This is the accepted version of:
Van Petegem, C., Deconinck, L., Mourisse, D., Maertens, R., Strijbol, N., Dhoedt, B., De Wever, B., Dawyndt, P., & Mesuere, B. (2023). Pass/Fail Prediction in Programming Courses. Journal of Educational Computing Research, 61(1), 68–95.
https://doi.org/10.1177/07356331221085595

Pass/fail prediction in programming courses

Charlotte Van Petegem, Louise Deconinck, Dieter Mourisse,
Rien Maertens, Niko Strijbol, Bart Dhoedt,
Bram De Wever, Peter Dawyndt & Bart Mesuere

2022-06-07

Abstract

We present a privacy-friendly early-detection framework to identify students at risk of failing in introductory programming courses at university. The framework was validated for two different courses with annual editions taken by higher education students (N=2 080) and was found to be highly accurate and robust against variation in course structures, teaching and learning styles, programming exercises, and classification algorithms. By using interpretable machine learning techniques, the framework also provides insight into what aspects of practising programming skills promote or inhibit learning or have no or minor effect on the learning process. Findings showed that the framework was capable of predicting students' future success already early on in the semester.

1 Introduction

A lot of educational opportunities are missed by keeping assessment separate from learning (Black & Wiliam, 1998; Wiliam, 2011). Educational technology can bridge this divide by providing real-time data and feedback to help students learn better, teachers teach better, and education systems become more effective (OECD, 2021). Earlier research demonstrated that the adoption of interactive platforms may lead to better learning outcomes (Khalifa & Lam, 2002) and allows to collect rich data on student behaviour throughout the learning process in non-evasive ways. Effectively using such data to extract knowledge and further improve the underlying processes, which is called educational data mining (Baker & Yacef, 2009), is increasingly explored as a way to enhance learning and educational processes (Dutt et al., 2017).

About one third of the students enrolled in introductory programming courses fail (Bennedsen & Caspersen, 2007; Watson & Li, 2014). Such high failure rates are problematic in light of low enrolment numbers and high industrial demand for software engineering and data science profiles (Watson & Li, 2014). To remedy this situation, it is important to have detection systems for monitoring at-risk students, understand why they are failing, and develop preventive strategies. Ideally, detection happens early on in the learning process to leave room for timely feedback and interventions that can help students increase their chances of passing a course.

Previous approaches for predicting performance on examinations either take into account prior knowledge such as educational history and socio-economic background of students or require extensive tracking of student behaviour. Extensive behaviour tracking may directly impact the learning process itself. Rountree et al. (2004) used decision trees to find that the chance of failure strongly correlates with a combination of academic background, mathematical background, age, year of study, and expectation of a grade other than "A". They conclude that students with a skewed view on workload and content are more likely to fail. Kovacic (2012) used data mining techniques and logistic regression on enrolment data to conclude that ethnicity and curriculum are the most important factors for predicting student success. They were able to predict success with 60% accuracy. Livieris et al. (2019) use semi-supervised machine learning to predict student outcomes with 80% accuracy using prior academic history. Xing & Du (2019) achieve accuracies

up to 95% when using deep learning to predict performance in MOOCs, though their dataset was very imbalanced. Asif et al. (2017) combine examination results from the last two years in high school and the first two years in higher education to predict student performance in the remaining two years of their academic study program. They used data from one cohort to train models and from another cohort to test that the accuracy of their predictions is about 80%. This evaluates their models in a similar scenario in which they could be applied in practice.

A downside of the previous studies is that collecting uniform and complete data on student enrolment, educational history and socio-economic background is impractical for use in educational practice. Data collection is time-consuming and the data itself can be considered privacy sensitive. Usability of predictive models therefore not only depends on their accuracy, but also on their dependency on findable, accessible, interoperable and reusable data (Wilkinson et al., 2016). Predictions based on educational history and socio-economic background also raise ethical concerns. Such background information definitely does not explain everything and lowers the perceived fairness of predictions (Binns et al., 2018; Grgić-Hlača et al., 2018). A student can also not change their background, so these items are not actionable for any corrective intervention.

It might be more convenient and acceptable if predictive models are restricted to data collected on student behaviour during the learning process of a single course. An example of such an approach comes from Vihavainen (2013), using snapshots of source code written by students to capture their work attitude. Students are actively monitored while writing source code and a snapshot is taken automatically each time they edit a document. These snapshots undergo static and dynamic analysis to detect good practises and code smells, which are fed as features to a nonparametric Bayesian network classifier whose pass/fail predictions are 78% accurate by the end of the semester. In a follow-up study they applied the same data and classifier to accurately predict learning outcomes for the same student cohort in another course (Vihavainen et al., 2013). In this case, their predictions were 98.1% accurate, although sample size was rather small. While this procedure does not rely on external background information, it has the drawback that data collection is more invasive and directly intervenes with the learning process. Students can't work in their preferred programming environment and have to agree with extensive behaviour tracking.

Approaches that are not using machine learning also exist. Feldman et al. (2019) try to answer the question "Am I on the right track?" on the level of individual exercises, by checking if the student's current progress can be used as a base to synthesise a correct program. However, there is no clear way to transform this type of approach into an estimation of success on examinations. Honour (1986) found significant ($p < 0.05$) correlations between students' college grades, the number of hours worked, the number of high school mathematics classes and the students' grades for an introductory programming course. Goold & Rimmer (2000) also looked at learning style (surveyed using LSI2) as a factor in addition to demographics, academic ability, problem-solving ability and indicators of personal motivation. The regressions in their study account for 42 to 65 percent of the variation in cohort performances.

In this manuscript, we present an alternative framework (Figure 1) to predict if students will pass or fail a course within the same context of learning to code. The method only relies on submission behaviour for programming exercises to make accurate predictions and does not require any prior knowledge or intrusive behaviour tracking. Interpretability of the resulting models was an important design goal to enable further investigation on learning habits. We also focused on early detection of at-risk students, because predictive models are only effective for the cohort under investigation if remedial actions can be started long before students take their final exam.

The manuscript starts with a description of how data is collected, what metadata is used and which machine learning methods have been evaluated to make pass/fail predictions. We evaluated the same models and features in multiple courses to test their robustness against differences in teaching styles and student backgrounds. The results are discussed from a methodological and educational perspective with a focus on i) accuracy (What machine learning algorithms yield the best predictions?), ii) early detection (Can we already make accurate predictions early on in the semester?), and iii) interpretability (Are resulting models clear about which features are important? Can we explain why certain features are identified as important? How self-evident are important features?).

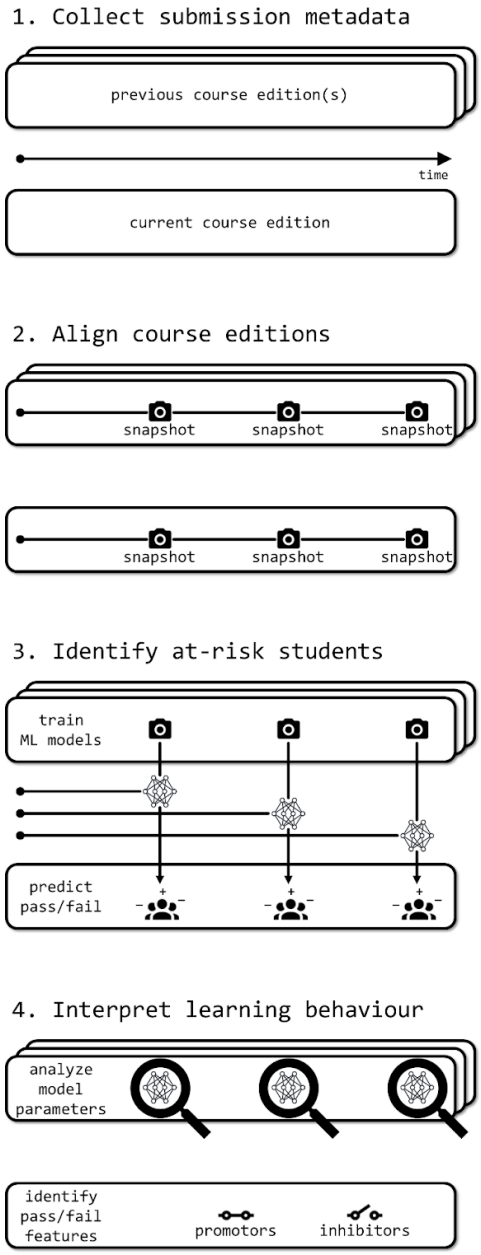

Figure 1: Step-by-step process of the proposed pass/fail prediction framework for programming courses: 1) Collect metadata from student submissions during successive course editions. 2) Align course editions by identifying corresponding time points and calculating snapshots at these time points. A snapshot measures student performance only from metadata available in the course edition at the time the snapshot was taken. 3) Train a machine learning model on snapshot data from previous course editions and predict which students will likely pass or fail the current course edition by applying the model on a snapshot of the current edition. 4) Infer what learning behaviour has a positive or negative learning effect by interpreting feature weights of the machine learning model. Teachers can use insights from both steps 3 and 4 to take actions in their teaching practice.

course	academic year	students	series	exercises	mandatory exercises	submitted solutions	attempts	pass rate
A	2016-2017	322	10	60	yes	167 675	9.56	60.86%
A	2017-2018	249	10	60	yes	125 920	9.19	61.44%
A	2018-2019	307	10	60	yes	176 535	10.29	65.14%
B	2016-2017	372	20	138	no	371 891	9.10	56.72%
B	2017-2018	393	20	187	no	407 696	7.31	60.81%
B	2018-2019	437	20	201	no	421 461	6.26	62.47%

Table 1: Statistics for course editions included in this study. The courses are taken by different student cohorts at different faculties and differ in structure, lecturers and teaching assistants. A series is a collection of exercises typically handled in one week/lab session. The number of attempts is the average number of solutions submitted by a student per exercise they worked on (i.e., for which the student submitted at least one solution in the course edition).

2 Materials and methods

2.1 Course structures

This study uses data from two introductory programming courses (referenced as course A and course B) collected during 3 editions of each course in academic years 2016-2017, 2017-2018 and 2018-2019. Both courses run once per academic year across a 12-week semester (September-December). They have separate lecturers and teaching assistants, and are taken by students of different faculties. The courses have their own structure, but each edition of a course follows the same structure. Table 1 summarises some statistics on the course editions included in this study.

Course A is subdivided into two successive instructional units that each cover five programming topics — one topic per week — followed by an evaluation about all topics covered in the unit. Students must solve six programming exercises on each topic before a deadline one week later. Submitted solutions for these mandatory exercises are automatically evaluated and considered correct if they pass all unit tests for the exercise. Failing to submit a correct solution for a mandatory exercise has a small impact on the score for the evaluation at the end of the unit. The final exam at the end of the semester evaluates all topics covered in the entire course. Students need to solve new programming exercises during evaluations (2 exercises) and exams (3 exercises), where the teaching staff manually evaluates and grades submitted solutions based on correctness, programming style used, choice made between the use of different programming techniques, and the overall quality of the solution. Each edition of the course is taken by about 300 students.

Course B has 20 lab sessions across the semester, with evaluations after the 10th and 17th lab session and a final exam at the end of the semester. Each lab session comes with a set of exercises and has an indicative deadline for submitting solutions. However, these exercises are not taken into account when computing the final score for the course, so students are completely free to work on exercises as a way to practice their coding skills. Students need to solve new programming exercises during evaluations (3 exercises) and exams (4 exercises). Solutions submitted during evaluations are automatically graded based on the number of passed unit tests for the exercise. Solutions submitted during exams are manually graded in the same way as for course A. Each edition of the course is taken by about 400 students.

We opted to use two different courses that are structured quite differently to make sure our framework is generally applicable in other courses where the same behavioural data can be collected.

2.2 Learning environment

Both courses use the same in-house online learning environment. This online learning environment promotes active learning through problem solving (Prince, 2004). Each course edition has its own

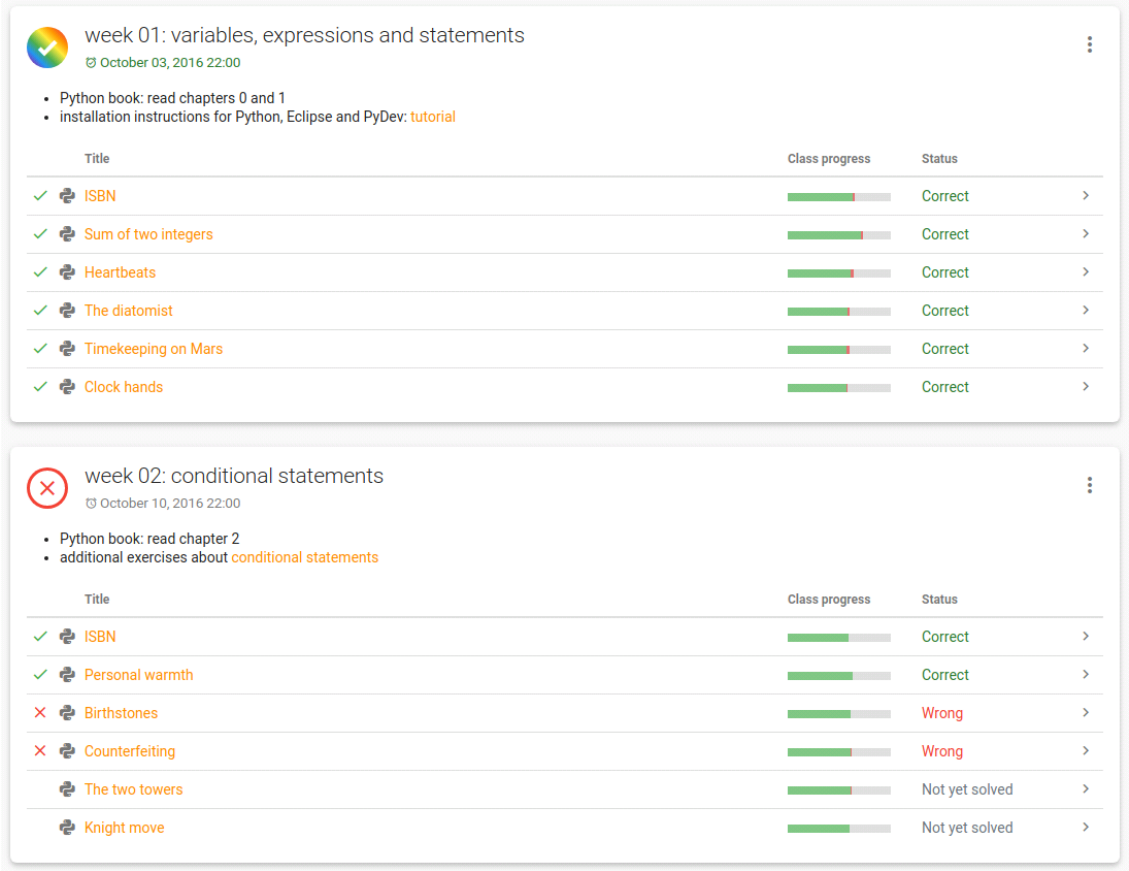

Figure 2: Student view of a module in the online learning environment from which we collected our data, showing two series of six exercises in the learning path of course A. Each series has its own deadline. The status column shows a global status for each exercise based on the last solution submitted. The class progress column visualises global status for each exercise for all students subscribed in the course. Icons on the left show a global status for each exercise based on the last submission submitted before the series deadline.

module, with a learning path that groups exercises in separate series (Figure 2). Course A has one series per covered programming topic (10 series in total) and course B has one series per lab session (20 series in total). A submission deadline is set for each series. The learning environment is also used to take tests and exams, within series that are only accessible for participating students.

Throughout a course edition, students can continuously submit solutions for programming exercises and immediately receive feedback upon each submission, even during tests and exams. This rich feedback is automatically generated by an online judge and unit tests linked to each exercise (Wasik et al., 2018). Guided by that feedback, students can track potential errors in their code, remedy them and submit an updated solution. A more detailed description of the process that students go through when solving programming exercises in the online learning environment can be found in supplementary material. There is no restriction on the number of solutions that can be submitted per exercise, and students can continue to submit solutions after a series deadline. All submitted solutions are stored, but only the last submission before the deadline is taken into account to determine the status (and grade) of an exercise for a student. One of the effects of active learning, triggered by exercises with deadlines and automated feedback, is that most learning happens during the semester as can be seen on the heatmap in Figure 3.

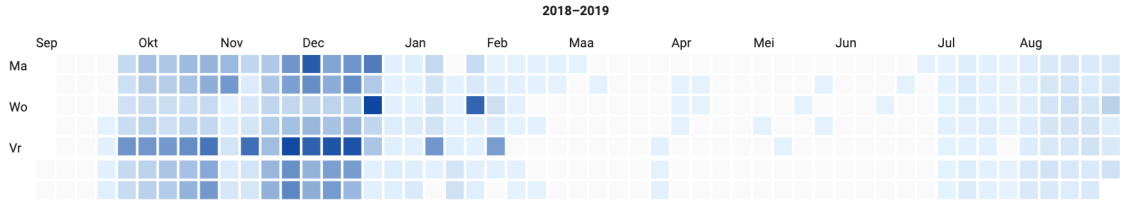

Figure 3: Heatmap showing the distribution per day of all 176 535 solutions submitted during the 2018-2019 edition of course A. The darker the colour, the more submissions were made on that day. A lighter blue means there are few submissions on that day. A light grey square means that no submissions were made that day. Weekly lab sessions for different groups on Monday afternoon, Friday morning and Friday afternoon, where we can see darker squares. Weekly deadlines for mandatory exercises were on Tuesdays at 22:00. There were four exam sessions for different groups in January. There is little activity in the exam periods, except for days where there was an exam. The course is not taught in the second semester, so there is very little activity there. Two exam sessions were organised in August and September granting an extra chance to students who failed on their exam in January.

2.3 Submission data

We exported data from the learning environment on all solutions submitted by students during each course edition included in the study. Each solution has a submission timestamp with precision down to the second and is linked to a course edition, series in the learning path, exercise and student. We did not use the actual source code submitted by students, but did use the status describing the global assessment made by the learning environment: correct, wrong, compilation error, runtime error, time limit exceeded, memory limit exceeded, or output limit exceeded.

Comparison of student behaviour between different editions of the same course is enabled by computing snapshots for each edition at series deadlines. Because course editions follow the same structure, we can align their series and compare snapshots for corresponding series. Corresponding snapshots represent student performance at intermediate points during the semester and their chronology also allows longitudinal analysis within the semester. Course A has snapshots for the five series of the first unit (labelled S1-S5), a snapshot for the evaluation of the first unit (labelled E1), snapshots for the five series of the second unit (labelled S6-S10), a snapshot for the evaluation of the second unit (labelled E2) and a snapshot for the exam (labelled E3). Course B has snapshots for the first ten lab sessions (labelled S1-S10), a snapshot for the first evaluation (labelled E1), snapshots for the next series of seven lab sessions (labelled S11-S17), a snapshot for the second evaluation (labelled E2), snapshots for the last three lab sessions (S18-S20) and a snapshot for the exam (labelled E3).

It is important to stress that a snapshot of a course edition measures student performance only using the information available at the time the snapshot was taken. As a result, the snapshot does not take into account submissions after its timestamp. The learning behaviour of a student is expressed as a set of features extracted from the raw submission data. We identified different types of features (see appendix A) that indirectly quantify certain behavioural aspects of students practicing their programming skills. When and how long do students work on their exercises? Can students correctly solve an exercise and how much feedback do they need to accomplish this? What kinds of mistakes do students make while solving programming exercises? Do students further optimise the quality of their solution after it passes all unit tests, based on automated feedback or publication of sample solutions? Note that there is no one-on-one relationship between these behavioural aspects and feature types. Some aspects will be covered by multiple feature types, and some feature types incorporate multiple behavioural aspects. We will therefore need to take into account possible dependencies between feature types while making predictions.

A feature type essentially makes one observation per student per series. Each feature type thus results in multiple features: one for each series in the course (excluding series for evaluations

and exams). In addition, the snapshot also contains a feature for the average of each feature type across all series. We do not use observations per individual exercise, as the actual exercises might differ between course editions. Snapshots taken at the deadline of an evaluation or later, also contain the score a student obtained for the evaluation. These features of the snapshot can be used to predict whether a student will finally pass/fail the course. In addition, the snapshot also contains a label indicating whether the student passed or failed that is used during training and testing of classification algorithms. Students that did not take part in the final examination, automatically fail the course.

Since course B has no hard deadlines, we left out deadline-related features from its snapshots (`first_dl`, `last_dl` and `nr_dl`; see appendix A). To investigate the impact of deadline-related features, we also made predictions for course A that ignore these features.

2.4 Classification algorithms

We evaluated four classification algorithms to make pass/fail predictions from student behaviour: stochastic gradient descent (Ferguson, 1982), logistic regression (Kleinbaum, 1994), support vector machines (Cortes & Vapnik, 1995), and random forests (Svetnik et al., 2003). We used implementations of these algorithms from scikit-learn (Pedregosa et al., 2011) and optimised model parameters for each algorithm by cross-validated grid-search over a parameter grid (`sklearn.model_selection.GridSearchCV`).

Readers unfamiliar with machine learning can think of these specific algorithms as black boxes, but we briefly explain the basic principles of classification for their understanding. Supervised learning algorithms use a dataset that contains both inputs and desired outputs to build a model that can be used to predict the output associated with new inputs. The dataset used to build the model is called the training set and consists of training examples, with each example represented as an array of input values (feature vector). Classification is a specific case of supervised learning where the outputs are restricted to a limited set of values (labels), in contrast to for example all possible numerical values with a range. Classification algorithms are validated by splitting a dataset of labelled feature vectors into a training set and a test set, building a model from the training set, and evaluating the accuracy of its predictions on the test set. Keeping training and test data separate is crucial to avoid bias during validation. A standard method to make unbiased predictions for all examples in a dataset is k-fold cross-validation: partition the dataset in k subsets and then perform k experiments that each take one subset for evaluation and the other k-1 subsets for training the model.

Pass/fail prediction is a binary classification problem with two possible outputs: passing or failing a course. We evaluated the accuracy of the predictions for each snapshot and each classification algorithm with three different types of training sets. As we have data from three editions of each course, the largest possible training set to make predictions for the snapshot of a course edition combines the corresponding snapshots from the two remaining course editions. We also made predictions for a snapshot using each of its corresponding snapshots as individual training sets to see if we can still make accurate predictions based on data from only one other course edition. Finally, we also made predictions for a snapshot using 5-fold cross-validation to compare the quality of predictions based on data from the same or another cohort of students. Note that the latter strategy is not applicable to make predictions in practice, because we will not have pass/fail results as training labels while taking snapshots during the semester. To make predictions for a snapshot, we can in practice rely only on corresponding snapshots from previous course editions. However, because we can assume that different editions of the same course yield independent data, we also used snapshots from future course editions in our experiments.

There are many metrics that can be used to evaluate how accurately a classifier predicted which students will pass or fail the course from the data in a given snapshot. Predicting a student will pass the course is called a positive prediction, and predicting they will fail the course is called a negative prediction. Predictions that correspond with the actual outcome are called true predictions, and predictions that differ from the actual outcome are called false predictions. This results in four possible combinations of predictions: true positives (TP), true negatives (TN), false positives

(FP) and false negatives (FN). Two standard accuracy metrics used in information retrieval are precision (TP/(TP+FP)) and recall (TP/(TP+FN)). The latter is also called sensitivity if used in combination with specificity (TN/(TN+FP)).

Many studies for pass/fail prediction use accuracy ((TP+TN)/(TP+TN+FP+FN)) as a single performance metric. However, this can yield misleading results. For example, let's take a dummy classifier that always "predicts" students will pass, no matter what. This is clearly a bad classifier, but it will nonetheless have an accuracy of 75% for a course where 75% of the students pass. In our study, we will therefore use two more complex metrics that take these effects into account: balanced accuracy and F_1-score. Balanced accuracy is the average of sensitivity and specificity. The F_1-score is the harmonic mean of precision and recall. If we go back to our example, the optimistic classifier that consistently predicts that all students will pass the course and thus fails to identify any failing student will have a balanced accuracy of 50% and an F_1-score of 75%. Under the same circumstances, a pessimistic classifier that consistently predicts that all students will fail the course has a balanced accuracy of 50% and an F_1-score of 0%.

2.5 Pass/fail predictions

In summary, Figure 1 outlines the entire flow of the proposed pass/fail prediction framework. It starts by extracting metadata for all submissions students made so far within a course (timestamp, status, student, exercise, series) and collecting their marks on intermediate tests and final exams (step 1). In practice, applying the framework on a student cohort in the current course edition only requires submission metadata and pass/fail outcomes from student cohorts in previous course editions. Successive course editions are then aligned by identifying fixed time points throughout the course where predictions are made, for example at submission deadlines, intermediate tests or final exams (step 2). We conducted a longitudinal study to evaluate the accuracy of pass/fail predictions at successive stages of a course (step 3). This is done by extracting features from the raw submission metadata of one or more course editions and training machine learning models that can identify at-risk students during other course editions. Our scripts that implement this framework are provided as supplementary material. Teachers can also interpret the behaviour of students in their class by analysing the feature weights of the machine learning models (step 4).

3 Results and discussion

We evaluated the performance of four classification algorithms for pass/fail predictions in a longitudinal sequence of snapshots from course A and B: stochastic gradient descent (Figure 4), logistic regression (Figure 5), support vector machines (Figure 6), and random forests (Figure 7). For each classifier, course and snapshot, we evaluated 12 predictions for the following combinations of training and test sets: train on one edition and test on another edition; train on two editions and test on the other edition; train and test on one edition using 5-fold cross validation. In addition, we made predictions for course A using both the full set of features and a reduced feature set that ignores deadline-related features. We discuss the results in terms of accuracy, potential for early detection, and interpretability.

3.1 Accuracy

The overall conclusion from the longitudinal analysis is that indirectly measuring how students practice their coding skills by solving programming exercises (formative assessments) in combination with directly measuring how they perform on intermediate evaluations (summative assessments), allows us to predict with high accuracy if students will pass or fail a programming course. The signals to make such predictions seem to be present in the data, as we come to the same conclusions irrespective of the course, classification algorithm, or performance metric evaluated in our study. Overall, logistic regression was the best performing classifier for both courses, but the difference compared to the other classifiers is small.

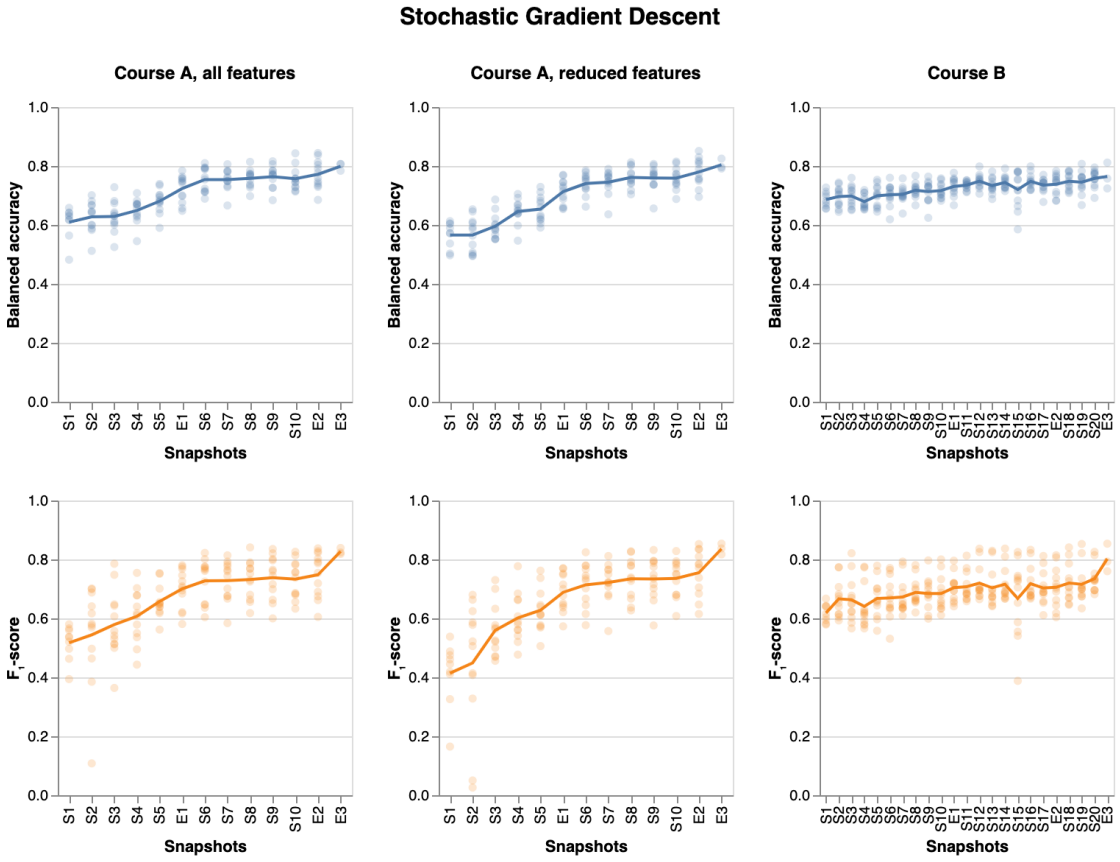

Figure 4: Performance of stochastic gradient descent classifiers for pass/fail predictions in a longitudinal sequence of snapshots from courses A (all features and reduced set of features) and B, measured by balanced accuracy and F_1-score. Dots represent performance of a single prediction, with 12 predictions for each group of corresponding snapshots (columns). Solid line connects averages of the performances for each group of corresponding snapshots.

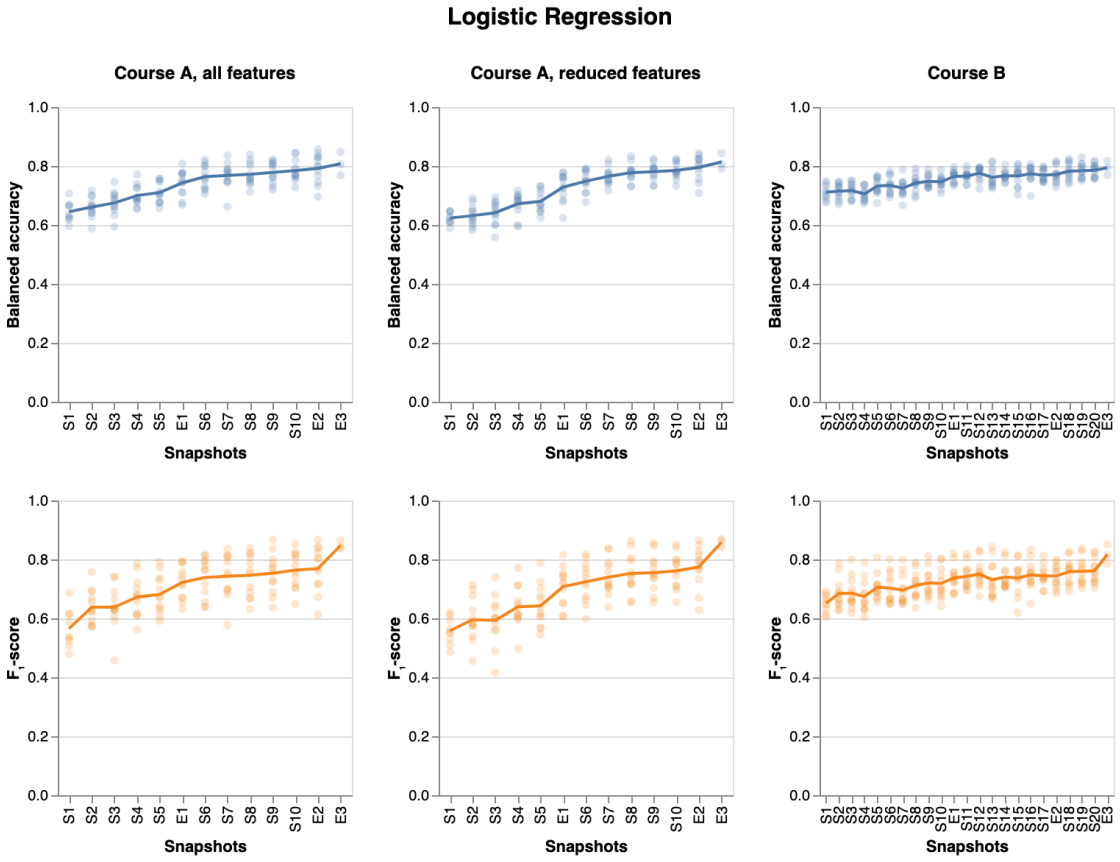

Figure 5: Performance of logistic regression classifiers for pass/fail predictions in a longitudinal sequence of snapshots from courses A (all features and reduced set of features) and B, measured by balanced accuracy and F_1-score. Dots represent performance of a single prediction, with 12 predictions for each group of corresponding snapshots (columns). Solid line connects averages of the performances for each group of corresponding snapshots.

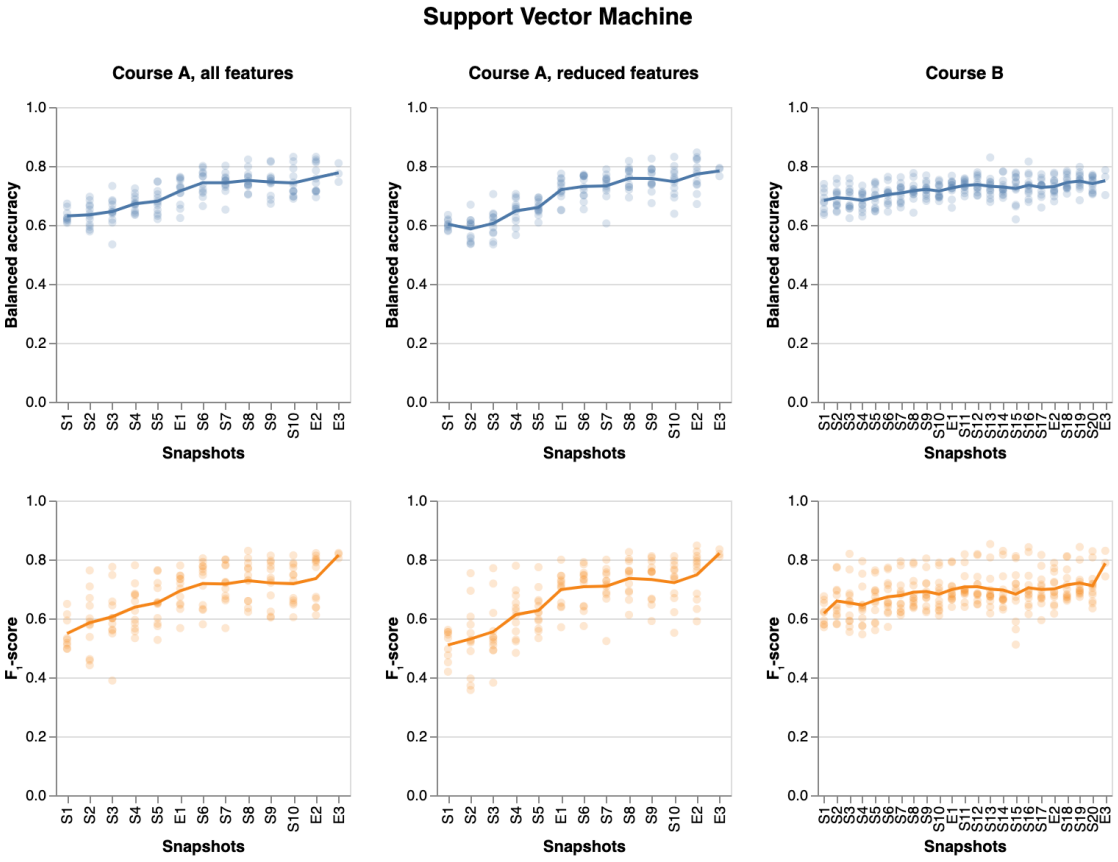

Figure 6: Performance of support vector machine classifiers for pass/fail predictions in a longitudinal sequence of snapshots from courses A (all features and reduced set of features) and B, measured by balanced accuracy and F_1-score. Dots represent performance of a single prediction, with 12 predictions for each group of corresponding snapshots (columns). Solid line connects averages of the performances for each group of corresponding snapshots.

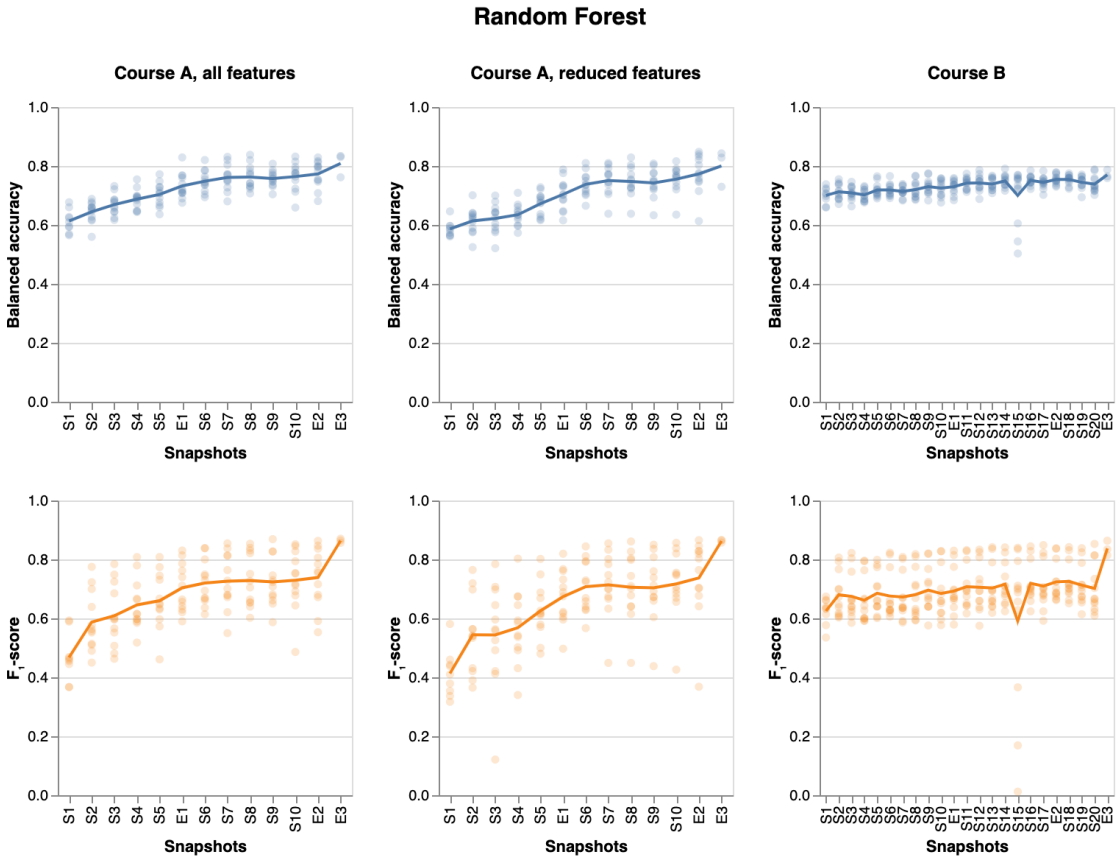

Figure 7: Performance of random forest classifiers for pass/fail predictions in a longitudinal sequence of snapshots from courses A (all features and reduced set of features) and B, measured by balanced accuracy and F_1-score. Dots represent performance of a single prediction, with 12 predictions for each group of corresponding snapshots (columns). Solid line connects averages of the performances for each group of corresponding snapshots.

When we compare the longitudinal trends of balanced accuracy for the predictions of both courses, we see that course A starts with a lower balanced accuracy at the first snapshot, but its accuracy increases faster and is slightly higher at the end of the semester. At the start of the semester at snapshot S1, course A has an average balanced accuracy between 60% and 65% and course B around 70%. Nearly halfway through the semester, before the first evaluation, we see an average balanced accuracy around 70% for course A at snapshot S5 and between 70% and 75% for course B at snapshot S8. After the first evaluation, we can make predictions with a balanced accuracy between 75% and 80% for both courses. The predictions for course B stay within this range for the rest of the semester, but for course A we can consistently make predictions with an average balanced accuracy of 80% near the end of the semester.

Compared to the accuracy results of Kovacic (2012), we see a 15-20% increase for our balanced accuracy results. Our balanced accuracy results are similar to the accuracy results of Livieris et al. (2019), who used semi-supervised machine learning. Asif et al. (2017) achieve an accuracy of about 80% when using one cohort of training and another cohort for testing, which is again similar to our balanced accuracy results. All of these studies used prior academic history as the basis for their methods, which we do not use in our framework. We also see similar results as compared to Vihavainen (2013) where we don't have to rely on data collection that interferes with the learning process. Note that we are comparing the basic accuracy results of prior studies with the more reliable balanced accuracy results of our framework.

F_1-scores follow the same trend as balanced accuracy, but the inclination is even more pronounced because it starts lower and ends higher. It shows another sharp improvement of predictive performance for both courses when students practice their programming skills in preparation of the final exam (snapshot E3). This underscores the need to keep organising final summative assessments as catalysts of learning, even for courses with a strong focus on active learning.

The variation in predictive accuracy for a group of corresponding snapshots is higher for course A than for course B. This might be explained by the fact that successive editions of course B use the same set of exercises, supplemented with evaluation and exam exercises from the previous edition, whereas each edition of course A uses a different selection of exercises.

Predictions made with training sets from the same student cohort (5-fold cross-validation) perform better than those with training sets from different cohorts (see supplementary material for details). This is more pronounced for F_1-scores than for balanced accuracy but the differences are small enough so that nothing prevents us from building classification models with historical data from previous student cohorts to make pass/fail predictions for the current cohort, which is something that can't be done in practice with data from the same cohort as pass/fail information is needed during the training phase. In addition, we found no significant performance differences for classification models using data from a single course edition or combining data from two course editions. Given that cohort sizes are large enough, this tells us that accurate predictions can already be made in practice with historical data from a single course edition. This is also relevant when the structure of a course changes, because we can only make predictions from historical data for course editions whose snapshots align.

The need to align snapshots is also the reason why we had to build separate models for courses A and B since both have differences in course structure. The models, however, were built using the same set of feature types. Because course B does not work with hard deadlines, deadline-related feature types could not be computed for its snapshots. This missing data and associated features had no impact on the performance of the predictions. Deliberately dropping the same feature types for course A also had no significant effect on the performance of predictions, illustrating that the training phase is where classification algorithms decide themselves how the individual features will contribute to the predictions. This frees us from having to determine the importance of features beforehand, allows us to add new features that might contribute to predictions even if they correlate with other features, and makes it possible to investigate afterwards how important individual features are for a given classifier (see section 3.3).

Even though the structure of the courses is quite different, our method achieves high accuracy results for both courses. The results for course A with reduced features also still gives accurate results. This indicates that the method should be generalisable to other courses where similar

data can be collected, even if the structure is quite different or when some features are impossible to calculate due to the course structure.

3.2 Early detection

Accuracy of predictions systematically increases as we capture more of student behaviour during the semester. But surprisingly we can already make quite accurate predictions early on in the semester, long before students take their first evaluation. Because of the steady trend, predictions for course B at the start of the semester are already reliable enough to serve as input for student feedback or teacher interventions. It takes some more time to identify at-risk students for course A, but from week four or five onwards the predictions may also become an instrument to design remedial actions for this course. Hard deadlines and graded exercises are a strong enforcement of active learning behaviour on the students of course A, and might disguise somewhat more the motivation of students to work on their programming skills. This might explain why it takes a bit longer to properly measure student motivation for course A than for course B.

3.3 Interpretability

So far, we have considered classification models as black boxes in our longitudinal analysis of pass/fail predictions. However, many machine learning techniques can tell us something about the contribution of individual features to make the predictions. In the case of our pass/fail predictions, looking at the importance of feature types and linking them to aspects of practising programming skills, might give us insights into what kind of behaviour promotes or inhibits learning, or has no or a minor effect on the learning process. Temporal information can tell us what behaviour makes a steady contribution to learning or where we see shifts throughout the semester.

This interpretability was a considerable factor in our choice of the classification algorithms we investigated in this study. Since we identified logistic regression as the best-performing classifier, we will take a closer look at feature contributions in its models. These models are explained by the feature weights in the logistic regression equation, so we will express the importance of a feature as its actual weight in the model. We use a temperature scale when plotting importances: white for zero importance, a red gradient for positive importance values and a blue gradient for negative importance values. A feature importance w can be interpreted as follows for logistic regression models: an increase of the feature value by one standard deviation increases the odds of passing the course by a factor of e^w when all other feature values remain the same (Molnar, 2019). The absolute value of the importance determines the impact the feature has on predictions. Features with zero importance have no impact because $e^0 = 1$. Features represented with a light colour have a weak impact and features represented with a dark colour have a strong impact. As a reference, an importance of 0.7 doubles the odds for passing the course with each standard deviation increase of the feature value, because $e^{0.7} \sim 2$. The sign of the importance determines whether the feature promotes or inhibits the odds of passing the course. Features with a positive importance (red colour) will increase the odds with increasing feature values, and features with a negative importance (blue colour) will decrease the odds with increasing feature values.

To simulate that we want to make predictions for each course edition included in this study, we trained logistic regression models with data from the remaining two editions of the same course. A label "edition 18-19" therefore means that we want to make predictions for the 2018-2019 edition of a course with a model built from the 2016-2017 and 2017-2018 editions of the course. However, in this case we are not interested in the predictions themselves, but in the importance of the features in the models. The importance of all features for each course edition can be found in the supplementary material. We will restrict our discussion by highlighting the importance of a selection of feature types for the two courses.

For course A, we look into the evaluation scores (Figure 8) and the feature types `correct_after_15m` (Figure 9) and `wrong` (Figure 10). Evaluation scores have a very strong impact on predictions, and high evaluation scores increase the odds of passing the course. This comes as no surprise, as both the evaluations and exams are summative assessments that are organised and graded in the

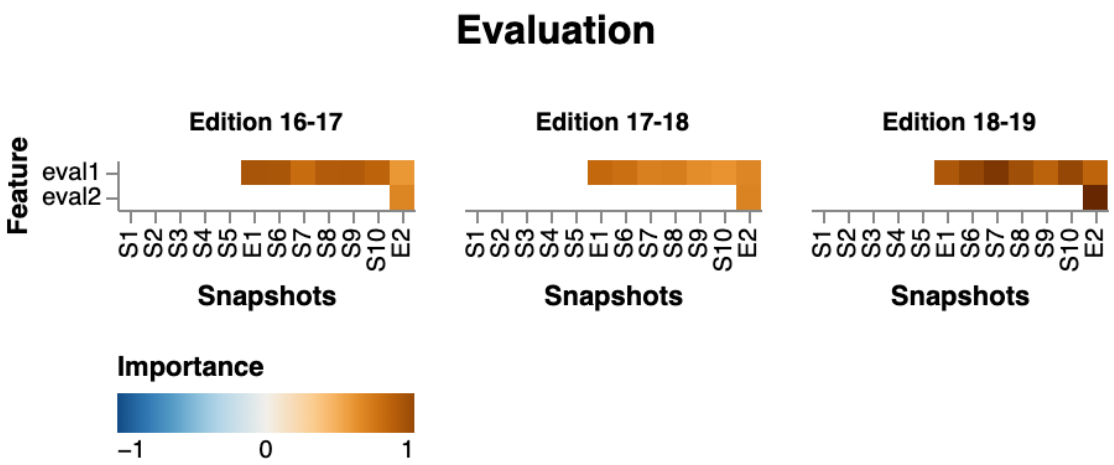

Figure 8: Importance of evaluation scores in the logistic regression models for course A (full feature set). Reds mean that a growth in the feature value for a student increases the odds of passing the course for that student. The darker the colour the larger this increase will be.

same way. Although the difficulty of evaluation exercises is lower than those of exam exercises, evaluation scores already are good predictors for exam scores. Also note that these features only show up in snapshots taken at or after the corresponding evaluation. They have zero impact on predictions for earlier snapshots, as the information is not available at the time these snapshots are taken.

The second feature type we want to highlight is `correct_after_15m`: the number of exercises in a series where the first correct submission was made within fifteen minutes after the first submission (Figure 9). Note that we can't directly measure how long students work on an exercise, as they may write, run and test their solutions on their local machine before their first submission to the learning platform. Rather, this feature type measures how long it takes students to find and remedy errors in their code (debugging), after they start getting automatic feedback from the learning platform.

For exercise series in the first unit of course A (series 1-5), we generally see that the features have a positive impact (red). This means that students will more likely pass the course if they are able to quickly remedy errors in their solutions for these exercises. The first and fourth series are an exception here. The fact that students need more time for the first series might reflect that learning something new is hard at the beginning, even if the exercises are still relatively easy. Series 4 of course A covers strings as the first compound data type of Python in combination with nested loops, where (unnested) loops themselves are covered in series 3. This complex combination might mean that students generally need more time to debug the exercises in series 4.

For the series of the second unit (series 6-10), we observe two different effects. The impact of these features is zero for the first few snapshots (grey bottom left corner). This is because the exercises from these series were not yet published at the time of those snapshots, where all series of the first unit were available from the start of the semester. For the later snapshots, we generally see a negative (blue) weight associated with the features. It might seem counterintuitive and contradicts the explanation given for the series of the first unit. However, the exercises of the second unit are a lot more complex than those of the first unit. This up to a point that even for good students it is hard to debug and correctly solve an exercise in only 15 minutes. Students that need less than 15 minutes at this stage might be bypassing learning by copying solutions from fellow students instead of solving the exercises themselves. An exception to this pattern are the few red squares forming a diagonal in the middle of the bottom half. These squares correspond with exercises that are solved as soon as they become available as opposed to waiting for the deadline. A possible explanation for these few slightly positive weights is that these exercises are

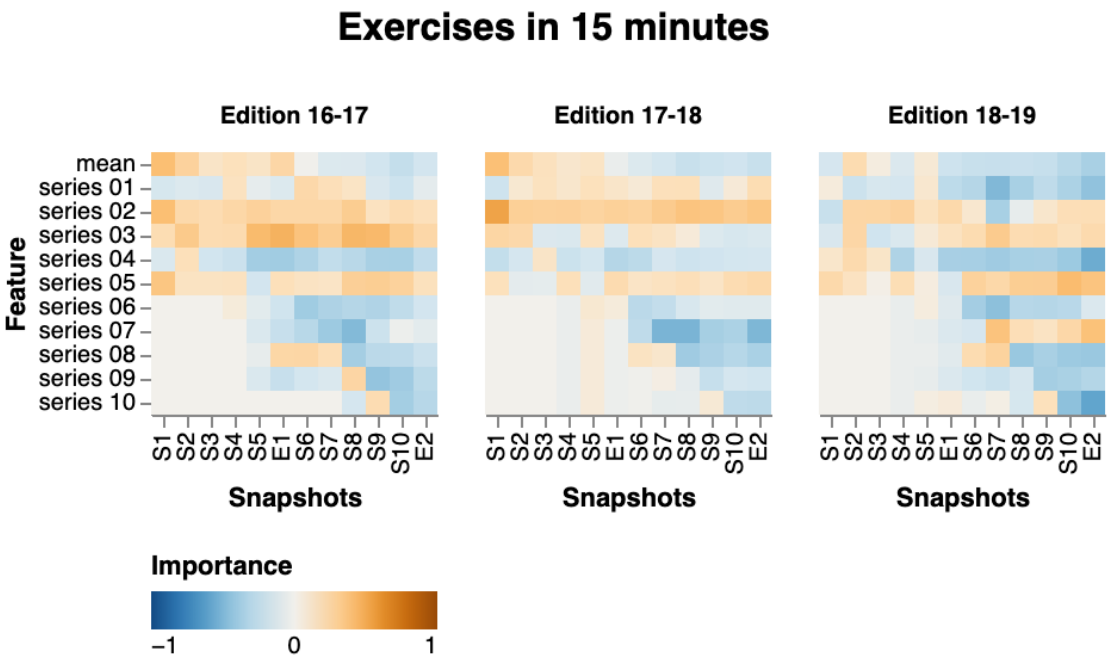

Figure 9: Importance of feature type `correct_after_15m` (the number of exercises in a series where the first correct submission was made within fifteen minutes after the first submission) in logistic regression models for course A (full feature set). Reds mean that a growth in the feature value for a student increases the odds of passing the course for that student. The darker the colour the larger this increase will be. Blues mean that a growth in the feature value decreases the odds. The darker the colour the larger this decrease will be.

solved by highly-motivated, top students.

Finally, if we look at the feature type `wrong` (Figure 10), submitting a lot of submissions with logical errors mostly has a positive impact on the odds of passing the course. This underscores the old adage that practice makes perfect, as real learning happens where students learn from their mistakes. Exceptions to this rule are found for series 2, 3, 9 and 10. The lecturer and teaching assistants identify the topics covered in series 2 and 9 by far as the easiest topics covered in the course, and identify the topics covered in series 3, 6 and 10 as the hardest. However, it does not feel very intuitive that being stuck with logical exercises longer than other students either inhibits the odds for passing on topics that are extremely hard or easy or promotes the odds on topics with moderate difficulty. This shows that interpreting the importance of feature types is not always straightforward.

For course B, we look into the evaluation scores (Figure 11) and the feature types `comp_error` (Figure 12) and `wrong` (Figure 13). The importance of evaluation scores is similar as for course A, although their relative impact on the predictions is slightly lower. The latter might be caused by automatic grading of evaluation exercises, where exam exercises are graded by hand. The fact that the second evaluation is scheduled a little bit earlier in the semester than for course A, makes that pass/fail predictions for course B can also rely earlier on this important feature. However, we do not see a similar increase of the global performance metrics around the second evaluation of course B, as we see for the first evaluation.

Learning to code requires mastering two major competences: i) getting familiar with the syntax rules of a programming language to express the steps for solving a problem in a formal way, so that the algorithm can be executed by a computer, and ii) problem solving itself. As a result, we can make a distinction between different kinds of errors in source code. Compilation errors are mistakes against the syntax of the programming language, whereas logical errors result from

Figure 10: Importance of feature type `wrong` (the number of wrong submissions in a series) in logistic regression models for course A (full feature set). Reds mean that a growth in the feature value for a student increases the odds of passing the course for that student. The darker the colour the larger this increase will be. Blues mean that a growth in the feature value decreases the odds. The darker the colour the larger this decrease will be

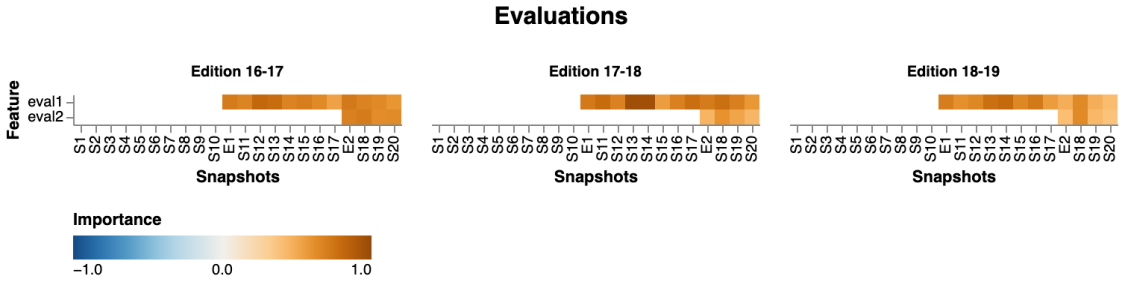

Figure 11: Importance of evaluation scores in the logistic regression models for course B. Reds mean that a growth in the feature value for a student increases the odds of passing the course for that student. The darker the colour the larger this increase will be.

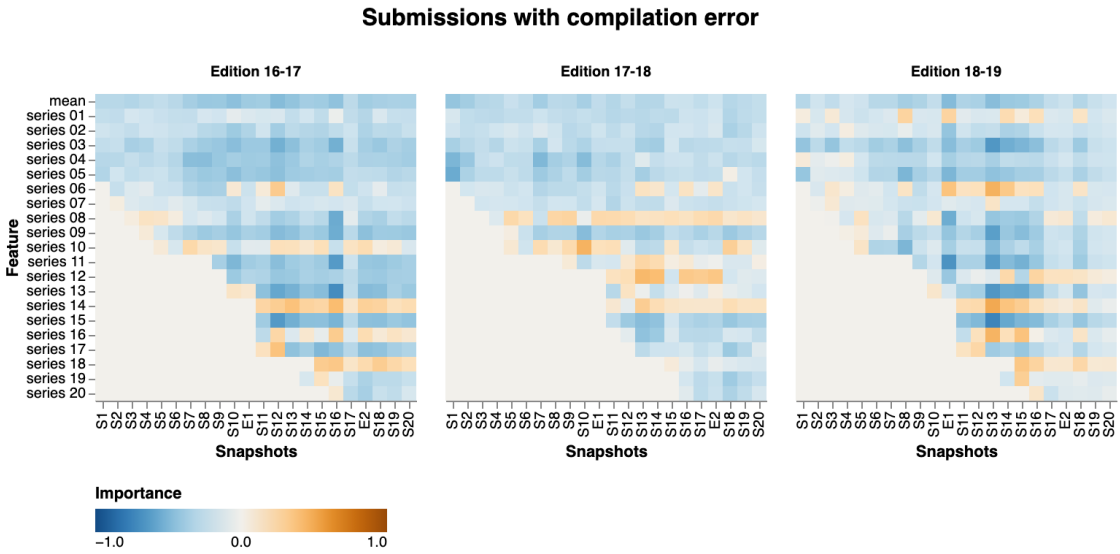

Figure 12: Importance of feature type `comp_error` (the number of submissions with compilation errors in a series) in logistic regression models for course B. Reds mean that a growth in the feature value for a student increases the odds of passing the course for that student. The darker the colour the larger this increase will be. Blues mean that a growth in the feature value decreases the odds. The darker the colour the larger this decrease will be.

solving a problem with a wrong algorithm. When comparing the importance of the number of compilation (Figure 12) and logical errors (Figure 13) students make while practising their coding skills, we see a clear difference. Making a lot of compilation errors has a negative impact on the odds for passing the course (blue colour dominates in Figure 12), whereas making a lot of logical errors makes a positive contribution (red colour dominates in Figure 13). This aligns with the claim of Edwards et al. (2018) that problem solving is a higher-order learning task in Bloom's Taxonomy (analysis and synthesis) than language syntax (knowledge, comprehension, and application). Students who get stuck longer in the mechanics of a programming language will more likely fail the course, whereas students who make a lot of logical errors and properly learn from them will more likely pass the course. So making mistakes is beneficial for learning, but it depends what kind of mistakes. We also looked at the number of solutions with logical errors while interpreting feature types for course A. Although we hinted there towards the same conclusions as for course B, the signals were less consistent. This shows that interpreting feature importances always needs to take the educational context into account. This can also be seen in Figure 9, where some weeks contribute positively and some negatively. The reasons for these differences depend on the content of the course, which requires knowledge of the course contents to interpret correctly.

4 Conclusions and future work

In this manuscript, we presented a classification framework for predicting if students will likely pass or fail introductory programming courses. The framework already yields high-accuracy predictions early on in the semester and is privacy-friendly because it only works with metadata from programming challenges solved by students while working on their programming skills. Being able to identify at-risk students early on in the semester opens windows for remedial actions to improve the overall success rate of students.

We validated the framework by building separate classifiers for two courses because of differences in course structure, but using the same set of features for training models. The results showed

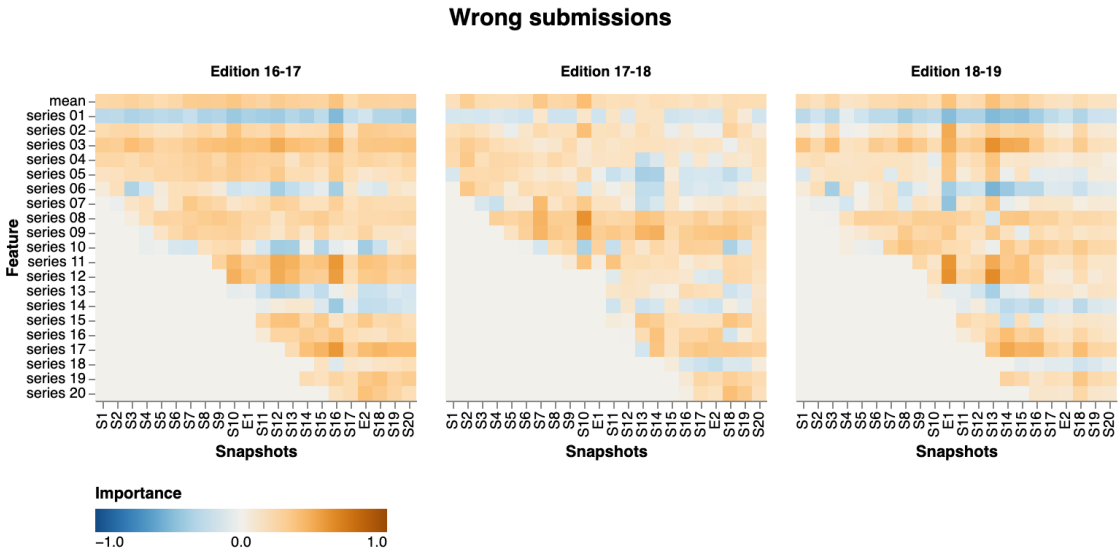

Figure 13: Importance of feature type `wrong` (the number of submissions with logical errors in a series) in logistic regression models for course B. Reds mean that a growth in the feature value for a student increases the odds of passing the course for that student. The darker the colour the larger this increase will be. Blues mean that a growth in the feature value decreases the odds. The darker the colour the larger this decrease will be.

that submission metadata from previous student cohorts can be used to make predictions about the current cohort of students, even if course editions use different sets of exercises or the courses are structured differently. Making predictions requires aligning snapshots between successive editions of a course, where students have the same expected progress at corresponding snapshots. Historical metadata from a single course edition suffices if group sizes are large enough. Different classification algorithms can be plugged into the framework, but logistic regression resulted in the best-performing classifiers.

Apart from their application to make pass/fail predictions, an interesting side-effect of classification models that map indirect measurements of learning behaviour onto mastery of programming skills is that they allow us to interpret what behavioural aspects contribute to learning to code. Visualisation of feature importance turned out to be a useful instrument for linking individual feature types with student behaviour that promotes or inhibits learning. We applied this interpretability to some important feature types that popped up for the two courses included in this study.

Our study has several strengths and promising implications for future practice and research. First, we were able to predict success based on historical metadata from earlier cohorts, and we are already able to do that early on in the semester. In addition to that, the accuracy of our predictions is similar to those of earlier efforts (Asif et al., 2017; Kovacic, 2012; Vihavainen, 2013) while we are not using prior academic history or interfering with the students' usual learning workflows. However, there are also some limitations and work for the future. While our visualisations of the features (Figures 9 through 13) are helpful to indicate which features are important at which stage of the course in view of increasing versus decreasing the odds of passing the course, they may not be oversimplified and need to be carefully interpreted and placed into context. This is where the expertise and experience of teachers comes in. These visualisations can be interpreted by teachers and further contextualised towards the specific course objectives. For example, teachers know the content and goals of every series of exercises and they can use the information presented in our visualisations in order to investigate why certain series of exercises are more or less important in view of passing the course. In addition, they may use the information to further redesign their course.

We can thus conclude that the proposed framework achieves the objectives set for accuracy, early prediction and interpretability. Having this new framework at hand immediately raises some follow-up research questions that urge for further exploration: i) Do we inform students about their odds of passing a course? How and when do we inform students about their performance in an educationally responsible way? What learning analytics do we use to present predictions to students, and do we only show results or also explain how the data led to the results? How do students react to the announcement of their chance at passing the course? How do we ensure that students are not demotivated? ii) What actions could teachers take upon early insights which students will likely fail the course? What recommendations could they make to increase the odds that more students will pass the course? How could interpretations of important behavioural features be translated into learning analytics that give teachers more insight into how students learn to code? iii) Can we combine student progress (what programming skills does a student already have and at what level of mastery), student preferences (what skills does a student wants to improve on), and intrinsic properties of programming exercises (what skills are needed to solve an exercise and how difficult is it) into dynamic learning paths that recommend exercises to optimise the learning effect for individual students?

References

Asif, R., Merceron, A., Ali, S. A., & Haider, N. G. (2017). Analyzing undergraduate students' performance using educational data mining. *Computers & Education*, *113*, 177–194. URL: https://www.sciencedirect.com/science/article/pii/S0360131517301124. doi:10.1016/j.compedu.2017.05.007.

Baker, R. S. J. d., & Yacef, K. (2009). The State of Educational Data Mining in 2009: A Review and Future Visions. *Journal of Educational Data Mining*, *1*, 3–17. URL: https://jedm.educationaldatamining.org. doi:10.5281/zenodo.3554657. Number: 1.

Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bulletin*, *39*, 32–36. URL: https://doi.org/10.1145/1272848.1272879. doi:10.1145/1272848.1272879.

Binns, R., Van Kleek, M., Veale, M., Lyngs, U., Zhao, J., & Shadbolt, N. (2018). 'It's Reducing a Human Being to a Percentage': Perceptions of Justice in Algorithmic Decisions. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* CHI '18 (pp. 1–14). New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/3173574.3173951. doi:10.1145/3173574.3173951.

Black, P., & Wiliam, D. (1998). Assessment and Classroom Learning. *Assessment in Education: Principles, Policy & Practice*, *5*, 7–74. URL: https://doi.org/10.1080/0969595980050102. doi:10.1080/0969595980050102. Publisher: Routledge _eprint: https://doi.org/10.1080/0969595980050102.

Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. In *Machine Learning* (pp. 273–297).

Dutt, A., Ismail, M. A., & Herawan, T. (2017). A Systematic Review on Educational Data Mining. *IEEE Access*, *5*, 15991–16005. doi:10.1109/ACCESS.2017.2654247. Conference Name: IEEE Access.

Edwards, J. M., Fulton, E. K., Holmes, J. D., Valentin, J. L., Beard, D. V., & Parker, K. R. (2018). Separation of syntax and problem solving in Introductory Computer Programming. In *2018 IEEE Frontiers in Education Conference (FIE)* (pp. 1–5). doi:10.1109/FIE.2018.8658852 iSSN: 2377-634X.

Feldman, M. Q., Wang, Y., Byrd, W. E., Guimbretière, F., & Andersen, E. (2019). Towards answering “Am I on the right track?” automatically using program synthesis.

In *Proceedings of the 2019 ACM SIGPLAN Symposium on SPLASH-E* SPLASH-E 2019 (pp. 13–24). New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/3358711.3361626. doi:10.1145/3358711.3361626.

Ferguson, T. S. (1982). An Inconsistent Maximum Likelihood Estimate. *Journal of the American Statistical Association*, *77*, 831–834. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1982.10477894. doi:10.1080/01621459.1982.10477894. Publisher: Taylor & Francis _eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1982.10477894.

Goold, A., & Rimmer, R. (2000). Factors affecting performance in first-year computing. *ACM SIGCSE Bulletin*, *32*, 39–43. URL: https://doi.org/10.1145/355354.355369. doi:10.1145/355354.355369.

Grgić-Hlača, N., Zafar, M. B., Gummadi, K. P., & Weller, A. (2018). The Case for Process Fairness in Learning: Feature Selection for Fair Decision Making. In *Symposium on Machine Learning and the Law* (p. 11). Barcelona, Spain.

Honour, W. (1986). Predicting student performance in a beginning computer science class. *ACM SIGCSE Bulletin*, . URL: https://dl.acm.org/doi/abs/10.1145/953055.5701. doi:10.1145/953055.5701. Publisher: ACM PUB27 New York, NY, USA.

Khalifa, M., & Lam, R. (2002). Web-based learning: effects on learning process and outcome. *IEEE Transactions on Education*, *45*, 350–356. doi:10.1109/TE.2002.804395. Conference Name: IEEE Transactions on Education.

Kleinbaum, D. G. (1994). Introduction to Logistic Regression. In D. G. Kleinbaum (Ed.), *Logistic Regression: A Self-Learning Text* Statistics in the Health sciences (pp. 1–38). New York, NY: Springer. URL: https://doi.org/10.1007/978-1-4757-4108-7_1. doi:10.1007/978-1-4757-4108-7_1.

Kovacic, Z. (2012). Predicting student success by mining enrolment data., . URL: https://repository.openpolytechnic.ac.nz/handle/11072/1486. Accepted: 2013-04-10T01:20:26Z.

Livieris, I. E., Drakopoulou, K., Tampakas, V. T., Mikropoulos, T. A., & Pintelas, P. (2019). Predicting Secondary School Students' Performance Utilizing a Semi-supervised Learning Approach. *Journal of Educational Computing Research*, *57*, 448–470. URL: https://doi.org/10.1177/0735633117752614. doi:10.1177/0735633117752614. Publisher: SAGE Publications Inc.

Molnar, C. (2019). *Interpretable Machine Learning*. URL: https://christophm.github.io/interpretable-ml-book/.

OECD (2021). *OECD Digital Education Outlook 2021*. URL: https://www.oecd-ilibrary.org/content/publication/589b283f-en type: doi:https://doi.org/10.1787/589b283f-en.

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*, 2825–2830. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

Prince, M. (2004). Does Active Learning Work? A Review of the Research. *Journal of Engineering Education*, *93*, 223–231. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/j.2168-9830.2004.tb00809.x. doi:10.1002/j.2168-9830.2004.tb00809.x. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/j.2168-9830.2004.tb00809.x.

Rountree, N., Rountree, J., Robins, A., & Hannah, R. (2004). Interacting factors that predict success and failure in a CS1 course. In *Working group reports from ITiCSE on Innovation and technology in computer science education* ITiCSE-WGR '04 (pp. 101–104). New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/1044550.1041669. doi:10.1145/1044550.1041669.

Svetnik, V., Liaw, A., Tong, C., Culberson, J. C., Sheridan, R. P., & Feuston, B. P. (2003). Random Forest: A Classification and Regression Tool for Compound Classification and QSAR Modeling. *Journal of Chemical Information and Computer Sciences*, *43*, 1947–1958. URL: https://doi.org/10.1021/ci034160g. doi:10.1021/ci034160g. Publisher: American Chemical Society.

Vihavainen, A. (2013). Predicting Students' Performance in an Introductory Programming Course Using Data from Students' Own Programming Process. In *2013 IEEE 13th International Conference on Advanced Learning Technologies* (pp. 498–499). doi:10.1109/ICALT.2013.161 iSSN: 2161-377X.

Vihavainen, A., Luukkainen, M., & Kurhila, J. (2013). Using students' programming behavior to predict success in an introductory mathematics course. In *Educational Data Mining 2013*. Citeseer.

Wasik, S., Antczak, M., Badura, J., Laskowski, A., & Sternal, T. (2018). A Survey on Online Judge Systems and Their Applications. *ACM Computing Surveys*, *51*, 3:1–3:34. URL: https://doi.org/10.1145/3143560. doi:10.1145/3143560.

Watson, C., & Li, F. W. (2014). Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* ITiCSE '14 (pp. 39–44). New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/2591708.2591749. doi:10.1145/2591708.2591749.

Wiliam, D. (2011). What is assessment for learning? *Studies in Educational Evaluation*, *37*, 3–14. URL: https://www.sciencedirect.com/science/article/pii/S0191491X11000149. doi:10.1016/j.stueduc.2011.03.001.

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., Bouwman, J., Brookes, A. J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C. T., Finkers, R., Gonzalez-Beltran, A., Gray, A. J. G., Groth, P., Goble, C., Grethe, J. S., Heringa, J., 't Hoen, P. A. C., Hooft, R., Kuhn, T., Kok, R., Kok, J., Lusher, S. J., Martone, M. E., Mons, A., Packer, A. L., Persson, B., Rocca-Serra, P., Roos, M., van Schaik, R., Sansone, S.-A., Schultes, E., Sengstag, T., Slater, T., Strawn, G., Swertz, M. A., Thompson, M., van der Lei, J., van Mulligen, E., Velterop, J., Waagmeester, A., Wittenburg, P., Wolstencroft, K., Zhao, J., & Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, *3*, 160018. URL: https://www.nature.com/articles/sdata201618. doi:10.1038/sdata.2016.18. Bandiera_abtest: a Cg_type: Nature Research Journals Number: 1 Primary_atype: Comments & Opinion Publisher: Nature Publishing Group Subject_term: Publication characteristics;Research data Subject_term_id: publication-characteristics;research-data.

Xing, W., & Du, D. (2019). Dropout Prediction in MOOCs: Using Deep Learning for Personalized Intervention. *Journal of Educational Computing Research*, *57*, 547–570. URL: https://doi.org/10.1177/0735633118757015. doi:10.1177/0735633118757015. Publisher: SAGE Publications Inc.

A Feature types

- `subm`: numbers of submissions by student in series
- `nosubm`: number of exercises student did not submit any solutions for in series
- `first_dl`: time difference in seconds between student's first submission in series and deadline of series
- `last_dl`: time difference in seconds between student's last submission in series before deadline and deadline of series
- `nr_dl`: number of correct submissions in series by student before series' deadline
- `correct`: number of correct submissions in series by student
- `after_correct`: number of submissions by student after their first correct submission in the series
- `before_correct`: number of submissions by student before their first correct submission in the series
- `time_series`: time difference in seconds between the student's first and last submission in the series
- `time_correct`: time difference in seconds between the student's first submission in the series and their first correct submission in the series
- `wrong`: number of submissions by student in series with logical errors
- `comp_error`: number of submissions by student in series with compilation errors
- `runtime_error`: number of submissions by student in series with runtime errors
- `correct_after_5m`: number of exercises where first correct submission by student was made within five minutes after first submission
- `correct_after_15m`: number of exercises where first correct submission by student was made within fifteen minutes after first submission
- `correct_after_2h`: number of exercises where first correct submission by student was made within two hours after first submission
- `correct_after_24h`: number of exercises where first correct submission by student was made within twenty-four hours after first submission