User Modeling and User-Adapted Interaction (2019) 29:759–788
https://doi.org/10.1007/s11257-019-09234-7

Detecting students-at-risk in computer programming classes with learning analytics from students' digital footprints

David Azcona[1] · I-Han Hsiao[2] · Alan F. Smeaton[1]

Received: 19 April 2018 / Accepted in revised form: 10 April 2019 / Published online: 25 April 2019

Abstract
Different sources of data about students, ranging from static demographics to dynamic behavior logs, can be harnessed from a variety sources at Higher Education Institutions. Combining these assembles a rich digital footprint for students, which can enable institutions to better understand student behaviour and to better prepare for guiding students towards reaching their academic potential. This paper presents a new research methodology to automatically detect students "at-risk" of failing an assignment in computer programming modules (courses) and to simultaneously support adaptive feedback. By leveraging historical student data, we built predictive models using students' offline (static) information including student characteristics and demographics, and online (dynamic) resources using programming and behaviour activity logs. Predictions are generated weekly during semester. Overall, the predictive and personalised feedback helped to reduce the gap between the lower and higher-performing students. Furthermore, students praised the prediction and the personalised feedback, conveying strong recommendations for future students to use the system. We also found that students who followed their personalised guidance and recommendations performed better in examinations.

Keywords Computer Science Education · Learning analytics · Predictive modelling · Machine learning · Peer learning · Educational data mining

✉ David Azcona
David.Azcona@insight-centre.org

I-Han Hsiao
Sharon.Hsiao@asu.edu

Alan F. Smeaton
Alan.Smeaton@dcu.ie

[1] Insight Centre for Data Analytics, Dublin City University, Glasnevin, Dublin, Ireland

[2] School of Computing, Informatics and Decision Systems Engineering, Arizona State University, Tempe, AZ 85281, USA

1 Introduction

In a recent literature review on learning computer programming, 10 years of survey results highlight that today's CS (Computer Sciences) classes still miss out on the use of diverse forms of Learning Analytics (Ihantola et al. 2015) to improve student performance in some way. Automated collection of data on computer programming activities is typically used in isolation within designated programming learning environments such as WebCAT (Edwards and Perez-Quinones 2008). Combining with other complementary data sources (i.e., assignment performance or demographics or prior learning) may require retrieval and aggregation from different course management systems. As a result, most of the data collection in the reported studies on CS learning is extremely customized and impossible to replicate and reproduce. Today, the majority of computer programming classes are delivered via a blended instructional strategy with face-to-face instruction in classrooms supported by online tools such as intelligent tutors, self-assessment quizzes, online assignment submission, and course management systems. New attempts in today's classrooms seek to combine multiple modalities of data such as gestures, gaze, speech or writing from video cameras or lecture recordings to leverage students' digital footprints (Blikstein and Worsley 2016; Ochoa 2017).

In this work, we propose to combine general data sources at a Higher Education Institution and to build predictive models that are able to identify students in need of assistance. Consequently, due to the nature of the data sources that we use are general and well established, this approach should be reproducible for other practitioners in a straightforward way. We use student characteristics, prior academic history, students' programming laboratory work, and all logged interactions between students' offline and online resources. We generate predictions of end-of-course outcome on a weekly basis, during the semester. In addition, during the second half of the semester, students can opt-in to receive pseudo real-time personalised feedback on their progress. The feedback includes notification of their predicted performance on the module outcome and suggestions from one of the top-performing students from the same class, a form of peer feedback. We conducted a semester-long classroom study and collected data from one of the programming courses that adopted our predictive system. Furthermore, lecturers on the course were updated each week regarding students' progress and those students who opted-in to receive customised notifications were surveyed for their views and impressions.

This work contributes to a new implementation of a predictive analytics system that aggregates multiple sources of students' digital footprints from blended classroom settings. This is multimodal data in the sense that it is derived from multiple sources of information about students and in the remainder of the paper we refer to it as the students' *digital footprints*. Advanced Data Mining techniques are adopted to engineer models to provide realtime prediction and dynamic feedback. This approach incorporates static and dynamic student data features to enhance predictive model scalability that can be extrapolated to other blended classrooms and to other subjects. Additionally, not only is the approach we take generic, it also permits applicability to limited data sets (e.g., behavior logs for laboratory material only) in order to be beneficial in helping students in need. Most importantly, the generated predictions allow us to generate adaptive feedback to each student according to each student's progression and provide guidance when in need.

The research questions can be stated as the following:

RQ1: How accurate are the proposed predictive models with generic static and dynamic student data features, in identifying those students in need in computer programming courses?
RQ2: What are the effects of timely automatic adaptive support and peer-programming feedback on students' performances?
RQ3: What are the students' and teachers' perspectives and experiences after adopting the predictive modelling and adaptive feedback system into their own classes?

2 Literature review

2.1 Learning analytics from multiple data sources

There are several approaches that can be adopted in using Learning Analytics based on data from multiple sources, to assist in gathering comprehensive information about a class of students. Examples include the collaborative confluence working space (Martinez-Maldonado et al. 2013), streamlining digitalizing physical artifacts (Hsiao et al. 2016, 2017b) or mobile devices (Prieto et al. 2017; VanLehn et al. 2016). Group Scribbles is one of the early innovations (Looi et al. 2008; Chen and Looi 2013; Lin et al. 2014) where students edit small cards on a private workspace then drag them into a group workspace, and where cards can be moved but not edited. The teachers' dashboard lets them view any group's workspace as well as control the class activities in various ways. MTClassroom is another system which supports small groups working around table-top computers (Martinez-Maldonado et al. 2013, 2015). The Formative Assessment with Computational Technology (FACT) system utilizes mobile devices to collect both physical and virtual data (VanLehn et al. 2016). Another example is, EduAnalysis (Hsiao and Lin 2017) which provides interfaces to facilitate paper-based artifact preparation for traditional blended classrooms, and ultimately to integrate data analytics. Moreover, the direction of affective computing research has attempted to detect affects from diverse sources and incorporated them in the learning environment. For instance, learner's joy and distress are two affective states that can be reliably tracked from a probabilistic method based on the system interactions in an educational game (Conati 2002). The Affective Learning Companion recognizes a learner's affect by monitoring facial features, posture patterns, and onscreen keyboard or mouse behaviors (Burleson 2006).

What all these methods have in common is that they combine and present student data drawn from diverse sources in addition to using the more traditional click-stream logs derived from use of the tailored educational systems. The data sources essentially encapsulate a wide range of learning activities that encode different aspects of the entire learning processes besides the interaction with a single educational system. For example, a student can learn from a face-to-face lecture, video/audio tutorial or recordings, various searching and reading activities, discussions, drawing and sketching. These activities can be encoded through body language (posture), facial expression (emotion), speech (linguistics), or writing (text). Essentially, this wide variety of example methods show that interactivity and multimodality can contribute to educational outcomes, both jointly or individually (Ritterfeld et al. 2009). The work by Ochoa (2017) summarizes the state of the art of Multimodal Learning Analytics research, which involves the most commonly captured and used data formats, including textual, audio, spatial, visual and linguistic.

There have been research efforts investigating larger scale of data collection and modelling (Tempelaar et al. 2017; Conijn et al. 2017), but most of the work still relies on the actual Learning Management System (LMS) or application-specific systems to harness the data. We are beginning to see more approaches that start integrating multiple sources of information and work with more formats of data analytics to support today's classrooms (Hsiao et al. 2017b). In the work in this paper, we propose a generic predictive modelling approach and aim to upgrade various forms of classes with advanced Learning Analytics.

2.2 Educational data mining in learning computer programming

Modelling a student's learning of computer programming is not a new topic. Student models reside in intelligent tutors or any similar adaptive educational systems. Students' learning is typically estimated based on behaviour logs, such as the interactions with tutors resulting in updates to the knowledge components. In modelling the learning of computer programming languages, there are several parameters used to estimate students' programming knowledge. For instance, learning can be gauged based on the computer programming problem solving success (Guerra et al. 2014; Sosnovsky and Peter 2015), programming assignments progression (Piech et al. 2012), dialogic strategies (Boyer et al. 2011), programming information seeking strategies (Lu and Hsiao 2017), help-seeking behaviour (Price et al. 2017), the use of hints (Rivers and Koedinger 2017; Price et al. 2017), assignment submission compilation behaviour (Altadmri and Brown 2015; Jadud and Dorn 2015), troubleshooting and testing behaviours (Buffardi and Edwards 2013), code snapshot process state (Carter et al. 2015), students' reviewing and reflecting behaviours on formal assessments across physical and digital space (Hsiao et al. 2017a), and generic Error Quotient measures (Carter et al. 2015). In the field of affect computing in learning of programming, we also see several attempts at successfully predicting students' achievements such as using keyboard dynamics and/or mouse behaviours to detect negative affective states (including boredom, confusion, and frustration) among novice C++ programming learning (Vea and Rodrigo 2016). Leveraging such data could not only provide students with feedback on their computer programming problems at hand, but also allow instructors to be informed about students' progression through a course, their programming learning paths, code quality, programming states over time, and ultimately which students are falling behind the rest of the class (Piech et al. 2012; Diana et al. 2017).

However, due to the fact that student data is not open and shared across or even within institutions, integrating different data sets, parameter selection and tuning results in different approaches and outcomes (Brooks and Thompson 2017). In addition, most of the published work has focused on generating models retrospectively. Only a few of those models are put into practise and use streamed student data to produce real-time predictions and interventions (Arnold and Pistilli 2012; Corrigan et al. 2015). To overcome the challenges and complexities of data integration in this area, we investigate a generic predictive modelling approach which combines various data sources and we evaluate it in blended instruction computer programming learning modules.

2.3 The importance of feedback in programming learning

Automated assessment is one of the most popular methods in scaled generation of student feedback. It also guarantees a fast turnaround time to deliver feedback to those students. Such techniques have already been widely used in many educational fields, such as computer programming, mathematics and physics. In programming, plug-ins built within an Integrated Development Environment (IDE) can provide students with direct feedback after they compile their codes. Other exemplar systems are WEB-CAT (Edwards and Perez-Quinones 2008), ASSYST (Jackson and Usher 1997) among many others. The common approach is to apply pattern-matching techniques that verify students' answers by comparing them with the correct answers. Unfortunately, in learning computer programming, automatic programming evaluation emphasizes only the concrete aspects of an answer. It does not take into account the flavour of the answer (i.e., whether the student seems to have been on the right track or if their logic/reasoning was somewhat correct). As a result, programming instructors frequently examine the program quality and issue feedback personally.

There are approaches that address the issue of scaling up feedback production by utilising parameterized exercises, peer production, data-driven hint generation. QuizJET (Hsiao et al. 2010) is an example program that utilizes parameterized exercises to create a sizeable collection of questions to facilitate automatic programming evaluation; PeerGrader (Gehringer 2001) and PeerWise (Denny et al. 2008) are examples that utilize student cohorts to leverage mass production; Python Programming Tutor (Rivers and Koedinger 2017) is an Intelligent Tutor that generates personalized next-step hints as the feedback based on the vast past solutions. Overall, the field of automatic programming evaluation is less focused on grading paper-based programming problems. Therefore, there is less support for personalization in this area.

A few relevant early innovations have attempted to process paper exams and handwritten work, and involve humans in the grading process (Bloomfield and Groves 2008), such as GradeScope (Singh et al. 2017), Online Judge (Cheang et al. 2003), and PGA (Hsiao 2016; Murphy 2017). This stream of work reports a few benefits of digitizing paper exams (i.e., some default feedback can be kept on the digital pages with the predefined rubrics; a student's identity can be kept anonymous, preventing potential grader bias that may have occurred if the grader recognized a student's name).

3 The student's digital footprint

Higher Education institutions collect data about students at multiple times and store this in different locations. This includes students' backgrounds and demographics at registration, interaction with online learning environments, geolocated physical data like lecture attendance or library accesses, and some aspects of their social activities like memberships of clubs and societies. Leveraging all these sources of information can shape a picture of the students' engagement and involvement on campus. Thus, we hypothesize that using data sources which form a more holistic overview of a student's life at University will construct a better estimation of students' academic progression which can be modeled over time.

Moreover, in learning some disciplines such as computer programming, students spend a fair amount of time in laboratory sessions. Students typically interact with an online platform to develop and submit their program code for specified problems leaving a far greater digital footprint that analytics platforms have make use of in the past. These types of program submission platforms are typically used to evaluate the correctness of the students' computer programming work. Unfortunately, these automated assessment systems are not the only tools that students and instructors will use and they often have to switch among several online educational platforms. Therefore, without collecting all the diverse interaction data, it is challenging to establish reliable groundtruth data on which to train predictive models.

We identified three data sources that researchers and data scientists are often able to leverage to model student interaction, engagement and effort in computer programming or laboratory-intensive courses in order to build models which achieve good predictive performance and these are straightforward to store and leverage. The data sources are as follows:

- First, a platform that manages student registration will contain student characteristics and demographics such as gender, date of birth, citizenship and domicile; prior academic history including prior-to-university test scores, equivalent to High School GPA and SAT exams in the US; and prior academic history at the University.
- Second, a custom learning platform for the teaching of computer programming or a web application that allows students to submit programs and instructors to review them. Each programming submission typically contains the program name, code, laboratory assignment sheet that it belongs to, whether the submission is correct or incorrect according to some pre-specified test cases, and the date and time of the submission.
- Finally, the same environment or the general Learning Management System (such as Blackboard, Moodle or Canvas) in the University will gather student clickstream data that reflects each student's engagement with the course material and every instance of student access to a resource is recorded and stored. These include the resource or page requested, date and time, student identifier, and the IP address of the device used for access.

These data sources contain rich information to model how students behave. In the following section, we will show how to leverage these generic data sources to build predictive analytics models in order to identify students in need of assistance. This can be extrapolated to similar data sources in other institutions. In this study, gender was not used as we chose not to extract patterns from a value that can be biased to previous student cohorts.

4 Context

Dublin City University offers a Bachelor of Science degree in Computer Applications (CA). This prepares students for a career in computing and information technology by giving them in-depth knowledge of software engineering and the practical skills to apply this knowledge to develop the technology behind computing-based products.

The first semester teaching period (Fall) typically runs from the end of September to mid-December. The second semester (Spring) runs from the end of January or beginning of February until the end of April. Laboratory sessions and computer-based examinations are carried out only during the teaching period. Dublin City University also provides in-lab peer-mentoring for most computer programming modules as well as the Lecturer (Professor) attending the laboratory sessions. In the UK and Ireland, the word module is used interchangeably with course.

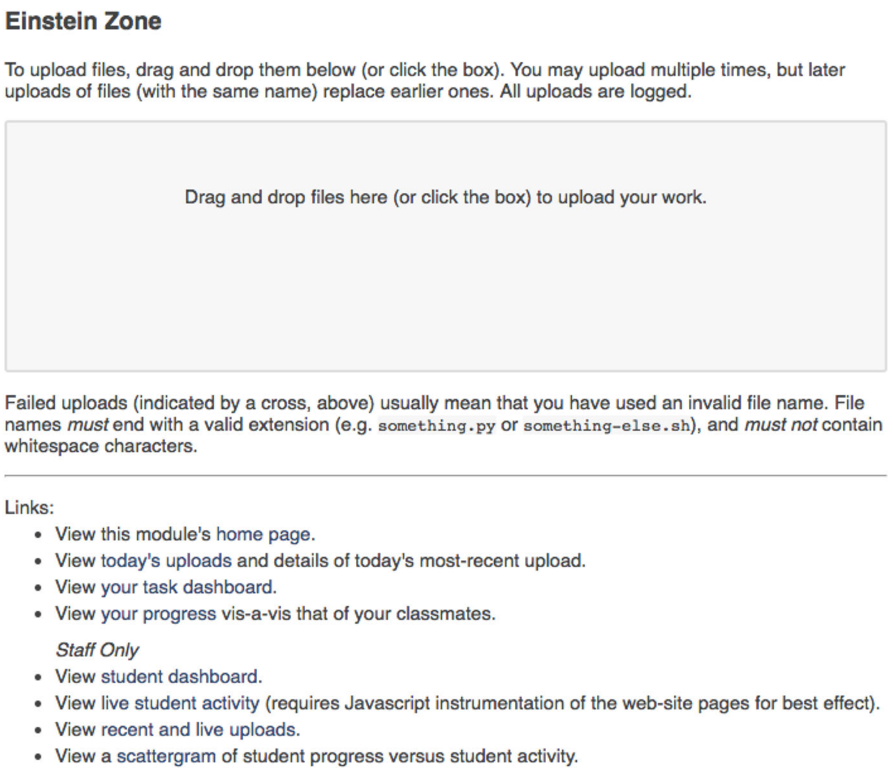
A custom Virtual Learning Environment (VLE) for the teaching of computer programming has been developed by Dr. Stephen Blott.[1] This automated grading platform is currently used in a variety of programming courses across Computer Science in Dublin City University. Students can browse course material (as on any LMS) and submit and verify their computer programming laboratory work. Figure 1 shows how students are able drag and drop their program files onto the platform. Figure 2 shows the real-time feedback students get when verifying a program by running a suite of pre-specified test cases on the grading sever.

This grading system has been used for almost four academic years as of February 2019 on a variety of computer programming courses. The log data generated by this system is leveraged using Artificial Intelligence and Machine Learning techniques and combining them with other student data sources to identify students having learning issues and adapting their learning to this discipline. Predictive models are trained using digital footprints from past student cohorts to automatically mine patterns and predict performance of current students in computer-based formal assessments.

Our previous study (Azcona and Smeaton 2017) was carried out on a Computer Programming I module (CS1) that introduces first-year Computer Applications students to computer programming and the fundamentals of computational problem solving during their first semester. We achieved this by combining two student data sources and manually identifying students for assistance during laboratory sessions. However, the Lecturer was not able to personally reach out to all students who might need assistance or be at risk during the laboratory sessions.

The research methodology used in this paper has been deployed in several courses in first and second year, building independent models per course. This paper will introduce and explain this approach in depth and showcase the analysis done on one

[1] Dr. Stephen is an Associate Professor at the School of Computing in Dublin City University http://www.computing.dcu.ie/~sblott/.



Einstein Zone

To upload files, drag and drop them below (or click the box). You may upload multiple times, but later uploads of files (with the same name) replace earlier ones. All uploads are logged.

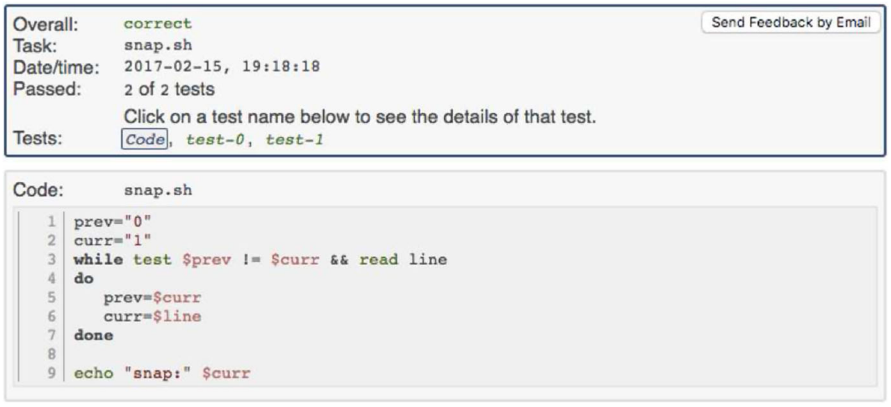
Drag and drop files here (or click the box) to upload your work.

Failed uploads (indicated by a cross, above) usually mean that you have used an invalid file name. File names *must* end with a valid extension (e.g. `something.py` or `something-else.sh`), and *must not* contain whitespace characters.

Links:
- View this module's home page.
- View today's uploads and details of today's most-recent upload.
- View your task dashboard.
- View your progress vis-a-vis that of your classmates.

Staff Only
- View student dashboard.
- View live student activity (requires Javascript instrumentation of the web-site pages for best effect).
- View recent and live uploads.
- View a scattergram of student progress versus student activity.

Fig. 1 VLE for the teaching of computer programming at Dublin City University



Overall:	correct
Task:	snap.sh
Date/time:	2017-02-15, 19:18:18
Passed:	2 of 2 tests
	Click on a test name below to see the details of that test.
Tests:	Code, test-0, test-1

Code: snap.sh

```
prev="0"
curr="1"
while test $prev != $curr && read line
do
   prev=$curr
   curr=$line
done

echo "snap:" $curr
```

Fig. 2 Instant feedback prompted to the student after submitting a program

of those courses **Computer Programming II (CS2)** as a successful example of the research methodology by combining student data from more sources and automatically and adaptively sending feedback to students about their course progression and further learning resources which are available to them and which they might use. We leverage our previous platform PredictCS (Azcona et al. 2018) to gather student data from different data sources and to carry out Artificial Intelligence and Machine Learning operations.

The CS2 module introduces first-year CA students to more advanced computer programming concepts, particularly object-oriented programming, programming libraries, data structures and file handling. Students are expected to engage extensively with hands-on computer programming in the Python programming language. In CS2 up to seven CA second-year students as well as the lecturer, give tutoring support during laboratory sessions. Students are assessed by taking two laboratory computer-based programming exams, a mid-semester and an end-of-semester assessment, during the teaching period. Students are also required to submit their laboratory work as it counts towards their final grade (10% of the overall grade for the course). This module's lectures are taught for 4 h each week for 12 weeks and the course also involves four additional hours of supervised laboratory work on two different days each week using the Python language. The course is a continuation of CS1, the introductory computer programming module. In terms of numbers, 134 students registered for CS2 during 2015/2016 academic year and 140 students in 2016/2017.

In any student intervention, it is important to capture students' opinions regarding the feedback shared with them, to understand how it affects their behaviour within the modules and whether it encourages them to try new solutions or to revise material. Thus, we gathered students' opinions about our system and the notifications via a written questionnaire at the end of the semester. The questions on the form were listed as follows:

- **Q1**: Did you opt-in? [Yes/No]
- **Q2**: If you opted-out, could you tell us why? [Comment]
- **Q3**: How useful did you find the weekly notifications? [1–5 star rating]
- **Q4**: Did you run any of the suggested working programs? [Yes/No/I was never suggested any]
- **Q5**: Would you recommend the system to a student taking this same module next year? [Yes/No]
- **Q6**: Would you like to see weekly the system notifications for other modules? [Yes/No]
- **Q7**: How could we improve the system for next year? Any other comments. [Comment]

  In addition, we enabled a discussion among the lecturers of the modules that adopted this system and formal feedback was sent to researchers via email.

5 Predictive modelling

A Predictive Analytics model can be developed to classify students and identify those who may be in need of learning assistance and are at-risk of failing their next formal laboratory assessment. The target classes are whether their performance is above or below a breakpoint between higher and lower performers, usually the pass threshold for the module. The two classes are typically imbalanced as we do not usually get equal numbers of students passing and failing these in-semester examinations.

The predictive model is trained using digital footprints from past student cohorts. It automatically extracts classification patterns and predicts performance of current students in computer-based formal assessments. Computer programming modules are quite dynamic and exercises can vary considerably from year to year, however the concepts and knowledge components being taught typically remain the same. Learning functions or classifiers can generate predictions in a timely basis, such as weekly, to detect students deviating from the desired patterns of success in each course.

5.1 Data processing and feature engineering

We trained classifier (a learning function) every week to distinguish higher versus lower-performing students based on extracting a number of features from the data. A combination of static and dynamic student features was used for each weekly function. A set of **static features** was extracted before the start of the semester for each course and each student based on their characteristics and prior academic performance. Then, each week, a set of **dynamic features** was collected for each student by building engagement and progression features based on their interactions and submissions to the platforms.

Every week, a classifier was built by concatenating the static and dynamic features from previous weeks and that week's dynamic features in order to account for each student's progression and engagement throughout the course. To clarify, every week a new set of dynamic features are appended to the pool of features extracted from previous weeks to develop a new classifier.

The data sources we leverage in order to model student interaction, engagement and effort in computer programming courses are the following:

- *Student characteristics*: student demographics.
- *Prior academic history*: prior-to-university test scores. In our previous work; Azcona and Smeaton (2017), we analysed 950 first-year Computer Science (CS) entrants across a 7 year period and showed the significant correlation between their entry points or mathematics skills and how they perform in their first year and in computer programming modules.
- *Programming submissions*: our custom platform allows students to submit their laboratory programs and provides immediate feedback for each submission based on a suite of unit tests. The information extracted for each submission is the program name, code, laboratory sheet that it belongs to, whether the submission is correct or incorrect according to the lecturer test cases, and the date and time of the submission.
- *Interaction logs*: students interact online with the course's custom VLE and every instance of student access to a page is recorded and stored. These are web logs from an Apache web server for the resource or page requested, date and time, student identifier, and the IP address of the device used for access.

The following set of **static features** are extracted before the start of the semester for each course and student:

- *Student characteristics*:
    - Age based on their date of birth and registration date.
    - Travel distance from home to university based on their domicile.
    - Route to university: Irish Leaving Certificate, Athletic scholarship, Disadvantaged background, Mature entry, etc.
- *Prior academic performance*:
    - Irish CAO points and Leaving Certificate exam scores (equivalent to High School GPA and SAT exams in the US)
    - Prior academic history at the university.
    - Final laboratory exam grades from prior courses in CS.

Then, each week, a set of **dynamic features** are extracted for each student based on raw log data, interaction events for students accessing material and corresponding computer programming submissions:

- *Programming effort*:
    - Computer programming laboratory work completed that week: percentage of correct exercises on that week's labsheets.
    - Cumulative computer laboratory work completed since the start of the semester.
- *Engagement*:
    - Lab attendance: whether the student attended the laboratory sessions or not.
    - Time spent on the online platform.
    - Ratio of during-laboratory to non-laboratory time accesses.
    - Material covered based on the resources made available each week.
    - Average time of the day the course material is accessed.
    - Average lapse time between a resource being made available and the student accessing it for the first time.
    - Ratio of on-campus to off-campus accesses based on IP address.
    - Ratio of weekday to weekend accesses.

Table 1 lists the features with an associated short name that we will use for graphs and tables in the remainder of this paper.

5.2 Training the model retrospectively

A retrospective analysis was carried out to verify the viability of the model. We developed a classification model trained with student data from 2015/2016 for CS2. A set of learning functions or binary classifiers, one per week, were built to predict a student's likelihood of passing or failing the next computer-based laboratory exam. In 2015/2016, there was a mid-semester exam in week 6 and an end-of-semester exam in week 12 for CS2. To clarify, classifiers from week 1 to 6 were trained to predict the mid-semester laboratory exam outcome (pass or fail for each student) and from 7 to 12 the end-of-semester's laboratory exam outcome. The dynamic features mentioned

Table 1 Feature names and short names

Feature name	Short name
Travel distance to university	Distance
Irish CAO points (high school GPA)	CAO points
Leaving certificate math exam score (SAT exams)	Math LC
Average grade on previous formal assignments	Avg. grade
Laboratory assignment *Course year*	Exam *Course year*
Computer programming work completed on week *x*	Results W*x*
Cumulative computer programming work completed on week *x*	Cumulative W*x*
Laboratory attendance week *x*	Lab attendance W*x*
Time spent on the platform on week *x*	Time W*x*
Ratio of during-laboratory to non-laboratory time accesses on week *x*	Lab access W*x*
Material covered based on the resources made available on week *x*	Coverage W*x*
Average time of the day the course material is accessed on week *x*	Hour access W*x*
Average lapse time students take to access the resources on week *x*	Checking W*x*
Ratio of on-campus to off-campus accesses on week *x*	On/off campus W*x*
Ratio of weekday to weekend accesses on *x*	Week(end) W*x*

Table 2 Details and course pass rates on groundtruth (training) data

Course	Academic year	Semester	Students	1st Exam Passing rate (%)	2nd Exam Passing rate (%)
CS2	2015/2016	2	149	44.30	46.98

above were extracted every week using the activity interactions and computer programming submissions. Every week, a classifier was built by concatenating the static student data, the dynamic features from previous weeks and that week's dynamic features in order to account for each student's characteristics, progression and engagement throughout the course.

The Empirical Error Minimization (EMR) approach was employed to calculate the misclassification error and determine which learning algorithm has the lowest empirical error from a bag of classifiers C (Devroye et al. 2013). We leveraged the Machine Learning Python library Scikit-learn (Pedregosa et al. 2011). Our approach is to pick a classifier that performs well on average for both classes (likely-to-pass and likely-to-fail) in the prediction problem. Generally, exam results are quite imbalanced as there are many more students who pass rather than fail an assessment. The resulting accuracy of a learning algorithm could be misinterpreted if we weigh the predictions based on the numbers per class. However, in this case, CS2 is a challenging class that requires a student to learn advanced computer programming concepts and the pass rates for computer-based programming assessments are typically not very high. The objective is still to identify students having issues and it is preferable to classify students "on the boundary" as likely to fail rather than not flagging them at all and miss the opportunity to intervene and help. See the training data pass rates in Table 2.

Table 3 Performance of the learning algorithms in the bag of classifiers when trained and cross-validated

Learning algorithm	Class	F1-score (%)	Precision (%)	Recall (%)
K-Neighbors	Fail	74.50	71.41	81.03
	Pass	59.81	68.80	58.74
Decision tree	Fail	72.53	72.14	76.07
	Pass	63.32	68.22	67.37
Random forest	Fail	75.39	71.64	83.07
	Pass	62.25	72.92	61.28
Logistic regression	Fail	73.13	72.85	76.33
	Pass	62.65	66.25	67.02
Linear SVM	Fail	70.23	71.35	72.80
	Pass	60.91	64.41	66.32
Gaussian SVM	Fail	64.52	50.47	94.79
	Pass	1.14	2.00	5.31

Our bag of classifiers contain simple learning algorithms like Logistic Regression, Support Vector Machines (SVM) with different kernels, a K-Neighbors classifier, a Decision Tree and a Random Forest ensemble classifier. See "Appendix" A for the list of classifiers we used.

Table 3 presents details on the performance for each learning algorithm in the bag of classifiers. These performance metrics indicate an average of the values for each of the 12 weeks of the semester. For each week, the resulting value is an average of the folds after performing tenfold cross validation.

The classifier selected was the K-Neighbors classifier (being K or the numbers of neighbors 12) which gave us high performance with metrics for F1, precision and recall for both classes compared with other classifiers trained throughout the semester. In addition, K-Neighbors gave us the highest F1 metric on weeks 5 and 6, and those weeks are key to identify who is struggling before their first assessment and help accordingly. Figure 3 shows the resulting average F1-metric between the folds for each week of the semester on the training data (and the shaded area is creating by adding and subtracting the average of the folds' standard deviation from the mean). Also, Fig. 4 indicates the F1 values for the Fail (likely-to-fail) class.

The SVM with a Gaussian kernel failed to learn properly and classified all students to belong to one of the classes for the first assessment and also for the second. The selected K-Neighbors classifier's predictions are statistically significant compared to the Gaussian SVM's ($p < 0.00001$). However, there are no statistically significant differences between K-Neighbor's predictions and the other classifiers. That indicates the data is limited and, even though, some learn slightly better than others, all but the Gaussian SVM perform similarly. Unfortunately we only have training data for one academic semester but in future years, we will have more data from student cohorts that will enable us to generalise better.

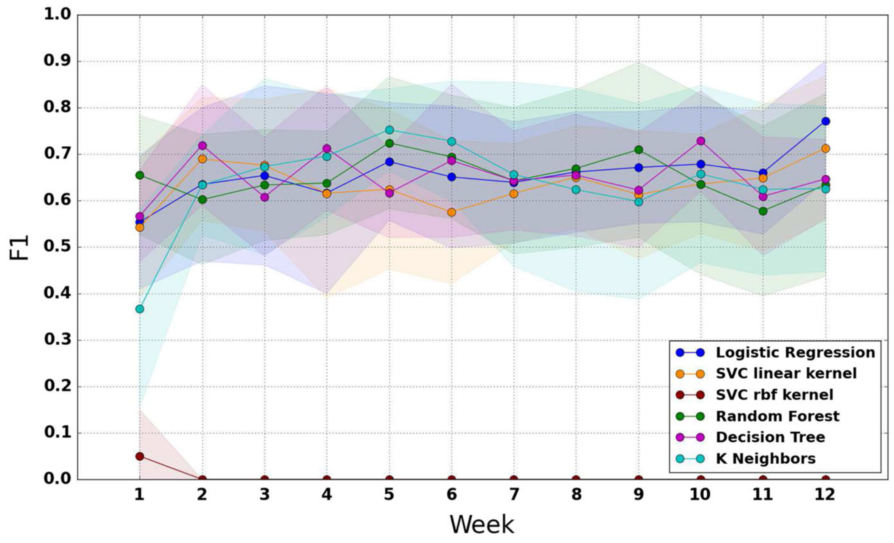


Fig. 3 Performance of the bag of classifiers for the F1 metric

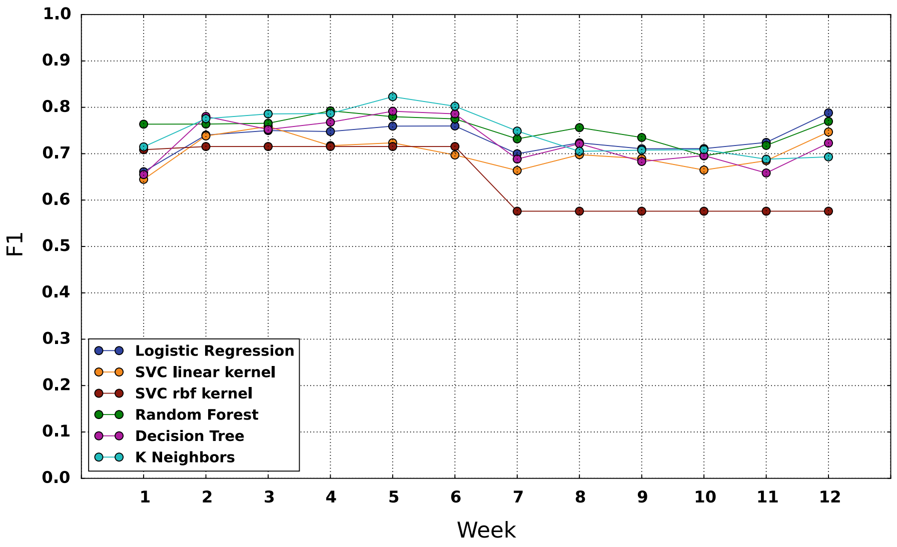


Fig. 4 Performance of the bag of classifiers for the F1 metric and the fail class

5.3 Importance of the features

We measured the predictive power of each of the set of individual features derived from the student digital footprint. For that, we used the linear and non-linear relationship between those values and the target, namely the next exam scores, using the Pearson and Spearman correlation coefficients and *p*-values. This was done to indicate the probability of no correlation and the null hypothesis to be incorrect. Table 4 shows

Table 4 2015/2016 Feature correlations with target values

Feature name	Pearson Coefficient (*p*-value)	Spearman Coefficient (*p*-value)
CS1 2nd Exam 2015/2016	0.35**	0.40**
Programming work week 1	0.53**	0.52**
Programming work week 2	0.60**	0.58**
Resources coverage week 3	0.39**	0.38**
Cumulative programming work week 3	0.64**	0.64**
Cumulative programming work week 4	0.70**	0.69**
Cumulative programming work week 5	0.72**	0.72**
Programming work week 6	0.73**	0.73**
Cumulative programming work week 7	0.62**	0.62**
Lab attendance week 10	0.24 ($p = 0.0027$)	0.28 ($p = 0.0007$)
Hours spent week 10	0.31 ($p = 0.0001$)	0.50**
Cumulative programming work week 12	0.67**	0.68**

**p-value < 0.0001

some examples of these features: *academic performance, static characteristics, interaction* and *programming* features. This analysis confirms the predictive power of our features and the *programming weekly* and *cumulative progress* features increasingly gain importance throughout the semester as students put more effort into the modules.

For instance, in order to predict the mid-semester exam performance for each student, the computer programming work features gain importance every week. For the second part of the semester, similar increasing importance is observed. To clarify, the cumulative computer programming work for week 3 is a different feature than the cumulative programming work from week 4 and both features will exist separately for the classifier in week 4.

In addition, an extra trees classifier, a type of forest that fits a number of randomized decision trees (Geurts et al. 2006), with 250 estimators is computed to also measure the importance of each feature. For each weekly learning function, an extra trees classifier is fitted and the features are ranked based on their associated normalized importance. Interestingly, we can observe how static features such as the exam for CS1 or the entry-to-university Mathematics score are important in the first week of the semester (see Fig. 5) while the *dynamic programming* work features and the effort students put in increasingly gain importance throughout the semester (see Fig. 6) relegating static features to the end of the ranked list of features.

A subset of the 10 highest ranked features each week can be used to develop a new model. This model would contain a weekly classifier with the top 10 features only. This approach for feature selection improved our metric values, see Fig. 7 (Hanley and McNeil 1982). This analysis also confirms the predictive power of the features. Other models using all the features, only dynamic features or even only programming work features do not work as well as selecting the top features for each week.

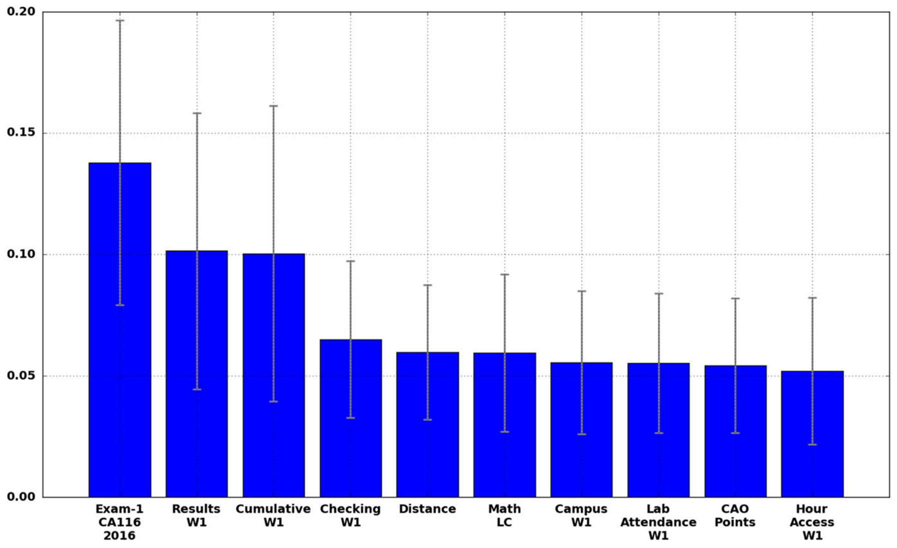


Fig. 5 Top features for week 1's learning function derived from an extra trees classifier

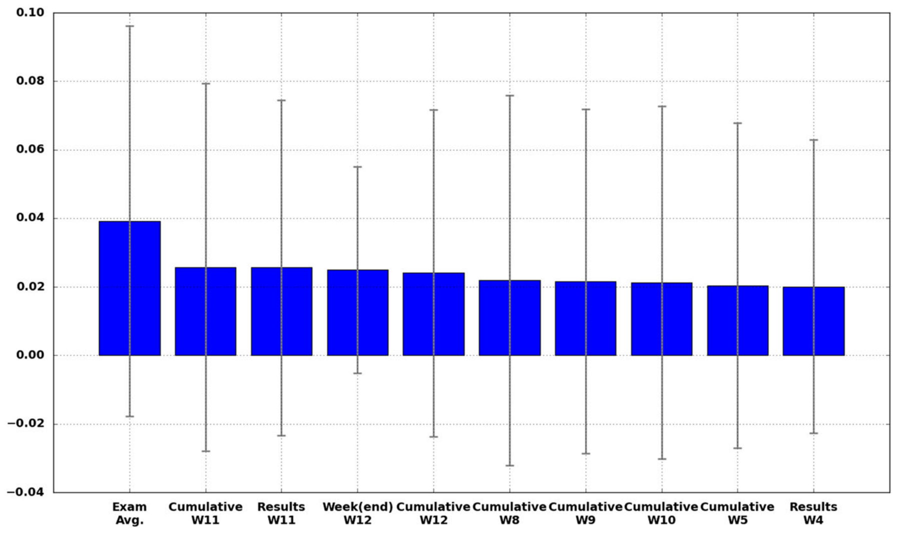


Fig. 6 Top features for week 12's learning function derived from an extra trees classifier

This retrospective analysis shows that we can successfully gather a rich student digital footprint about students' learning progress in computer programming. For CS2, we leverage the groundtruth data collected from the previous academic year, including student demographics, prior academic history, computer programming submissions and log interactions with the material, which was used to extract features and build models which embed patterns. Then, pseudo real-time predictions were generated with

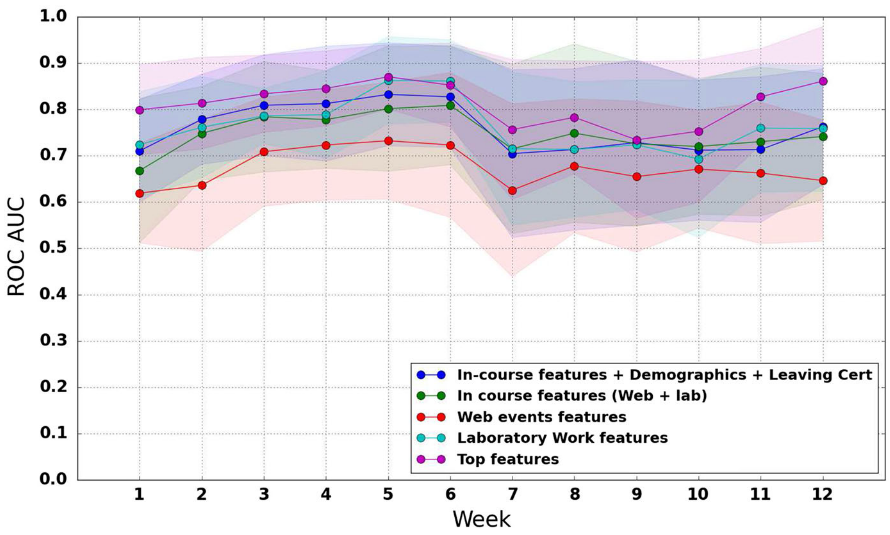


Fig. 7 Comparative feature analysis throughout the training semester

Table 5 2016/2017 Student opt-ins and opt-outs to interventions

Students	Replied	No-reply	Opt-ins (out of the replies)	Opt-outs
140	122 (87.14%)	18 (12.86%)	111 (90.98%)	11 (9.02%)

current students and incoming student data from the data sources mentioned above. The results will be analysed in Sect. 7.

6 Automatic and adaptive peer-programming student feedback

In computer programming modules, after the first laboratory exam in week 6, we started sending weekly notifications to students in the class via email.

We enabled a feature in the submission platform for the teaching of computing programming where students could opt-in or out from these notifications and read about the project. After the feature was enabled, students were not able to submit any programs before they chose to either opt-in or out. Table 5 shows the number of students who replied to the opt-in option. Students who did not reply were disengaged from the module from that time. From the replies, a significant number of students (91%) opted-in and they received weekly notifications from that moment onwards.

The feedback to each student was personalized in the following ways:

– By leveraging our weekly predictions and based on the associated probability of failing the next laboratory exam, students were ranked and divided into deciles. Hence, there were 10 custom messages we sent based on their performance. From *"For the last week our records show that you are engaging really well with the courseware and are well on top of module {{ course }}. Well done you."* for those

in the top 10% of the class to *"For the last week our records show that you are not engaging enough with the courseware for module {{ course }} and you really need to work harder. Please try to make more effort to keep up this week and if you are finding this difficult then do contact the Lecturer."* for those in the lowest 10% of the class, with other 8 custom messages in between.
- If a student did not spend any time on the platform, we added the message *"Remember, computer programming is a skill that requires practice and this module is no exception."* in bold.
- For each student we checked whether (s)he attended any of the lab sessions the week before and regardless of the predicted probability we acknowledged if they did or we added the message *Try to make it to the next lab session so the lecturer and tutors can help you resolve any issue."* if they did not.
- In addition, as a form of peer feedback, for each notification we included one computer programming suggestion if the student had submitted a program that failed any of the testcases and was incorrect. We developed a knowledge graph and based on the concepts the lecturer considered more important each week, we started suggesting programs for those gaps in knowledge. Typically, the most recent labsheet exercises were suggested first, then the previous labsheet exercises and so on. The order of the exercises selected from a particular labsheet is the normal order in the lab. The first exercise in the labsheet will be offered before the second if both were failed submissions for a particular student. The solution suggested is the closest program from a top-rated student in the class that week who got that program working as expected. The top students are the 10% highest ranked in that class from our predictions each week. We recommended the closest submission by text similarity between the programs after removing the comments.
- Students were given an explanation about the project and how the predictions are computed. They were also provided with support resources to reach out to the Lecturer of the module, our project or the Support Services at the University if they needed assistance.
- At the end of the note, students could find links to read the Terms and Conditions for the project, or unsubscribe from these notifications if desired. Nobody unsubscribed from these notifications throughout the semester.

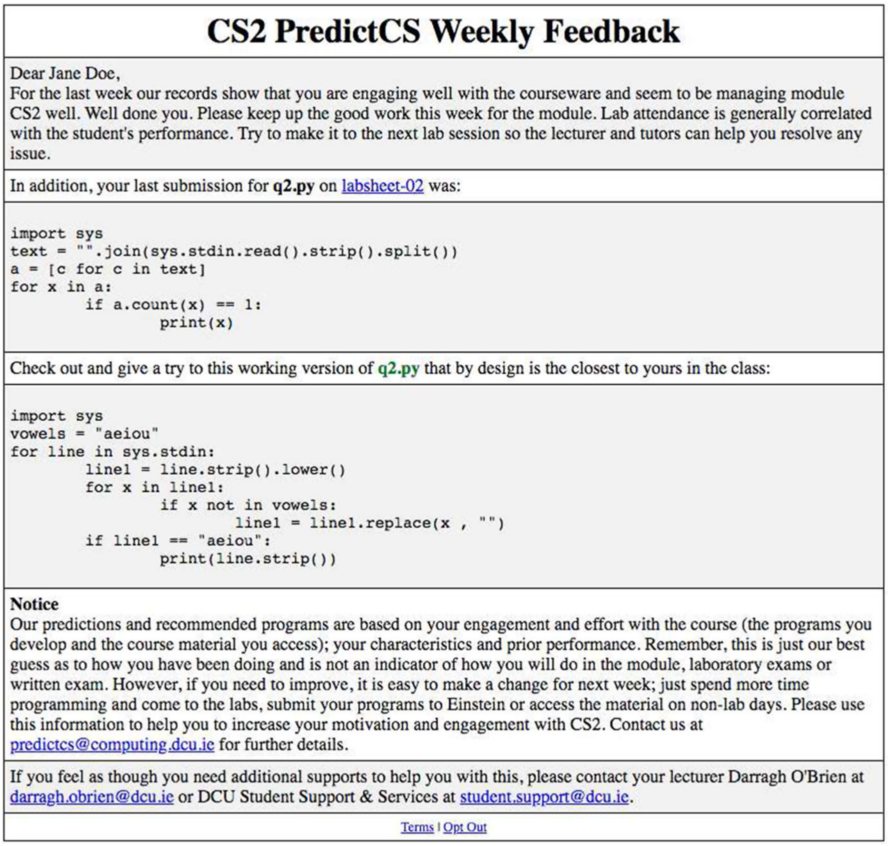
See Fig. 8 for a sample of the notifications.

7 Results

We will now analyse the results obtained by running predictions on data from 2016/2017's incoming CS2 students along with the feedback sent to them and what this means for the research questions proposed.

7.1 RQ1: Predictions

Predictions were calculated on a pseudo real-time basis every week for students who registered for the module during the second semester of 2016/2017 using the model



CS2 PredictCS Weekly Feedback

Dear Jane Doe,
For the last week our records show that you are engaging well with the courseware and seem to be managing module CS2 well. Well done you. Please keep up the good work this week for the module. Lab attendance is generally correlated with the student's performance. Try to make it to the next lab session so the lecturer and tutors can help you resolve any issue.

In addition, your last submission for **q2.py** on labsheet-02 was:

```
import sys
text = "".join(sys.stdin.read().strip().split())
a = [c for c in text]
for x in a:
        if a.count(x) == 1:
                print(x)
```

Check out and give a try to this working version of **q2.py** that by design is the closest to yours in the class:

```
import sys
vowels = "aeiou"
for line in sys.stdin:
        line1 = line.strip().lower()
        for x in line1:
                if x not in vowels:
                        line1 = line1.replace(x , "")
        if line1 == "aeiou":
                print(line.strip())
```

Notice
Our predictions and recommended programs are based on your engagement and effort with the course (the programs you develop and the course material you access); your characteristics and prior performance. Remember, this is just our best guess as to how you have been doing and is not an indicator of how you will do in the module, laboratory exams or written exam. However, if you need to improve, it is easy to make a change for next week; just spend more time programming and come to the labs, submit your programs to Einstein or access the material on non-lab days. Please use this information to help you to increase your motivation and engagement with CS2. Contact us at predictcs@computing.dcu.ie for further details.

If you feel as though you need additional supports to help you with this, please contact your lecturer Darragh O'Brien at darragh.obrien@dcu.ie or DCU Student Support & Services at student.support@dcu.ie.

Terms | Opt Out

Fig. 8 Anonymised customized notification sent to a student in CS2 on 2016/2017

trained with our groundtruth data. Individual reports were emailed to the Lecturers every week and posted on a visual analytics web application accessible to them at any time. In order to evaluate how our predictions performed, we compared the corresponding weeks' predictions with the actual results of the two laboratory exams that took place in weeks 6 (first exam) and 12 (second exam) in 2016/2017. Table 6 contains details of the accuracy of our predictions. CS2's predictions were run in 2016/2017 academic year on Semester 2.

One-hundred and thirty-three students registered in the 2016/2017 academic year. On the first mid-semester laboratory exam, 58.57% passed the test and our predictions had flagged 74.29% students as "at-risk" of failing that laboratory exam. We created a confusion matrix with the expected pass/fail and the actual results by looking at the true positives, true negatives, false positives and false negatives and from this we calculated accuracy, precision, recall and F1-score. We shared that information with the lecturer to get a feel of how accurate is the information they get automatically every week. In addition, we looked at the probability associated with failing the laboratory exam that the classification gives us and correlate that with the actual result that students get. See

Table 6 2016/2017 Module information, pass rates, at-risk prediction rates, prediction metric results and correlations for the incoming cohort

Information	1st Exam	2nd Exam
Number of students	133	133
Passing rate	58.57%	42.86%
Predicted at-risk rate	74.29%	44.29%
Model's accuracy	64.29%	77.14%
Model's precision	94.44%	67.95%
Model's recall	41.46%	88.33%
Model's F1-score	57.63%	76.81%
Pearson correlation coefficient (*p* value)	0.62 ($p < 0.0001$)	0.65 ($p < 0.0001$)
Spearman correlation coefficient (*p* value)	0.60 ($p < 0.0001$)	0.70 ($p < 0.0001$)

Table 7 Interventions sent to students

Course	Year	Notifications	Suggestions	# Corrected
CS2	2016/2017	438	181	21 (11.60%)

the last two rows for the linear and non-linear analyses. That indicates our confidence predictions are highly correlated with the actual results.

Individual reports containing lists of students and their associated probabilities of experiencing issues ahead of their next examination were sent to instructors each week and shared on our analytics web application. As the semester progressed, our early alert system gathered more information about students' progression and our classifiers were able to learn more as shown by the increased accuracy and F1-score measures and the decreasing number of students flagged as "at-risk" from the mid-semester exam to the end-of-semester's. In short, we could automatically distinguish in a better way, who is going to pass or fail the next laboratory exam. We could have compared the actual results with the predictions from all weeks but we felt the 2 weeks where the laboratory exams happened already showed their improved performance.

7.2 RQ2: Interventions

Notifications were sent to opt-in students for the second half of the semester classes as shown in Table 7. 438 notifications were sent to students, 181 of these contained a computer programming suggestion and the remaining did not have any program to suggest to the students. 21 of those programs suggested were corrected by the students after the recommendation was sent. Notifications were generated and sent on Monday morning for this course using the programs up to 2 weeks before as students were required to submit these programs (part of their continuous assessment) and the deadline for submitting them in that class was 2 weeks.

Table 8 Static features regarding student groups developed

Student Group	Academic Year	# Stud.	Average age [SD]	Gender Male/female	Average GPA [SD]
Whole class	2015/2016	149	18.94 [±3.08]	M: 133 F: 16	453.35 [±52.29]
Whole class	2016/2017	137	18.82 [±2.43]	M: 120 F: 17	440.10 [±34.47]
Opted IN	2016/2017	108	18.82 [±2.55]	M: 93 F: 15	440.62 [±34.82]
Opted OUT		11	18.82 [±0.72]	M: 10 F: 1	447.14 [±28.39]
Fixed a program	2016/2017	16	18.63 [±0.93]	M: 11 F: 5	443.93 [±37.37]
Did not fix any		51	18.53 [±1.57]	M: 45 F: 6	442.70 [±41.11]
Passed 1st exam	2016/2017	82	18.88 [±2.84]	M: 76 F: 5	439.60 [±35.08]
Failed 1st exam		58	18.75 [±1.66]	M: 44 F: 12	440.97 [±33.37]

Table 9 Differential improvement between students who fixed any of the programs suggested to them and students who were suggested code solutions but did not fix any

Student Group	Number Students	Avg. grade 1st Exam (%)	Avg. grade 2nd Exam (%)	Avg. grade Improvement (%)	Differential Improvement (%)
Opted IN	108	54.51	37.75	− 16.76	+ 16.05
Opted OUT	11	52.27	19.45	− 32.81	

In order to compare the impact these interventions may have had, we grouped the students in several ways and compare the differential performance between the first and second examination. The groups are the following:

(i) Students who opted in vs. students who opted out to receive these notifications during that academic year;
(ii) Students who fixed a program vs. students who did not fix any program recommended to them during that academic year;
(iii) Students who passed the first exam (higher-achievers) vs. students who failed that exam in between two different academic years.

First, we looked at the static data including demographics and academic performance for these groups of students. As shown in Table 8 there are no significant differences in between these groups of students.

We then created two groups from the students, the ones who opted in to receive the notifications and the students who opted out shown in Table 9. There were students who never replied to whether they wanted to opt-in or out and they were not added to either group. All students performed poorly on the second examination but students who opted-in to these notifications did not decrease their performance grade as much as students who opted-out. Based on 1 year of data, opt-in students might have benefited from these customized notifications that include the student's performance and code solutions.

Second, we created two different groups from the students, the ones who corrected any of the suggested programs with the solution outlined or another solution, and the ones who were suggested one or more computer programming suggestions but did not

Table 10 Differential improvement between students who fixed any of the programs they were suggested and students who were suggested code solutions but did not fix any

Student Group	Number Students	Avg. grade 1st Exam (%)	Avg. grade 2nd Exam (%)	Avg. grade Improvement (%)	Differential Improvement (%)
Fixed a program	16	32.81	27.62	− 5.19	+ 5.85
Did not fix any	53	45.28	34.24	− 11.04	

Table 11 Differential improvement between higher and lower-performing students over the two academic years

Academic Year	1st Exam	# Stud.	Avg. grade 1st Exam	Avg. grade 2nd Exam	Avg. grade Improvement (%)
2015/2016	'Passed'	66	75.23% [± 20.08%]	55.06% [± 29.93%]	− 20.17
	'Failed'	83	14.70% [± 13.65%]	24.40% [± 24.74%]	+9.70%
Differential:					+29.87%
2016/2017	'Passed'	82	76.22% [± 21.70%]	47.85% [± 26.42%]	− 28.37
	'Failed'	58	8.62% [± 11.88%]	12.02% [± 14.64%]	+3.40
Differential:					+31.76

correct any and these are shown in Table 10. There were 16 students who corrected some of their failing programs as suggested and 53 who were suggested computer programming code solutions but they did not. For the ones who did, their average grade on the second exam was more than 5 points below. However, for the ones that did not, their's was more than 11 points. Again, and based on 1 year of data, students that fixed any of their programs might have benefited by learning from programs from their own peers that are offered to them as advice for failed submissions.

As an alternative way to view the results, we also group students into higher and lower-performing groups based on their results in the first assessment in order to measure whether notifications had any impact on their subsequent performance. Table 11 shows how the the gap between the students who passed the first assessment ('Passed') vs. students who failed that assessment ('Failed') get reduced on the second assessment. That is likely as the group who failed the first exam had more room for improvement and for the group who passed that exam maintaining their grade is already an accomplishment. In this course, students who failed CS2's first laboratory exam improved almost 2 points on average respectively on 2016/2017 when the predictions were run, students were ranked, programs were suggested and notifications were sent to students. The differential improvement between the two groups of students, and the ones who passed their first assessments and the ones who failed, are statistically significant ($p < 0.0001$) for both academic years. In both scenarios, students who learned from the programs suggested and lower-performing students, showed a learning improvement over the two other groups.

Table 12 Survey responses from students about the project with respect to CS2

Response rate (%)	Q1 (%)	Q3	Q4 (%)	Q5 (%)	Q6 (%)
75.71	93.40	3.49 ⋆	33.70	85.15	83

7.3 RQ3: Students and lecturers have their say

Overall, feedback was very positive from both students and lecturers and responses can be found in Table 12. The survey results were anonymized. Most students would recommend this system to students attending the same module next year or would like to see this system included in other modules as shown in answers to questions 5 and 6 respectively.

The questionnaire was completed on the second-last day of the semester classes for CS2 during an evaluation for another module where all CA first-year students should have attended. That allowed us to gather a good number of responses. The response to the first question shows the percentage of students who opted-in and the fourth shows whether they ran any of the suggested programs. The responses in both questions are inaccurate as, for instance, some students claimed they opted-in when they did not or they were not suggested any program when in reality they were. That indicates that some students may not check their mail regularly or they opted in without realising what they were signing up for or what the system would do with their digital footprint. Email notifications via email may not be the best way to communicate with students as some of them pointed out in the improvements and comments section and a better way to measure how they interact with these customized messages should be leveraged.

The last question regarding how students would improve the system had really interesting comments. However, students who were doing well or very well, were getting a similar response every week and the notification might seem monotonous. In general, students demanded a more personalised notification and some other additional learning resources. Finally, the following are some positive and negative quotes, comments and suggested improvements from students for this last question of the survey:
A good amount of students demanded more personalised feedback:

– “More detailed responses, where you can improve”
– “More varied responses”
– “Give more personalised feedback”
– “Maybe more detailed feedback”
– “More in depth analysis of progress, more precise areas to focus on”

Some students also asked for other features:

– “Feedback on each step of each task would be good, such as, better ways to do things”
– “Have it from the start of the year”
– “Send a(n) end of module feedback of the whole module”
– “Maybe more suggested working programs”
– “Give advice in what the student is performing poor in”

Others did not enjoy notifications that much as they could be very repetitive for students who are well on top of the module and do not have any failed programs for which they could get suggestions:

– “Less repetition / automation. (It) would be nice to receive other feedback besides “you are doing OK”.”
– “Always said the same thing”
– “More information relevant to how you are doing, it is too vague”

Overall the feedback was very positive and some students were motivated with the weekly notifications:

– “It gave me confidence about the module. It gave me reassurance as to how I was getting on.”
– “Good service, very helpful and effective way to manage your module”

In addition, the lecturer of this module said “(He) would be happy for you to run further experiments in future deliveries of CS2”. “(He) liked the fact you could tailor the release of concepts to students in order to keep pace with the delivery of the module e.g., avoiding sharing solutions by advanced students that used lambda expressions not yet covered”.

8 Conclusions and future work

8.1 Summary

This paper describes a new research methodology that combines diverse data sources in Higher Education to classify students and distinguish students at risk. Personalised notifications were sent to students regarding their progression and suggested course material and code they should view based on the engineered predictive models. Due to these models involve sets of static and dynamic features covering ranges of students offline resources (i.e., demographics) and online information (i.e., behavioural logs), which permit to be further extrapolated to other domains as appropriately according to the data availability. The predictive students’ performance outcomes provide tremendous value to the at-risk students directly as well as the lecturers to be better at identifying them and to react upon. In addition, in large classes, where lecturers are not able to personalise their teaching and invest enough class time with each individual student, this methodology can supply the penalisation asynchronously by suggesting material and program code to students, keep them more engaged, improve their success rates and aid their learning of computer programming skills.

8.2 Limitations

There are a number of limitations in this work, they are listed as following:

– The features we extracted from the diverse data sources at our university are mainly student records and logs. However, there are still various of other data sources not

considered in the current study, such as video or speech. These can potentially shed more insights and understandings on some detail level of how students learn and progress in Higher Education. Additionally, the availability of more data from previous student cohorts would potentially enable us to develop more complex models and also higher-level representations of student learning by stacking linear models with advanced machine learning techniques such as Deep Learning.
- Modelling is not perfect and has a probability of error. Some students will be mis-classified. Students who are predicted as likely to fail but in reality who end up passing are not a concern for us at this point. We may have affected their learning with our notifications. However, we want to minimize the students that we classify as likely to pass (false positive) or not-at-risk when they actually fail (false negative). For that, we maximize the fail class on the grountruth data when training.
- Students get bombarded with emails from lecturers, faculty, Student's Union and so on. Some students who opted-in to receive the personalised notifications may have never looked at them. Therefore, to better quantify the effectiveness of the use of personalised notification, the opted-in and opted-out statistics may be still too naive. Thus, the next version of this project has already implemented the tracking method to record student's access of suggested resources and materials.

8.3 Contributions

In this work, the main contributions are a new implementation of a predictive analytics system that aggregates multiple sources of students' digital footprints from blended classroom settings. Advanced Data Mining techniques are applied to engineer models to provide real-time prediction and dynamic feedback. This approach incorporates non-subject-specific static features and subject-specific dynamic student data features. As a result, the make-up of the predictive models can be extrapolated and further scaled to other blended classrooms or to other subjects. Additionally, not only is the approach we take generic, it also permits applicability to limited data sets (i.e., laboratory material behavioural logs only) in order to be beneficial in helping students in need. Most importantly, the generated predictions allow us to generate adaptive feedback for each student according to each student's progression and provide guidance when in need.

8.4 Future work

This new implementation of a multimodal predictive analytics system has proven to be useful to predict students struggling with their learning during the semester. Additionally, with just 1 year of training data, predictions turned out to be surprisingly accurate. In following years we will be able to generalise far better by having more student cohorts from which to extract usage patterns. In addition, we are planning to add more modalities to the system such as video recordings, the use of the laboratory resources or physical access to buildings in our university to better model their behaviour and to guide their learning.

The approach employed for the recommendations on computer programming code was to suggest the closest text program from top-ranked students in the same class that year. The system could be further advanced by tokenizing the programs, identifying variables and choosing the closest programming submission, both syntactically and semantically. In contrast, we explored Collaborative Filtering as used in recommender systems today by looking at the closest person to you in the class or within the top-students group, and to recommend one of their programs from the closest person. Assuming programming design learning is constant, like tastes in a movie recommendation engine, students can have programs suggested from their closest peer in the class. However, we were more interested in students being able to identify what is wrong with their problems at a glance and the closest text solution from a top-student worked quite well, particularly for shorter programs. We also tested crowdsourcing solutions by recommending the program that had been uploaded the most. Interestingly, even lecturers are now providing sample solutions and explanatory code after seeing the students' positive feedback to these notifications.

Harnessing more student modalities was beneficial to better understand the students' behaviour. Guiding students by suggesting where to focus has proven to have an effect on students who were submitting the programs suggested afterwards and on lower-performing students. However, students might have performed better than the model expected, which is our end goal and our model was trained with no interventions being carried out on the groundtruth data.

We are eager to provide our students with more detailed computer programming recommendations, suitable material and other actions to fill the knowledge programming gaps they may have while learning CS programming design. This study has proven to successfully create a digital footprint we can leverage by combining several sources of student data. Our research methodology selects a learning algorithm and a subset of predictors on data models and enabled us to identify students having issues in computer programming courses and to provide them with timely interventions. The system and these notifications have been implemented in further computer programming modules in order to have a broader impact. In addition, we have added learning resources for students to access based on their progression such as labsheets and slides. Lecturers are also posting new code solutions with explanations and desired solutions as part of their material based on the outcome of this research and survey responses. We will also be more vocal about the project so more students can opt-in, understand and benefit from this research.

Finally, since the practical aspects of this work was carried out the GDPR legislation has been enacted in Europe (in May 2018). The work has been approved by the University's Research Ethics Committee though We realise that for future instantiations of this work we need to remove data generated by students who have chosen to opt-out or who, even in mid-semester, choose to opt out. That means that we could not recommend a computer program submitted by a student who opted out, to another student in the same class, but we can use aggregated information such as access logs and statistics about demographics, in mining patterns for subsequent students.

Acknowledgements This research was supported by the Irish Research Council in association with the National Forum for the Enhancement of Teaching and Learning in Ireland under project number

GOIPG/2015/3497, by Science Foundation Ireland under grant number 12/RC/2289, and by Fulbright Ireland. The authors are indebted to Dr. Stephen Blott who developed the grading platform and Dr. Darragh O'Brien, lecturer on the module which is the subject of this work, for their help. We would also like to thank all students who participated in this initiative for their comments and feedback and to the anonymous reviewers for their helpful and constructive feedback.

Appendix A: Bag of classifiers

- Logistic Regression Classifier with regularization ($C = 1$) and L2 penalty.
- SVM with a Linear kernel with regularization ($C = 1$).
- SVM with a Gaussian kernel with regularization ($C = 1$) and kernel coefficient ($\gamma = 0.7$).
- Random Forest Classifier with 10 trees in the forest.
- A Decision Tree Classifier with best split looking at all nodes if necessary.
- A K-Neighbors Classifier with neighbors $K = 12$, uniform weights and using the Euclidean distance between the nodes.

References

Altadmri, A., Brown, N.C.C.: 37 million compilations: investigating novice programming mistakes in large-scale student data. In: Proceedings of the 46th ACM Technical Symposium on Computer Science Education, pp. 522–527. ACM (2015)

Arnold, K.E., Pistilli, M.D.: Course signals at purdue: using learning analytics to increase student success. In: Proceedings of the 2nd International Conference on Learning Analytics and Knowledge, pp. 267–270. ACM (2012)

Azcona, D., Hsiao, I.H., Smeaton, A.F.: PredictCS: personalizing programming learning by leveraging learning analytics. In: Companion Proceedings of the 8th International Conference on Learning Analytics and Knowledge (LAK 2018), pp. 462–468 (2018)

Azcona, D., Smeaton, A.F.: Targeting at-risk students using engagement and effort predictors in an introductory computer programming course. In: European Conference on Technology Enhanced Learning (EC-TEL'17), pp. 361–366. Springer, NY (2017)

Azcona, D., Corrigan, O., Scanlon, P., Smeaton, A.F.: Innovative learning analytics research at a data-driven HEI. In: Third International Conference on Higher Education Advances. Editorial Universitat Politecnica de Valencia (2017)

Blikstein, P., Worsley, M.: Multimodal learning analytics and education data mining: using computational technologies to measure complex learning tasks. J. Learn. Anal. **3**(2), 220–238 (2016)

Bloomfield, A., Groves, J.F.: A tablet-based paper exam grading system. In: ACM SIGCSE Bulletin, Vol. 40, No. 3, pp. 83–87. ACM (2008)

Boyer, K.E., Phillips, R., Ingram, A., Ha, E.Y., Wallis, M., Vouk, M., Lester, J.: Investigating the relationship between dialogue structure and tutoring effectiveness: a hidden markov modeling approach. Int. J. Artif. Intell. Educ. **21**(1–2), 65–81 (2011)

Brooks, C., Thompson, C.: Predictive modelling in teaching and learning. In: Lang, C., Siemens, G., Wise, A.F., Gasevic, D. (eds.) The Handbook of Learning Analytics, 1st edn, pp. 61–68. Society for Learning Analytics Research (SoLAR), Alberta (2017)

Buffardi, K., Edwards, S.H.: Effective and ineffective software testing behaviors by novice programmers. In: Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research, pp. 83–90. ACM (2013)

Burleson, W.: Affective Learning Companions: Strategies for Empathetic Agents with Real-time Multimodal Affective Sensing to Foster Meta-cognitive and Meta-affective Approaches to Learning, Motivation, and Perseverance. Ph.D. Thesis, Massachusetts Institute of Technology (2006)

Carter, A.S., Hundhausen, C.D., Adesope, O.: The normalized programming state model: Predicting student performance in computing courses based on programming behavior. In: Proceedings of the Eleventh Annual International Conference on International Computing Education Research, pp. 141–150. ACM (2015)

Cheang, B., Kurnia, A., Lim, A., Oon, W.C.: On automated grading of programming assignments in an academic institution. Comput. Educ. **41**(2), 121–131 (2003)

Chen, W., Looi, C.K.: Group scribbles-supported collaborative learning in a primary grade 5 science class. In: Productive Multivocality in the Analysis of Group Interactions, pp. 257–263. Springer (2013)

Conati, C.: Probabilistic assessment of user's emotions in educational games. Appl. Artif. Intell. **16**(7–8), 555–575 (2002)

Conijn, R., Chris, S., Ad, K., Uwe, M.: Predicting student performance from LMS data: a comparison of 17 blended courses using Moodle LMS. IEEE Trans. Learn. Technol. **10**(1), 17–29 (2017)

Corrigan, O., Smeaton, A.F., Glynn, M., Smyth, S.: Using educational analytics to improve test performance. In: Design for Teaching and Learning in a Networked World, pp. 42–55. Springer (2015)

Denny, P., Luxton-Reilly, A., Hamer, J.: Student use of the peerwise system. In: ACM SIGCSE Bulletin, Vol. 40, No. 3, pp. 73–77. ACM (2008)

Devroye, L., Györfi, L., Lugosi, G.: A Probabilistic Theory of Pattern Recognition, vol. 31. Springer (2013)

Diana, N., Eagle, M., Stamper, J.C., Grover, S., Bienkowski, M.A., Basu, S.: An instructor dashboard for real-time analytics in interactive programming assignments. In: LAK, pp. 272–279 (2017)

Edwards, S.H., Perez-Quinones, M.A.: Web-cat: automatically grading programming assignments. In: ACM SIGCSE Bulletin, Vol. 40, pp. 328–328. ACM (2008)

Gehringer, E.F.: Electronic peer review and peer grading in computer-science courses. ACM SIGCSE Bull. **33**(1), 139–143 (2001)

Geurts, P., Ernst, D., Wehenkel, L.: Extremely randomized trees. Mach. Learn. **63**(1), 3–42 (2006)

Guerra, J., Sahebi, S., Lin, Y.R., Brusilovsky, P.: The problem solving genome: Analyzing sequential patterns of student work with parameterized exercises. The 7th International Conference on Educational Data Mining EDM 2014, pp. 153–160 (2014)

Hanley, J.A., McNeil, B.J.: The meaning and use of the area under a receiver operating characteristic (ROC) curve. Radiology **143**(1), 29–36 (1982)

Hsiao, I.H.: Mobile grading paper-based programming exams: automatic semantic partial credit assignment approach. In: European Conference on Technology Enhanced Learning, pp. 110–123. Springer (2016)

Hsiao, I.H., Lin, Y.L.: Enriching programming content semantics: an evaluation of visual analytics approach. Comput. Hum. Behav. **72**, 771–782 (2017)

Hsiao, I.H., Sosnovsky, S., Brusilovsky, P.: Guiding students to the right questions: adaptive navigation support in an e-learning system for java programming. J. Comput. Assist. Learn. **26**(4), 270–283 (2010)

Hsiao, I.H., Pandhalkudi Govindarajan, S.K., Lin, Y.L.: Semantic visual analytics for today's programming courses. In: Proceedings of the Sixth International Conference on Learning Analytics and Knowledge, pp. 48–53. ACM (2016)

Hsiao, I.-H., Huang, P.-K., Murphy, H.: Integrating programming learning analytics across physical and digital space. IEEE Trans. Emerg. Top. Comput. **1**, 1–12 (2017a)

Hsiao, I.H., Huang, P.K., Murphy, H.: Uncovering reviewing and reflecting behaviors from paper-based formal assessment. In: Proceedings of the Seventh International Learning Analytics and Knowledge Conference, pp. 319–328. ACM (2017b)

Ihantola, P., Vihavainen, A., Ahadi, A., Butler, M., Borstler, J., Edwards, S.H., Isohanni, E., Korhonen, A., Petersen, A., Rivers, K., et al.: Educational data mining and learning analytics in programming: literature review and case studies. In: Proceedings of the 2015 ITiCSE on Working Group Reports, pp. 41–63. ACM, NY (2015)

Jackson, D., Usher, M.: Grading student programs using ASSYST. In: ACM SIGCSE Bulletin, vol. **29**, pp. 335–339. ACM (1997)

Jadud, M.C., Dorn, B.: Aggregate compilation behavior: Findings and implications from 27,698 users. In: Proceedings of the Eleventh Annual International Conference on International Computing Education Research, pp. 131–139. ACM (2015)

Lin, C.P., Chen, W., Yang, S.J., Xie, W., Lin, C.C.: Exploring students' learning effectiveness and attitude in group scribbles-supported collaborative reading activities: a study in the primary classroom. J. Comput. Assist. Learn. **30**(1), 68–81 (2014)

Looi, C.K., Lin, C.P., Liu, K.P.: Group scribbles to support knowledge building in jigsaw method. IEEE Trans. Learn. Technol. **1**(3), 157–164 (2008)

Lu, Y., Hsiao, I.H.: Personalized information seeking assistant (PISA): from programming information seeking to learning. Inf. Retr. J. **20**(5), 433–455 (2017)

Martinez-Maldonado, R., Dimitriadis, Y., Martinez-Mones, A., Kay, J., Yacef, K.: Capturing and analyzing verbal and physical collaborative learning interactions at an enriched interactive tabletop. Int. J. Comput. Support. Collab. Learn. **8**(4), 455–485 (2013)

Martinez-Maldonado, R., Clayphan, A., Yacef, K., Kay, J.: MTFeedback: providing notifications to enhance teacher awareness of small group work in the classroom. IEEE Trans. Learn. Technol. **8**(2), 187–200 (2015)

Murphy, H.E.: Digitalizing paper-based exams: an assessment of programming grading assistant. In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, pp. 775–776. ACM (2017)

Ochoa, X.: Multimodal learning analytics. In: The Handbook of Learning Analytics, 1 ed., C. Lang, G. Siemens, A. F. Wise and D. Gasevic, Eds., pp. 129–141. Society for Learning Analytics Research (SoLAR), Alberta, Canada (2017)

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)

Piech, C., Sahami, M., Koller, D., Cooper, S., Blikstein, P.: Modeling how students learn to program. In: Proceedings of the 43rd ACM technical symposium on Computer Science Education, pp. 153–160. ACM, New York (2012)

Price, T.W., Zhi, R., Barnes, T.: Hint generation under uncertainty: the effect of hint quality on help-seeking behavior. In: International Conference on Artificial Intelligence in Education, pp. 311–322. Springer, Berlin (2017)

Prieto, L.P., Sharma, K., Kidzinski, L., Dillenbourg, P.: Orchestration load indicators and patterns: In-the-wild studies using mobile eye-tracking. IEEE Transactions on Learning Technologies (2017)

Ritterfeld, U., Shen, C., Wang, H., Nocera, L., Wong, W.L.: Multimodality and interactivity: connecting properties of serious games with educational outcomes. Cyberpsychol. Behav. **12**(6), 691–697 (2009)

Rivers, K., Koedinger, K.R.: Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. Int. J. Artif. Intel. Educ. **27**(1), 37–64 (2017)

Singh, A., Karayev, S., Gutowski, K., Abbeel, P.: Gradescope: a fast, flexible, and fair system for scalable assessment of handwritten work. In: Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale, pp. 81–88. ACM (2017)

Sosnovsky, S., Peter, B.: Evaluation of topic-based adap-tation andstudent modeling in QuizGuide. User Model. User-AdaptedInteract. **25**(4), 371–424 (2015)

Tempelaar, D.T., Rienties, B., Nguyen, Q.: Towards actionable learning analytics using dispositions. IEEE Trans. Learn. Technol. **10**(1), 6–16 (2017)

VanLehn, K., Cheema, S., Wetzel, J., Pead, D.: Some less obvious features of classroom orchestration systems. In: Educational Technologies: Challenges, Applications and Learning Outcomes. Nova Science Publishers, Inc. (2016)

Vea, L., Rodrigo, M.M.: Modeling negative affect detector of novice programming students using keyboard dynamics and mouse behavior. In: Pacific Rim International Conference on Artificial Intelligence, pp. 127–138. Springer, Berlin (2016)

David Azcona is a Ph.D. candidate in the Insight Centre for Data Analytics at Dublin City University in Ireland. David visited Arizona State University as a Fulbright research scholar in 2017/2018. His research focuses on personalizing Computer Science Education. His interests are representational learning for programming learning, machine learning and deep learning and Data Science for social good.

I-Han Hsiao is an Assistant Professor in the School of Computing, Informatics and Decision Systems Engineering at Arizona State University. Her research focuses on the intersection of Informatics and Com-

putational Technologies for Learning, involving Adaptive Educational Technologies, Computer Science Data Analytics, Open User Modeling, and Educational Data Mining.

Alan F. Smeaton is a Professor of Computing at Dublin City University and Founding Director of the Insight Centre for Data Analytics. Previously Alan has been Head of School and Dean of Faculty at DCU. He is an IEEE Fellow, a member and Gold Medal winner of the Royal Irish Academy and Chair of ACM SIGMM (Multmedia).