Casey and Azcona *International Journal of Educational Technology in Higher Education* (2017) 14:4
DOI 10.1186/s41239-017-0044-3

International Journal of Educational Technology in Higher Education

RESEARCH ARTICLE Open Access

Utilizing student activity patterns to predict performance

Kevin Casey[1*] and David Azcona[2]

* Correspondence: kevin.casey@nuim.ie
[1]Maynooth University, Maynooth, Ireland
Full list of author information is available at the end of the article

Abstract

Apart from being able to support the bulk of student activity in suitable disciplines such as computer programming, Web-based educational systems have the potential to yield valuable insights into student behavior. Through the use of educational analytics, we can dispense with preconceptions of how students consume and reuse course material. In this paper, we examine the speed at which students employ concepts which they are being taught during a semester. To show the wider utility of this data, we present a basic classification system for early detection of poor performers and show how it can be improved by including data on when students use a concept for the first time. Using our improved classifier, we can achieve an accuracy of 85% in predicting poor performers prior to the completion of the course.

Keywords: Learning Analytics, Data Mining, Virtual Learning Environments, Student behavior, Early intervention

Introduction

Computer Science degree programmes have typically suffered from high attrition rates (Beaubouef & Mason, 2005; Biggers, Brauer, & Yilmaz, 2008). In the UK, Dehnadi and Bornat (2006) report that the majority of undergraduate students fail introductory courses and the authors note that, despite this, little hard analysis has been performed to examine why this is the case.

As a result of this high degree of attrition, there is a shortage of suitably skilled graduates to fill the needs of ICT companies (Lang, McKay, & Lewis, 2007; Lister, 2008; Slonim, Scully, & McAllister, 2008). Estimates vary widely, but in the US for example, there was between 1.25 million (Thibodeau, 2011) and 400,000 (Davis, 2011) unfilled IT jobs in 2011 (when the US unemployment rate was 9%). Estimates of up to half a million unfilled IT jobs in the US have led to the $100 million TechHire initiative (Lapowsky, 2015). In one recent report (Huntley & Young, 2015), it is projected that only 30% of 1.4 million specialist IT jobs in 2020 will be filled by suitably qualified graduates.

Tackling this problem is non-trivial and there are a number of points at which it can be addressed. One of the obvious places to address this skills shortage is at the undergraduate stage which students have traditionally found difficult to complete successfully. There is no shortage of innovation in teaching techniques to try something different, for example pair programming (Teague & Roe, 2007) and problem-based learning (O'Kelly et al., 2004; O'Kelly, Mooney, Bergin, Gaughran, & Ghent, 2004).

While based upon generally sound principles, many of these curriculum changes had a focus on changing existing practice without necessarily analyzing what the underlying problem is with Computer Science education.

With the advent of analytics and big data in general (Manyika et al., 2011), Learning Analytics (Siemens & Long, 2011) has become more widespread and has the potential to make significant contributions to understanding learner behavior. Good quality data underpins Educational Analytics and Virtual Learning Environments (VLEs) or Learning Management Systems (LMSs) have the potential to generate the necessary data in the quality and quantity required. With appropriate processing, this learner-produced data can then provide valuable insight into what is actually happening in the learning process, and suggest ways in which educators can make improvements, for example identifying students at risk of dropping out or needing additional support in the learning process.

The analysis of data from VLEs is well established. Casey and Gibson (2010) examined Moodle data for fifteen modules in three different programmes. The data stored in the system about the activities of both teachers and students typically represents when, where, and who performed particular actions. They found some interesting correlations that link with high performance such as daily module logins, the amount of material reviewed, or Moodle usage over a weekend. Surprisingly, they found that extremely high student activity levels on Moodle for certain modules are often a negative indicator for student performance. This could be used to detect students with difficulties ahead of time through early intervention.

Early intervention is a key goal of Learning Analytics (Macfadyen & Dawson, 2010), and several universities have schemes in place to identify at-risk students. Course Signals (Arnold & Pistilli, 2012) is a system at Purdue University that feeds information back to faculty members on the likelihood of student success. Based on data from four different categories (performance, effort, prior academic history, and student characteristics), faculty members can identify at-risk students and send them personalized emails to assist them.

Initiatives such as Course Signals are a proven method for improving student retention, but they are typically a one-size-fits-all solution. However, with a tailored solution for subject areas such as computer programming, where the majority of student activity takes place online, the opportunities for fine-grained detailed analysis are far greater (Blikstein, 2011). These subject areas, programming in particular, have been the areas where students have traditionally struggled.

An interesting project, which has many parallels to the VLE discussed in this paper, is the Blackbox project (Brown, Kölling, McCall, & Utting, 2014) where users of the popular BlueJ IDE can opt to contribute analytics on their programming. Brown et al. report that over one hundred thousand users have signed up. This project has the potential has the potential to yield data at an unprecedented scale.

Romero-Zaldivar, Pardo, Burgos, and Kloos, (2012) report on a successful trial examining the viability of virtual machines within a learning analytics context. The authors describe how they equipped each student in a second year undergraduate engineering course with an instrumented virtual machine. This virtual machine recorded data as the students used it, and submitted it to a central server. While the analytics are also high-level, the authors do note that useful actionable information was obtained that could be fed back into the teaching process. Specifically, they were able to observe that hardly any students used a particular tool (the debugger) during the course.

Berland, Martin, Benton, Petrick Smith, and Davis (2013) discuss the importance of *tinkering* in the learning process. To measure it, they capture the various states of a program as a student edits it. The authors then analyze how students move though the state space of potential programs. While they found that different students took diverse paths, they were able to identify three phases to their learning. The result of their work is the EXTIRE framework, which characterizes the transitions that take place during tinkering. Other research concentrates on measuring the efficacy of tutoring software determining how robust learning is in an online tutor (Baker, Gowda, & Corbett, 2010; 2011), knowledge that can then be fed back into the instructional design process.

Ahadi, Lister, Haapala, and Vihavainen, (2015) outline a promising classifier (based on decision trees) approach to predicting low-performing and high-performing programming students. Based on a number of features, the most effective being how the students performed on a subset of Java programming exercises they were given during the course. Using this approach, the authors report an accuracy of between 70 and 80%.

One notable feature of the VLE system presented in this paper is the potential for real-time analytics. Edwards (2013) notes that many VLEs do not provide real time feedback to educators. The platform presented in Fine grained data section, however, has the potential to operate in real-time with minimal work and, as such, has the ability to be a useful tool in the context of a laboratory session, where a tutor could intervene if a student was deemed to be struggling.

Fine grained data

Over a 3 year period at <university redacted>, we have developed a new specialized platform for module delivery, which has been trialed for second year undergraduate Computer Science students on one of their core programming modules (Computer Architecture and Assembly Language Programming). This platform, SOCS (Shared Online CPU Simulator), handles both module content and general learning activities within the module, all through a web-browser (Fig. 1). While module content delivery is reasonably standard, being handled by mainstream VLEs such as Moodle, our customized platform allows for far more fine-grained analysis of how students consume material, for example being able to determine how much time students are spending on individual lecture slides.

SOCS is implemented as a client-side Typescript/Javascript program. Students authenticate with the system using their campus login. The client-side application interacts with a CoreOS hosted server (using Docker containers running Node.js) to retrieve learning materials such as course notes and weekly lab exercises. Usage data is collected by the Javascript client in JSON format, periodically compressed using a Javascript zlib library and posted to the server via a RESTful interface. To reduce load on the server, the data remains compressed until analysis is required.

The application simulates an 8-bit x86 microprocessor with a restricted amount of memory. Loosely based on Baur's Microprocessor Simulator (Baur, 2006), a Microsoft Windows application, the simulator allows students to create small assembly programs, compile them, and execute them. As programs are executed, students can see a visual representation of CPU registers, memory, and a host of connected devices. Students can either run programs freely (adjusting their speed via slider) or can step through the programs instruction by instruction. Being browser-based, the application can be run in

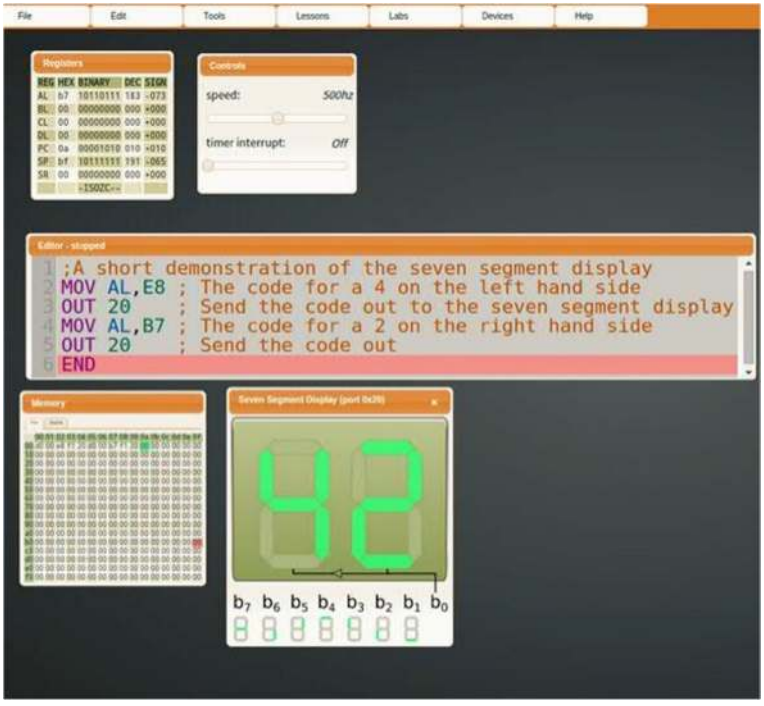

Fig. 1 VLE for Assembly Language Programming

any OS with a reasonable web browser, though only Chromium browser was supported actively. Students could save their work and sessions on any computer, and resume sessions when they logged in elsewhere.

Learning materials were also integrated into SOCS. A series of 20 lessons, identical to lecture slides, were made available in the platform. Students were encouraged to break away from the learning materials to try concepts out in the simulator, hence the tight coupling between the simulator and the learning material. Additionally, eight lab exercises were also available. The labs and lessons were represented as HTML5 slides using the popular Reveal.js library (Ferreira, 2013). Although it was decided against it at the time, due to the experimental nature of the software, the learning materials could have been decoupled from the simulator and analytics collected on the server side via a Tin-Can API (Kelly & Thorn, 2013).

Hosting of general learning activities in SOCS is possible because the module is largely a programming module. We have been able to move the tools that typically would have been used on the desktop into the browser itself, allowing students to program wherever they have a web-browser, with no need to install additional software. As students interact with the system, fine-grained data on their interactions is recorded centrally with a view to improving the learning experience. The fact that so much day-to-day course activity is taking place on an instrumented platform allows unprecedented opportunities in learning analytics and personalized content delivery.

An introduction to the course and the platform is provided to the students at the beginning of the semester when the course commences. Students are informed of the possibility to remove the data associated with their activity at the end of the semester (an opt-out policy). Then, after a 3 month waiting period, all data is anonymized prior

to analysis by our classifier. No students opted out in the year of analysis and data from 111 students was available.

We observed more than 9 million interaction events (including low-level data such as mouse tracking data) from the students through the semester on the platform. Figure 2 shows those transactions for all students, where each line represents a single student's activity through the module. The x-axis is divided into periods that represent the lab classes and the time between labs (1 week). This digital footprint, based on the students' extensive interaction with the platform, is an excellent source of information about the learning process.

The activity diagram shows three spikes that correspond with the periods before the lab exams and the final written exam. The two lab exams are activity intensive as well, even though these lab sessions are just two hours in duration. Each lab session was typically composed of a set of programming tasks that were to be completed within the timeframe of the lab. Apart from the two lab exams, labs were not graded and attendance was recorded as students were expected to attend. Attendance was typically in excess of 90%. The most interesting aspect of the diagram is the degree of activity immediately preceding the two lab exams and the final written exam. This shows the extent to which students on the module are assessment-driven.

Concept adoption

Apart from compressing successfully compiled programs to estimate the complexity of the code that students are writing, the availability of the code offers other opportunities. For example, it is possible to identify certain key concepts as core material for the selected module. One can then scan the programs the students have successfully written to test for the use of these concepts.

For this study we identified six different concepts as shown in Table 1. These represented a selection of key concepts drawn from the module curriculum. For each of these, we developed an automatic test that analyses the programs the students write

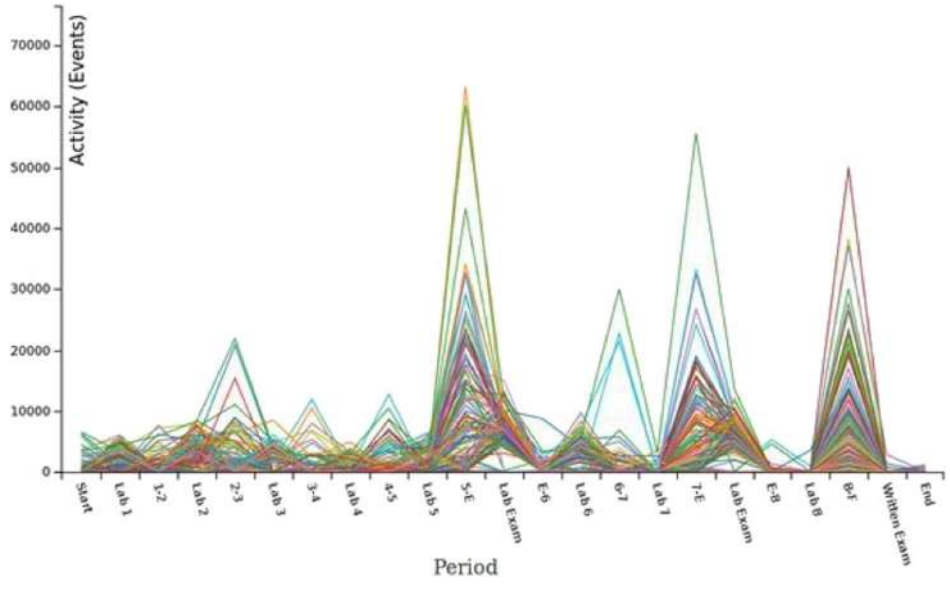

Fig. 2 Overall activity on the platform for a single module

Table 1 The concepts tested

Concept	Brief description
ISR	Interrupt Service Routine
Procedure	Procedure (including call and return)
Timer	Timer interrupt (specific form of ISR)
Conditional	Conditional Jump
Indirect	Indirect Addressing (MOVs of a specific type)
SevenSeg	Seven Segment display (output device)

and identifies if the concept is present. Some concepts are easier to test for than others. For example, Indirect Addressing is present if a program contains a single instruction of a specific form. Other concepts are slightly more involved to test for, such as Interrupt Service Routines (ISRs). These require checking several items in a program. (Did the student set up an Interrupt Vector Table? Did they setup an Interrupt Service routine at the appropriate point in memory? Did they return from that ISR? Did they enable interrupts in the first place?) Nonetheless, it is possible to write automated tests for each of these concepts and to efficiently scan the 58,785 programs the students successfully wrote.

With these automated tests, it was possible to determine the time in the semester at which a student first successfully compiled a program containing a specific concept. This gave an insight into student behavior. Figures 3 and 4 show the adoption times of two basic concepts–Indirect addressing and Conditionals. For each of the figures, we normalize the time during the semester at which we first detect that a student has used a concept. On the y-axis, zero represents the beginning of the semester and 1.0 represents the end of the semester. Students who never employ a concept are placed at 1.0 on the y-axis. Finally, students are sorted on the x-axis according to how early they first used the relevant concept.

The main observation from Figs. 3 and 4 is the similar patterns that they both exhibit. Most students (approximately 63%) adopt the relevant concept quite early in the semester, usually about 10% of the way through the semester. Another feature that is noteworthy is the stepped nature of the adoption times. These typically coincide with specific events such as a lab session where the students are given worksheets or a lab exam in order to focus their studies.

Figures 5 and 6 show the adoption times for slightly more complicated concepts, namely the Seven Segment Display and Procedures. Some differences are evident from the previous adoptions times. Most notable is that, overall, adoption times for these concepts are later than the previous concepts. This result is to be expected. Conditionals and Indirect Addressing are fundamental concepts taught early in the semester, while the Seven Segment Display and Procedures are taught somewhat later. The other difference is that for Procedures and the Seven Segment Display, there are a considerable number of students who never use the concept (9 and 20% respectively). This was a surprise to the lecturer who delivered the module, particularly in the case of the Seven Segment Display as significant effort was expended into development and into teaching students how to use this.

What is significant about Figs. 5 and 6 is that they demonstrate that if a student has not used a concept by a certain point in the semester (when just over 60% of the semester has

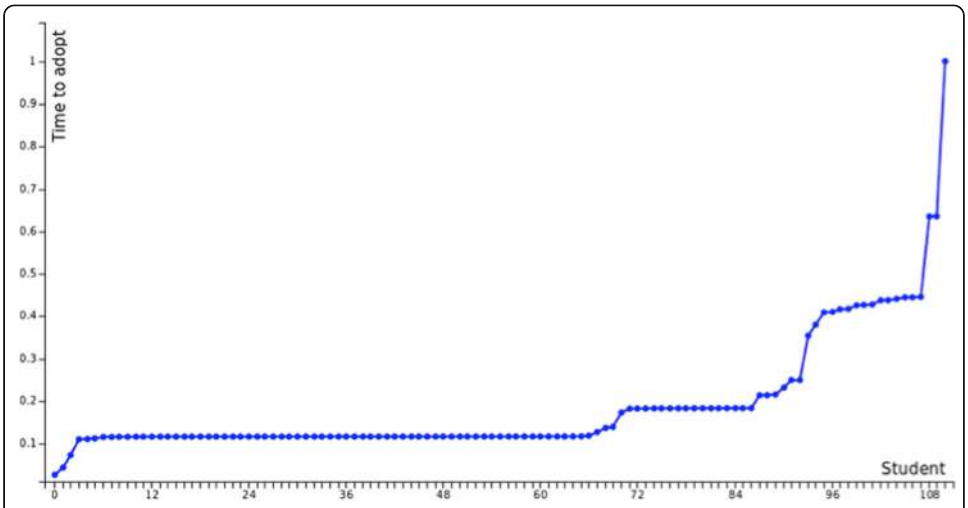

Fig. 3 Indirect Addressing adoption times

elapsed), then it is unlikely that they ever will. An appropriate intervention at this point would be compulsory, targeted tutorials and worksheets. Figures 5 and 6 also show the same stepped nature we have observed in Figs. 3 and 4. Again this is due to certain events such as labs or lab exams triggering the students to use a certain concept.

Figure 7 shows the concept adoption times for Interrupt Service Routines (ISRs). Apart from eight students who never use the concept, most students use ISRs for the first time at two narrow time periods. These coincide with lab sessions dedicated to this concept. Similar to before, if a student has not used a concept by approximately 60% through the semester, then it is unlikely they ever will, without intervention. The concept times for the Timer Interrupt, a specific version of the ISR are virtually identical and are omitted for brevity.

Overall, identifying specific concepts and examining when students use them is a worthwhile process as it can yield some results that are surprising to the educator (in this case the number of students who never use the Seven Segment Display). It also

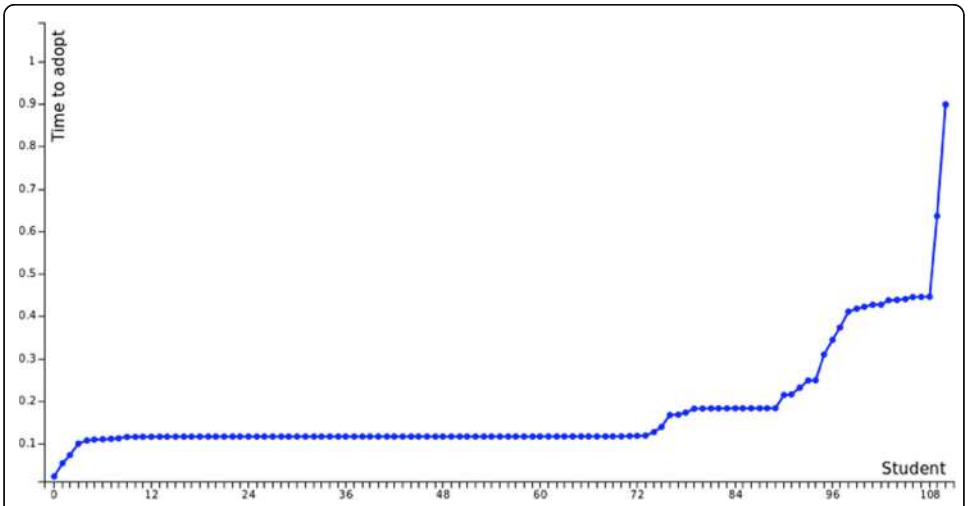

Fig. 4 Conditional adoption times

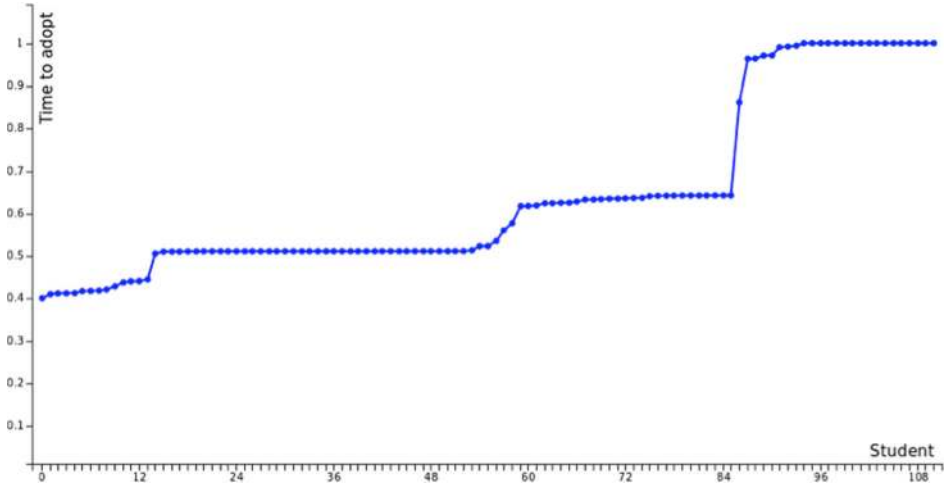

Fig. 5 Seven Segment adoption times

allows us to identify points in time during the course where targeted intervention would be advisable.

In our analysis, we found that students adopt different concepts at different times. Concept adoption times are quite varied and it is too simplistic to classify students as *early adopters* or *late adopters*. This strengthens the value of such fine-grained analysis and subsequent targeted intervention. Instead of just trying to identify students who are having difficulty in general, we can identify the particular concepts certain students are struggling with.

In the next section we discuss our pass-fail classifier, which uses a more significant portion of the digital footprint of students during the semester (not just the programs that they compile) and predicts whether they will pass or fail the final written exam based on their online activity. We show how the use of the concept-adoption times above can improve the accuracy of the classifier.

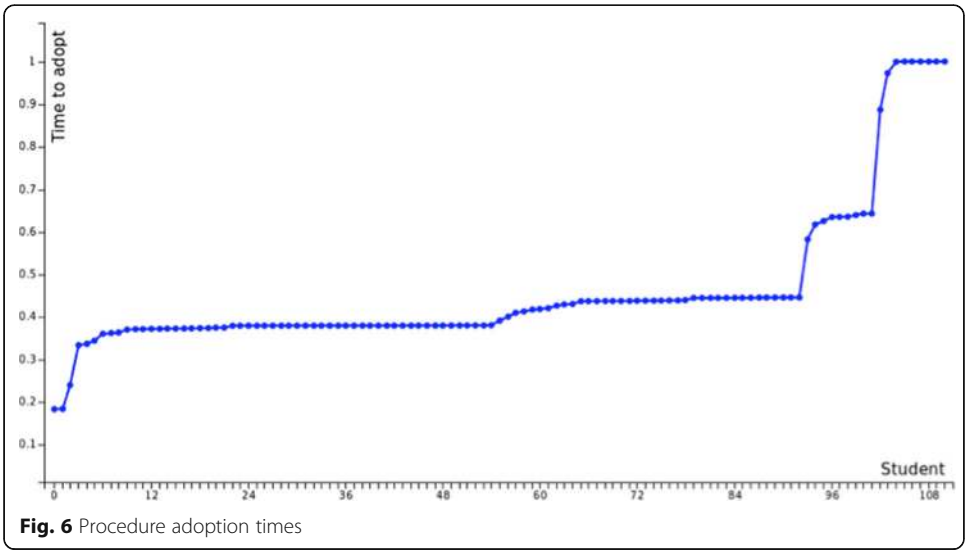

Fig. 6 Procedure adoption times

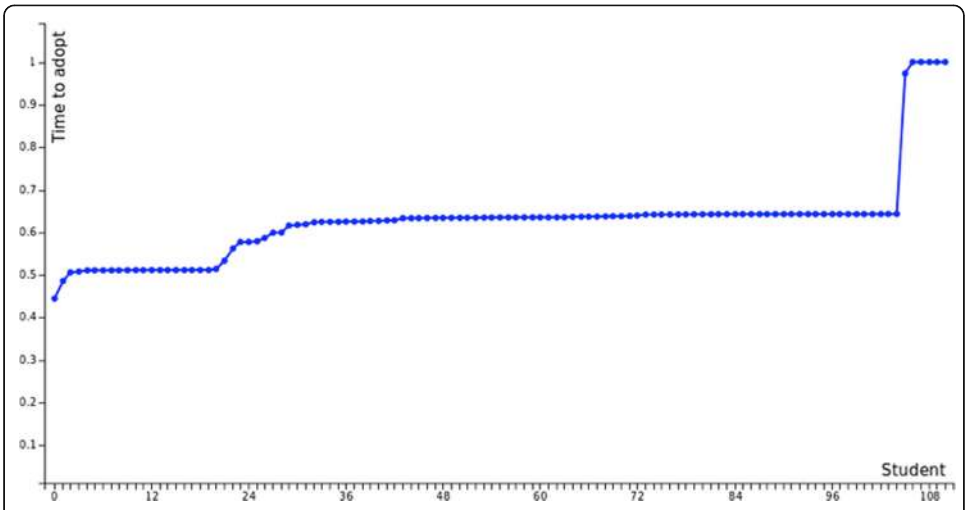

Fig. 7 ISR adoption times

Pass-fail classifier

We have developed a binary classifier (Györfi, Devroye, & Lugosi, 1996) that predicts a student's performance in the course's final written exam (pass/fail) based on a number of dimensions. This final exam is a 3 h long exam with a mixture of short questions (weighted at 20%) and 4 longer questions (weighted at 80%). The questions typically test a student's knowledge of assembly language programming with an emphasis on low level programming ability rather than problem solving skills.

To construct the pass-fail classifier, a bag of classifiers approach was adopted. This bag of classifiers is composed of a Linear Regression classifier, a Logistic Regression classifier, Gaussian Naive Bayes classifier, Multinomial Naive Bayes classifier, Bernoulli Naive Bayes classifier, Support Vector Machine with Radial Basis function kernel classifier, a K-Neighbors classifier with $K = 12$ and a Decision Tree classifier. To compare and evaluate different pre-processing techniques and models, we employed a cross-validation approach. In our case, we used a variant called "k-fold cross-validation" (Refaeilzadeh, Tang, & Liu, 2009) in order to compare the classifiers in the set. As the decision tree classifier in Scikit-learn is the best performing one in later sections, it is worth noting that the Scikit implementation is an optimized version of CART (Classification and Regression Trees) (Breiman, Friedman, Olshen, & Stone, 1984), which is quite similar to C4.5 (Quinlan, 1996).

The dimensions used for the classifier have been derived by processing a number of aspects of the digital footprint of students' interaction with the system over an entire semester. The output of the classifier is a prediction of whether a student will pass or fail the final written exam. These simple dimensions are based on data such as the number of successful and failed compilations, on-campus vs. off-campus connections, time spent on the platform, material covered, etc. some of which are presented in Table 2. It is worth noting that these measurements are based exclusively on browser interaction and consequently some measures such as time spent particularly on slides need to be treated with a degree of suspicion. For example, what happens if a student goes for a cup of coffee in the middle of browsing slides? Despite these reservations, we have found these dimensions contribute to the accuracy of the classifier.

Table 2 Metrics used for the classifier

• Number of Successful Compilations
• Successful Compilations Average Complexity
• Number of Failed Compilations
• Failed Compilations Average Complexity
• Ratio between On-campus and Off-campus Connections
• Number of Connections
• Time spent on the Platform
• Time spent on slides within the Platform
• Slides Coverage
• Number of Slides visited
• Number of Slides opened
• Number of transactions during Labs
• Number of transactions outside Labs
• Number of transactions in the Platform

One particularly successful feature that we have used as an input to our classifier is program complexity. In order to derive this, we strip off the comments of the compiled programs, compress them using a standard compression algorithm and measure the compressed code's length. This technique, examined in detail by Jbara and Feitelson (2014) is a useful proxy for code complexity.

As often is the case with real world data, the data collected was somewhat problematic, with missing, duplicated or corrupted entries. Thus, considerable efforts were spent on data cleansing (Hernández & Stolfo, 1998). The preprocessed data listed in Table 2 contain attributes with a mixture of scales for different quantities. The machine learning methods we apply either expect or are more effective if the data attributes all have the same scale. The two scaling methods we applied on our data were normalization and standardization.

In addition, we reduced the number of features or dimensions in order to verify whether feature reduction improves our prediction accuracy. Feature engineering, the judicious selection and pre-processing of such features, is one of the most challenging and important phases for such data-driven algorithms (Anderson et al., 2013).

The number of potential classifiers that could be used in an analysis can be somewhat daunting. This can be overcome by evaluating all possible classifiers and choosing the classifier that gives the greatest accuracy. The selection of classifiers evaluated was provided by the Python-based machine learning library, Scikit-learn (Pedregosa et al., 2011). To test the performance of each classifier, we employed a k-fold cross validation approach (Refaeilzadeh et al., 2009). Essentially this involves partitioning the data in multiple ways each of which avoids training the classifier on the same data that is used to evaluate its performance.

The Receiver Operating Characteristic (ROC), or ROC curve shown in Fig. 8 is a useful method to evaluate the performance of binary classifiers such as our pass-fail classifier. On the curve, the 50% (0.5) point represents performance no better than a coin flip. The 100% point (1.0) represents perfect accuracy in classification, which in our case is predicting perfectly which students will pass and which will fail. By the end of the semester the classifier is able to predict fail-pass results with just under an 85% accuracy. This is typically classified as good accuracy using the traditional academic points system (Tazhibi et al., 2011).

From the ROC curve in Fig. 8, it is notable how the classifier performance improves over time. This is due to the fact that the accumulated data upon which the classifier makes its decisions increases as the semester progresses. The lower accuracy earlier in the semester is exacerbated by the fact that students exhibit a strong back-loading behavior in this subject,

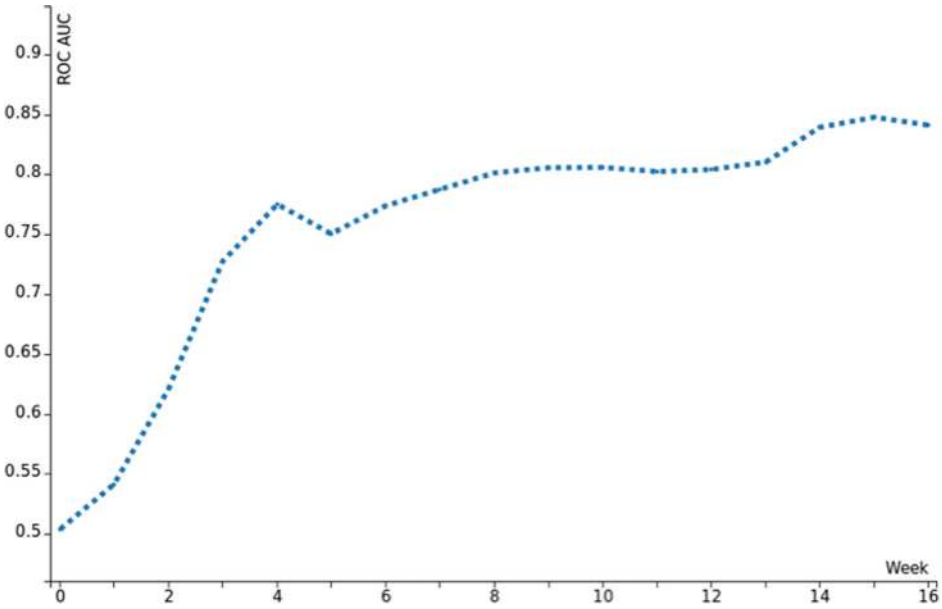

Fig. 8 Prediction accuracy on a weekly basis

leaving a significant portion of their learning activities until late in the semester. This can be seen from the activity levels depicted in Fig. 2. In short, accuracy is low earlier in the semester simply because students have not interacted with the system enough.

As a preliminary step to incorporating the concept adoption times into our classifier, we first examined if there was a correlation between the times and students' written exam grades. In all cases the Spearman correlation (shown in Table 3) indicated a weak to moderate correlation between how early students adopted a concept to how well they performed in the exam.

We then incorporated the concept adoption times into our pass-fail classifier presented above and the results are presented in Fig. 9. As can be seen from the diagram, the concept adoption times improve the accuracy of the classifier quite effectively from week 10 onwards.

In order to explain why this is only the case from week 10 onwards, in Fig. 10 we show the cumulative number of programs written and the total number of concepts detected. At six possible concepts per student and with 111 students, this gives a maximum of 666 concepts detected by the end of the module. In practice, this total is not reached because some students never adopt some concepts. In this figure, we can see that the data for concept adoption times is more or less complete by week 10. At that

Table 3 Metrics used for the classifier

Concept	Correlation	*P*-value
ISR	0.288	2.20E-003
Procedure	0.453	5.90E-007
Timer	0.265	4.87E-003
Conditional	0.373	5.46E-005
Indirect	0.386	2.93E-005
SevenSeg	0.352	1.53E-004

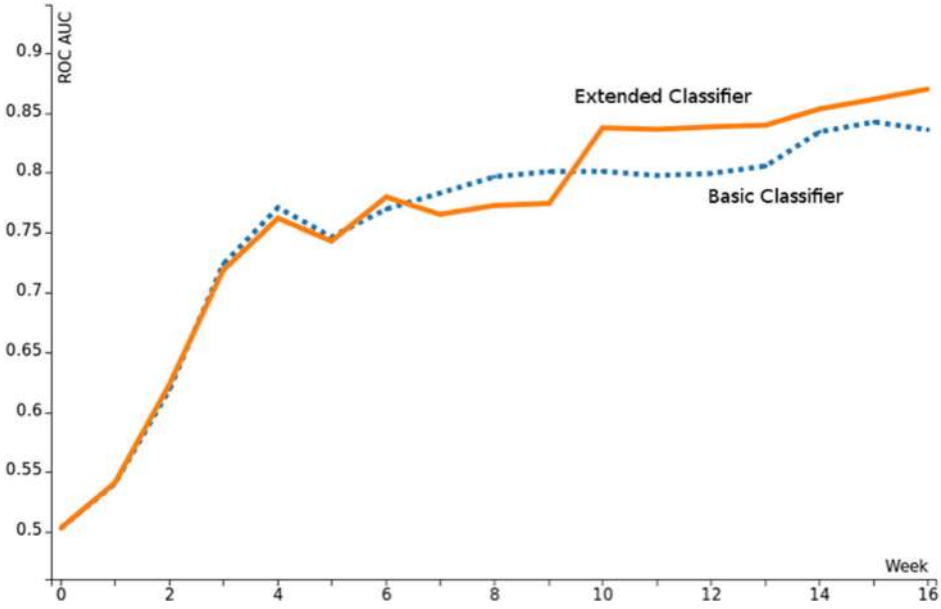

Fig. 9 Improved prediction accuracy

point, most students have adopted a concept (if they are ever going to). Before that point, there simply isn't enough data to improve the classifier.

Conclusion

In this paper we have shown the value of collecting the work that students perform during a module, and the value of providing tools to educators to analyze this work. In the particular case study presented, the analysis uncovered the fact that a large number of students were not using a key concept which the lecturer had spent considerable

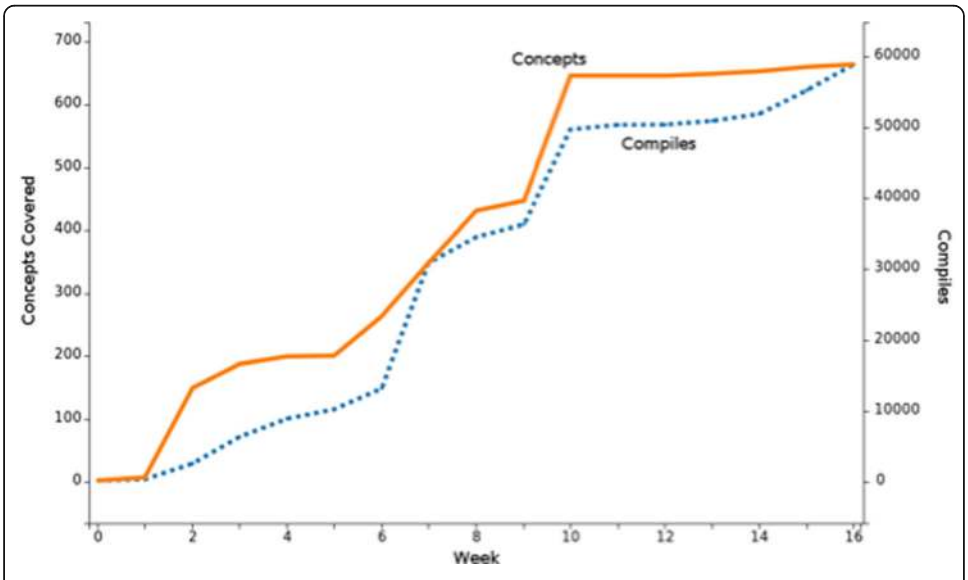

Fig. 10 Overall compilation and concept adoption behavior

effort in teaching. These results are interesting in that they highlight topics students find more difficult or uninteresting.

Early intervention is invaluable to identify at-risk students before they fail a module. Pass-fail classifiers can be useful in identifying these at-risk students, but only if they are accurate. We have been able to improve the accuracy of our basic pass-fail classifier by adding the concept adoption times for the six chosen concepts as dimensions to the classifier. The improvement in the classifier is not apparent until week 10 of the 16-week module. This is due to a degree of back-loading of student effort where they do not spread their effort out evenly during the semester. However, once enough data becomes available on student compilations (around week 10) the accuracy of the classifier improves significantly. At this point in time, 6 weeks remain before the final written exam, leaving some scope for intervention.

In future work, we will use other methods of analysis relating to how students adopt concepts. At present, we focus solely on the time a student first successfully uses a concept. This usage could be via an example program obtained from course materials or by the student writing their own code. Currently we do not distinguish between the two on the basis that learning is likely to occur in both scenarios. The adoption-time metric also does not distinguish between a student who has used a concept several times versus one who has only used the concept once. We also do not currently track when a student has attempted to use a concept unsuccessfully. The task of identifying an incorrect attempt to use a concept is one of measuring student intent, something that is simply not always possible solely by examining the programs they write.

The contribution to the pass-fail classifier can also be improved further. Currently, there is a positive contribution to classifier accuracy once most of the first-concept usage times are known (around week 10). Immediately before this, the classifier is made slightly worse by the use of the concept adoption times. It will be worth exploring mechanisms to avoid this performance cost. The classifier itself can be improved, for example by employing an ensemble approach (Galar, Fernández, Barrenechea, Bustince, & Herrera, 2011).

Beyond its use in the pass-fail classifier, the availability of the programs students write is an invaluable resource that lends itself to many forms of analysis. One of these forms of analysis is identifying key concepts and searching for when students first use these concepts. Through this, we can gain insights into how students relate to individual topics and identify optimal points in time for early, targeted, intervention. In general, this level of data permits better understanding as to how students learn by enabling an instructor to analyze exactly what students have been working on and when, all at an extremely high level of detail.

While the work on this paper concentrates on the challenging problem of how to predict student performance in the discipline of computer programming, the results are relevant wherever students expend a significant portion of their learning effort online.

Acknowledgements
None.

Funding
No funding agency was involved in this project.

Authors' contributions
Dr Casey was responsible for approximately 75% of the experimental work and paper authoring. Mr Azcona was responsible for the remaining 25%. Both authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Maynooth University, Maynooth, Ireland. [2]Dublin City University, Dublin, Ireland.

Received: 31 July 2016 Accepted: 12 December 2016
Published online: 13 February 2017

References

Ahadi, A., Lister, R., Haapala, H., & Vihavainen, A. (2015). Exploring machine learning methods to automatically identify students in need of assistance. In *Proceedings of the 11th annual International Conference on International Computing Education Research* (pp. 121–130). New York: ACM.

Anderson, M. R., Antenucci, D., Bittorf, V., Burgess, M., Cafarella, M. J., Kumar, A., Niu, F., Park, Y., Ré, C., & Zhang, C. (2013). Brainwash: A Data System for Feature Engineering. In *The 6th Biennial Conference on Innovative Data Systems Research*.

Arnold, K. E., & Pistilli, M. D. (2012). Course signals at Purdue: using learning analytics to increase student success. In *The 2nd international conference on learning analytics and knowledge* (pp. 267–270). New York: ACM.

Baker, R. S., Gowda, S. M., & Corbett, A. T. (2011). Towards predicting future transfer of learning. In *Proceedings of the 15th International Conference on Artificial Intelligence in Education* (pp. 23–30). Heidelberg: Springer Berlin.

Baker, R. S., Gowda, S., & Corbett, A. T. (2010). Automatically detecting a student's preparation for future learning: Help use is key. In *Proceedings of the 4th International Conference on Educational Data Mining*. Massachusetts: The International Educational Data Mining Society.

Baur, N. (2006). Microprocessor Simulator for Students. http://www.softwareforeducation.com/sms32v50/index.php. Accessed 17 Nov 2016.

Beaubouef, T., & Mason, J. (2005). Why the high attrition rate for computer science students: some thoughts and observations. *SIGCSE Bulletin, 37*(2), 103–106. doi:10.1145/1083431.1083474.

Berland, M., Martin, T., Benton, T., Petrick Smith, C., & Davis, D. (2013). Using learning analytics to understand the learning pathways of novice programmers. *Journal of the Learning Sciences, 22*(4), 564–599.

Biggers, M., Brauer, A., & Yilmaz, T. (2008). Student perceptions of computer science: a retention study comparing graduating seniors with CS leavers. *SIGCSE Bulletin, 40*(1), 402–406. doi:10.1145/1352135.1352274.

Blikstein, P. (2011). Using learning analytics to assess students' behavior in open-ended programming tasks. In *Proceedings of the 1st international conference on learning analytics and knowledge* (pp. 110–116). New York: ACM.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees (CART)*. Belmont: Wadsworth International Group.

Brown, N. C. C., Kölling, M., McCall, D., & Utting, I. (2014). *Blackbox: a large scale repository of novice programmers' activity*. In *Proceedings of the 45th ACM technical symposium on Computer science education* (pp. 223–228). New York: ACM.

Casey, K., & Gibson, P. (2010). Mining moodle to understand student behaviour. In *International Conference on Engaging Pedagogy*.

Davis, J. (2011). CompTIA: 400K IT Jobs Unfilled, Channel Insider. http://tinyurl.com/ca699dr. Accessed 20 Jul 2016.

Dehnadi, S., & Bornat, R. (2006). The camel has two humps. Middlesex University, UK, 1–21. http://tinyurl.com/yhxlkfb. Accessed 20 July 2016.

Edwards, S. (2013) Continuous Data-driven Learning Assessment. In Future Directions in Computing Education Summit White Papers (SC1186). Dept. of Special Collections and University Archives, Stanford University Libraries, Stanford, Calif. http://tinyurl.com/jep5vgt. Accessed 17 Nov 2016.

Ferreira, D. (2013). *Instant HTML5 Presentations How-to*. Birmingham: Packt Publishing Ltd.

Galar, M., Fernández, A., Barrenechea, E., Bustince, H., & Herrera, F. (2011). An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes. *Pattern Recognition, 44*(8), 1761–1776.

Györfi, L., Devroye, L., & Lugosi, G. (1996). *A probabilistic theory of pattern recognition*. New York: Springer.

Hernández, M. A., & Stolfo, S. J. (1998). Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery, 2*(1), 9–37.

Huntley, H., & Young, A. (2015). *Service Providers Are Waging War Against U.S. Talent Shortage With Unconventional Methods*. Gartner Corporation Special Report (G00271791). 21st April, 2015. Stamford: Gartner Inc.

Jbara, A., & Feitelson, D. G. (2014). Quantification of code regularity using preprocessing and compression. *manuscript, Jan, 132*. http://www.cs.huji.ac.il/~feit/papers/RegMet14.pdf.

Kelly, D., & Thorn, K. (2013). Should Instructional Designers care about the Tin Can API? *eLearn, 2013*, 3. doi:10.1145/2446514.2446579.

Lang, C., McKay, J., & Lewis, S. (2007). Seven factors that influence ICT student achievement. *SIGCSE Bulletin, 39*(3), 221–225. doi:10.1145/1269900.1268849.

Lapowsky, I. (2015). Obama Has a $100M Plan to Fill the Tech Talent Shortage. Wired Magazine. https://www.wired.com/2015/03/techhire-initiative/. Accessed 3 Nov 2016.

Lister, R. (2008). After the gold rush: toward sustainable scholarship in computing. In *Proceedings of the tenth conference on Australasian computing education-Volume 78* (pp. 3–17). Darlinghurst: Australian Computer Society, Inc.

Macfadyen, L. P., & Dawson, S. (2010). Mining LMS data to develop an "early warning system" for educators: A proof of concept. *Computers & Education, 54*(2), 588–599.

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., & Byers, A. H. (2011). *Big data: The next frontier for innovation, competition, and productivity*. Washington: McKinsey Global Institute.

O'Kelly, J., Bergin, S., Dunne, S., Gaughran, P., Ghent, J., & Mooney, A. (2004). Initial findings on the impact of an alternative approach to Problem Based Learning in Computer Science. In *Pleasure by Learning*.

O'Kelly, J., Mooney, A., Bergin, S., Gaughran, P., & Ghent, J. (2004). An overview of the integration of problem based learning into an existing computer science programming module. In *Pleasure by Learning*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*(Oct), 2825–2830.

Quinlan, J. R. (1996). Bagging, boosting, and C4. 5. In *AAAI/IAAI, Vol. 1* (pp. 725–730).

Refaeilzadeh, P., Tang, L., & Liu, H. (2009). Cross Validation, In *Encyclopedia of Database Systems* (pp. 532–538). New York: Springer.

Romero-Zaldivar, V. A., Pardo, A., Burgos, D., & Kloos, C. D. (2012). Monitoring student progress using virtual appliances: A case study. *Computers & Education, 58*(4), 1058–1067.

Siemens, G., & Long, P. (2011). Penetrating the Fog: Analytics in Learning and Education. *EDUCAUSE review, 46*(5), 30.

Slonim, J., Scully, S., & McAllister, M. (2008). Crossroads for Canadian CS enrollment. *Communications of the ACM, 51*(10), 66–70.

Tazhibi M., Bashardoost N., Ahmadi M. (2011). Kernel smoothing for ROC curve and estimation for thyroid stimulating hormone. *International Journal of Public Health Research, Special Issue.* (pp. 239–242).

Teague, D., & Roe, P. (2007). Learning to program: Going pair-shaped. *Innovation in Teaching and Learning in Information and Computer Sciences, 6*(4), 4–22.

Thibodeau, P. (2011). Romney Sees Tech Skills Shortage, More H-1B Visas Needed Computer World, http://tinyurl.com/76l4qxo. Accessed 20 Jul 2016.