the IDE used. However, one should remember that the clickstream data are not always available for the course staff using a MOOC, as appeared to be in our case. But even if such data were available, cleaning, formatting, and processing may become tedious and require considerable computational skills, whereas programming submissions are



A



B

Figure 4: A) Top 5 predictive features of the 3 models. The features description appears in Table 3. To simplify comparison, the scores were normalized to [0,1] regardless of the sign of the correlation (negative/positive). B) The correlation between to top 5 predictive features.

much more accessible and intuitive to process by any programming instructor. Still, we hope to enrich our findings in the future using clickstream data.

6 CONCLUSIONS

The current study analyzed consistency and persistence in programming assignments in a CS101 MOOC, using methods of visual data mining, extracting learners' code submission engagement and performance profiles, and predicting disengagement towards programming.

We believe this study has two main contributions: (1) The visual methodology presented in this paper may facilitate a birds-eye view investigation for large scale learning environments, which is easy to interpret and provide the instructor with valuable insights on learning behavior. For our own CS101 MOOC, the research supports either changing the course structure or designing a focused intervention in the strategic checkpoint. We believe that our proposed technique can be generalized to visualize students' learning patterns and struggles in other CS courses as well; (2) It presents a simple data-driven approach for the investigation of interaction with programming assignments, which can be generalized to other hand-on types of interactions in MOOCs. While our methods lean on standard tools, some aspects of the analysis are designed to capture the unique nature of hands-on programming experience.

ACKNOWLEDGMENTS

This study was supported by Len Blavatnik and the Blavatnik Family foundation.

REFERENCES

[1] https://www.classcentral.com/report/the-second-year-of-the-mooc [link extracted September 2021]

[2] Ng, Andrew, and Jennifer Widom. "Origins of the modern MOOC (xMOOC)." Hrsg. Fiona M. Hollands, Devayani Tirthali: MOOCs: Expectations and Reality: Full Report (2014): 34-47.

[3] Rodriguez, Osvaldo. "The concept of openness behind c and x-MOOCs (Massive Open Online Courses)." Open Praxis 5.1 (2013): 67-73.

[4] Littlejohn, Allison, *et al.* "Learning in MOOCs: Motivations and self-regulated learning in MOOCs." The Internet and Higher Education 29 (2016): 40-48.

[5] Wong, Jacqueline, *et al.* "Supporting self-regulated learning in online learning environments and MOOCs: A systematic review." International Journal of Human–Computer Interaction 35.4-5 (2019): 356-373.

[6] Khalil, Hanan, and Martin Ebner. "MOOCs completion rates and possible methods to improve retention-A literature review." EdMedia+ innovate learning. Association for the Advancement of Computing in Education (AACE), 2014.

[7] Onah, Daniel FO, Jane Sinclair, and Russell Boyatt. "Dropout rates of massive open online courses: behavioural patterns." EDULEARN14 proceedings 1 (2014): 5825-5834.

[8] Topali, Paraskevi, *et al.* "Exploring the problems experienced by learners in a MOOC implementing active learning pedagogies." European MOOCs Stakeholders Summit. Springer, Cham, 2019.

[9] Zheng, Saijing, *et al.* "Understanding student motivation, behaviors and perceptions in MOOCs." Proceedings of the 18th ACM conference on computer supported cooperative work & social computing. 2015.

[10] Alexandron, Giora, *et al.* "Are MOOC Learning Analytics Results Trustworthy? With Fake Learners, They Might Not Be!." International Journal of Artificial Intelligence in Education 29.4 (2019): 484-506.

[11] Alexandron, Giora, *et al.* "Copying@ Scale: Using harvesting accounts for collecting correct answers in a MOOC." Computers & Education 108 (2017): 96-114.

[12] Romero, Cristobal, and Sebastian Ventura. "Educational data mining and learning analytics: An updated survey." Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 10.3 (2020): e1355.

[13] Fischer, Christian, *et al.* "Mining big data in education: Affordances and challenges." Review of Research in Education 44.1 (2020): 130-160.

[14] Taylor, Colin, Kalyan Veeramachaneni, and Una-May O'Reilly. "Likely to stop? predicting stopout in massive open online courses." arXiv preprint arXiv:1408.3382 (2014).

[15] Van Goidsenhoven, Steven, *et al.* "Predicting student success in a blended learning environment." Proceedings of the Tenth International Conference on Learning Analytics & Knowledge. 2020.

[16] Gardner, Josh, and Christopher Brooks. "Student success prediction in MOOCs." User Modeling and User-Adapted Interaction 28.2 (2018): 127-203.
[17] Saint, John, *et al.* "Using process mining to analyse self-regulated learning: a systematic analysis of four algorithms." LAK21: 11th International Learning Analytics and Knowledge Conference. 2021.
[18] Bell, Tim, and Jan Vahrenhold. "CS unplugged—how is it used, and does it work?." Adventures between lower bounds and higher altitudes. Springer, Cham, 2018. 497-521.
[19] Armoni, Michal. "Teaching CS in kindergarten: How early can the pipeline begin?." Acm Inroads 3.4 (2012): 18-19.
[20] The Joint Task Force on Computing Curricula, C. O. R. P. O. R. A. T. E., ed. "Computing curricula 2001." Journal on Educational Resources in Computing (JERIC) 1.3es (2001): 1-es.
[21] Cooper, Stephen, Wanda Dann, and Randy Pausch. "Teaching objects-first in introductory computer science." Acm Sigcse Bulletin 35.1 (2003): 191-195.
[22] Chor, Benny, and Rani Hod. "CS1001. py: a topic-based introduction to computer science." Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education. 2012.
[23] Sands, Phil, and Aman Yadav. "Self-Regulation for High School Learners in a MOOC Computer Science Course." Proceedings of the 51st ACM Technical Symposium on Computer Science Education. 2020.
[24] Marin, Victor J., *et al.* "Automated personalized feedback in introductory Java programming MOOCs." 2017 IEEE 33rd International Conference on Data Engineering (ICDE). IEEE, 2017.
[25] Liu, Xiao, *et al.* "Automatic grading of programming assignments: an approach based on formal semantics." 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET). IEEE, 2019.
[26] Daradoumis, Thanasis, *et al.* "A review on massive e-learning (MOOC) design, delivery and assessment." 2013 eighth international conference on P2P, parallel, grid, cloud and internet computing. IEEE, 2013.
[27] Nguyen, Huy, *et al.* "Exploring Metrics for the Analysis of Code Submissions in an Introductory Data Science Course." LAK21: 11th International Learning Analytics and Knowledge Conference. 2021.
[28] Cornelissen, Bas, *et al.* "A systematic survey of program comprehension through dynamic analysis." IEEE Transactions on Software Engineering 35.5 (2009): 684-702.
[29] Azcona, David, *et al.* "user2code2vec: Embeddings for profiling students based on distributional representations of source code." Proceedings of the 9th International Conference on Learning Analytics & Knowledge. 2019.
[30] Shi, Yang, *et al.* "Toward Semi-Automatic Misconception Discovery Using Code Embeddings." LAK21: 11th International Learning Analytics and Knowledge Conference. 2021.
[31] McBroom, Jessica, *et al.* "A data-driven method for helping teachers improve feedback in computer programming automated tutors." International Conference on Artificial Intelligence in Education. Springer, Cham, 2018
[32] Blikstein, Paulo, *et al.* "Programming pluralism: Using learning analytics to detect patterns in the learning of computer programming." Journal of the Learning Sciences 23.4 (2014): 561-599
[33] Porter, Leo, Daniel Zingaro, and Raymond Lister. "Predicting student success using fine grain clicker data." Proceedings of the tenth annual conference on International computing education research. 2014.
[34] Shi, Yuling, Zhiyong Peng, and Hongning Wang. "Modeling student learning styles in MOOCs." Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. 2017.
[35] Crues, R. Wes, *et al.* "How do gender, learning goals, and forum participation predict persistence in a computer science MOOC?." ACM Transactions on Computing Education (TOCE) 18.4 (2018): 1-14.
[36] Polso, Kukka-Maaria, *et al.* "Achievement goal orientation profiles and performance in a programming MOOC." Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education. 2020.
[37] Rodrigo, Ma Mercedes T., and Ryan SJd Baker. "Coarse-grained detection of student frustration in an introductory programming course." Proceedings of the fifth international workshop on Computing education research workshop. 2009.
[38] Jadud, Matthew C., and Brian Dorn. "Aggregate compilation behavior: Findings and implications from 27,698 users." Proceedings of the eleventh annual international conference on international computing education research. 2015.
[39] Leinonen, Juho, *et al.* "Automatic inference of programming performance and experience from typing patterns." Proceedings of the 47th ACM Technical Symposium on Computing Science Education. 2016.
[40] Pereira, Filipe Dwan, *et al.* "Early performance prediction for CS1 course students using a combination of machine learning and an evolutionary algorithm." 2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT). Vol. 2161. IEEE," 2019.
[41] Chen, Chen, *et al.* "Computational thinking and assignment resubmission predict persistence in a computer science MOOC." Journal of Computer Assisted Learning 36.5 (2020): 581-594.
[42] https://courses.campus.gov.il/courses/course-v1:TAU+ACD_TAU_cs101x+2020_1/info [link extracted September 2021]
[43] https://codeboard.io [link extracted September 2021]
[44] https://edx.readthedocs.io/projects/edx-partner-course-staff/en/latest/student_progress/course_student.html#accessing-anonymized-learner-ids [link extracted September 2021]
[45] Dalipi, Fisnik, Ali Shariq Imran, and Zenun Kastrati. "MOOC dropout prediction using machine learning techniques: Review and research challenges." 2018 IEEE Global Engineering Education Conference (EDUCON). IEEE, 2018.
[46] Xiong, Yao, *et al.* "Examining the relations among student motivation, engagement, and retention in a MOOC: A structural equation modeling approach." Global Education Review 2.3 (2015): 23-33.
[47] Wang, Lipo, ed. Support vector machines: theory and applications. Vol. 177. Springer Science & Business Media, 2005.
[48] Peng, Chao-Ying Joanne, Kuk Lida Lee, and Gary M. Ingersoll. "An introduction to logistic regression analysis and reporting." The journal of educational research 96.1 (2002): 3-14.
[49] Quinlan, J. Ross. "Induction of decision trees." Machine learning 1.1 (1986): 81-106.
[50] Breiman, Leo, *et al.* Classification and regression trees. CRC press, 1984.
[51] https://scikit-learn.org/stable [link extracted September 2021]