Transfer-Learning Methods in Programming Course Outcome Prediction

JARKKO LAGUS, KRISTA LONGI, ARTO KLAMI, and ARTO HELLAS,
University of Helsinki, Finland

The computing education research literature contains a wide variety of methods that can be used to identify students who are either at risk of failing their studies or who could benefit from additional challenges. Many of these are based on machine-learning models that learn to make predictions based on previously observed data. However, in educational contexts, differences between courses set huge challenges for the generalizability of these methods. For example, traditional machine-learning methods assume identical distribution in all data—in our terms, traditional machine-learning methods assume that all teaching contexts are alike. In practice, data collected from different courses can be very different as a variety of factors may change, including grading, materials, teaching approach, and the students.

Transfer-learning methodologies have been created to address this challenge. They relax the strict assumption of identical distribution for training and test data. Some similarity between the contexts is still needed for efficient learning. In this work, we review the concept of transfer learning especially for the purpose of predicting the outcome of an introductory programming course and contrast the results with those from traditional machine-learning methods. The methods are evaluated using data collected in situ from two separate introductory programming courses.

We empirically show that transfer-learning methods are able to improve the predictions, especially in cases with limited amount of training data, for example, when making early predictions for a new context. The difference in predictive power is, however, rather subtle, and traditional machine-learning models can be sufficiently accurate assuming the contexts are closely related and the features describing the student activity are carefully chosen to be insensitive to the fine differences.

CCS Concepts: • **Social and professional topics → Computer science education**; • **Computing methodologies → Machine learning**; **Transfer learning**;

Additional Key Words and Phrases: Machine learning, transfer learning, source code snapshots, introductory programming, educational data mining, learning analytics, novice programmers, course outcome prediction

ACM Reference format:
Jarkko Lagus, Krista Longi, Arto Klami, and Arto Hellas. 2018. Transfer-Learning Methods in Programming Course Outcome Prediction. *ACM Trans. Comput. Educ.* 18, 4, Article 19 (July 2018), 18 pages.
https://doi.org/10.1145/3152714

This work was supported in part by the Academy of Finland (decision numbers 251170 and 266969).
Authors' addresses: J. Lagus, K. Longi, A. Klami, and A. Hellas, University of Helsinki, Department of Computer Science, P. O. Box 68, Helsinki, FI-00014, Finland; email: firstname.lastname@helsinki.fi.

1 INTRODUCTION

Educational sciences have started to adopt machine-learning methods, computer algorithms that learn predictive models based on observed data. These methods have been applied in research to, for example, automate tutoring [17], predict course outcomes [23], identify at-risk students [1], detect student characteristics [7], predict learning disabilities [25, 40], model students' learning process [30], and much more. Many of these tasks can be performed transparently, as they only use data that can be naturally gathered from existing learning environments.

One stream of such research is related to the analysis of source code snapshots, i.e., process data, that is recorded as students work on programming assignments [18]. Typically, the models are applied in post hoc manner for analyzing data from a single course and semester, with only a few studies seeking to perform cross-course or cross-context analysis [18]. Analyzing multiple contexts, such as the same course during different semesters or similar courses in different universities, in combination has two obvious advantages: It increases the amount of available data for learning, and it allows us to study relationships between different contexts.

Cross-context datasets can, however, be problematic from the point of view of traditional machine-learning methods. The core principle of machine learning is that the predictive model is learned from some observations collected for training the model, and, consequently, the algorithms learn to make predictions for future data instances that are similar to those used for training the model—in technical terms, they are assumed to be drawn from the same probability distribution. The more this assumption is violated the less useful the models become. If the contexts are sufficiently different, then it can be better to ignore the different contexts altogether. Indeed, using machine-learning methods to construct a predictive model on data from one semester and using it to predict the outcomes of another semester has been observed to produce worse predictions than using data from the same semester for both learning the model and making the predictions [1, 6].

In this work, which is motivated for the need of context-independent methodologies [18, 29], we study whether transfer-learning methodologies could be used for the task of predicting students' outcomes in a cross-context setting. Transfer learning is an area of machine learning that deals with data coming from multiple sources that may have different distributions in data, e.g., different programming assignments, teaching approaches, and so on, and combines that information to improve the prediction model of the task at hand. Naturally, as is often the case with machine-learning approaches, transfer learning relies on having at least some data with values—i.e., course outcomes—from all contexts.

Our analysis is based on two introductory programming courses. One of the courses had lectures and individual assignments, while the other had no lectures, and the students had both pairwise and individual assignments. The grading scheme of both courses also differs from each other.

This article is organized as follows. First, in Section 2, we provide a brief overview of machine learning and transfer learning and visit the related work on predicting course outcomes. Then, in Section 3 we discuss our research methodology and data in more detail, followed by the results of our study in Section 4. The results are discussed in Section 5, where we also address the main limitations of our work. Finally, in Section 6, we briefly review the main contributions of this article and outline possible future directions for study.

2 MACHINE-LEARNING METHODS AND PREDICTION OF COURSE OUTCOMES

In this section, we first provide a brief overview of machine learning with the focus on supervised machine-learning methods and how they are evaluated. We then proceed to describe in more detail the concept of transfer learning and practical methods for it. Finally, we provide an overview of related studies that have sought to predict students' performance in programming courses using machine-learning methods.

2.1 Overview of Machine Learning

The Merriam-Webster online dictionary [24] defines learning as "the activity or process of gaining knowledge or skill by studying, practicing, being taught, or experiencing something: the activity of someone who learns." Machine learning, in contrast, is based on defining a learning algorithm that provides guidelines for the machine, but the actual model is learned from a collection of observations (called data). In other words, the machine has access to a set of examples and its task is to learn a generic model that can, to some extent, make predictions that are similar to those examples. Machine-learning methods are typically applied when it is too difficult to compose an exact algorithm for a particular task. This is also where machine learning differs from traditional algorithm design, where the decision process of the algorithm is encoded manually to the program.

For the purpose of course outcome predictions, the most relevant type of machine learning is called supervised machine learning. The training data are provided as pairs of inputs and outputs, and the task of the learning algorithm is to provide the output for any future input. For example, we can use the knowledge of which assignments a student solved during the first half of the course as the input and the final course grade as the output. After training a machine-learning model based on a collection of such pairs, we can use it for predicting the grade for any future student at halfway through the course.

2.1.1 Data and Features. Machine-learning models are trained on observed data. Furthermore, data quality and quantity strongly influence what can be achieved—if we are only given the age of the student, we cannot hope to make accurate predictions about his or her study performance. However, a rich representation of the students' past performance and activity during a course should be sufficient for creating somewhat accurate predictions of the final exam performance.

In typical modeling scenarios the data are provided as a set of training instances, here students. The information available for each student is represented as a feature vector, a set of features that each describe a particular property of that student. The features can be either background information, such as the student's year of birth or whether the student has previous experience on the topic, or they may be extracted from data that a course management system collects as the student works on course tasks—here, a typical feature could be whether the student completed a particular assignment or not.

Typically, the features used for training the model are a result of manual engineering effort that aims to produce a set of features that are maximally rich in capturing the performance of the students, while still being reliable and easy to estimate. To identify relevant features, we can perform algorithmic feature selection to extract a more compact description, discarding features initially thought to be relevant but that do not really help in solving the learning task. Feature selection can be done with respect to the variable that we are predicting, or in a more generic way where one does not care about the target variable but seeks to maximize the overall information in the data [15]. Highly correlated features are also often removed as much of their information is shared—in such a case, only one of the features may be needed to catch most of the relevant information in a set of features [16].

2.1.2 Learning and Evaluating the Model. The process of machine-learning can be summarized as (1) specification of the model and algorithm, (2) specification of the learning goal, and (3) learning the parameters of the model. For classification, the learning goal is typically to minimize the number of errors made by the algorithm. Learning the parameters of the model means that the machine-learning algorithm seeks to identify parameters that minimize the error.

This is an optimization problem, the details of which heavily depend on the machine-learning algorithm in question. A core differentiation from classical optimization literature, however,

Table 1. A Contingency Table of a Classification Task Where Each Instance Is Categorized Either into Class "True" or Class "False"

		Predicted	
		Class = True	Class = False
Actual	Class = True	TP	FN
	Class = False	FP	TN

concerns the nature of the learning goal—the final goal of machine learning is not to minimize errors in training but instead to minimize the errors for future test instances (generalization error). For finite data collections, the lowest generalization error is typically not obtained by the model that minimizes the training error, but instead we need to regularize the model to avoid overfitting—solutions that describe the training data well but do not generalize to test instances.

To tackle overfitting, the data are usually divided into training and validation sets. Training data are used to build the model, and the goodness of the model is evaluated with a separate validation set to recognize models that have low generalization error. One of the most often used methods for model validation is cross-validation, where a fixed number of partitions, folds, of the data is chosen. Then, iterating over all partitions, one of the partitions is used for validation, and the rest is used for training. The final performance estimate is calculated as the average of these iterations. Research suggests the use of 10 as the partition count [39].

2.1.3 Performance Metrics. The evaluation process described above is carried out with respect to a performance metric that quantifies the quality of the solution. One of the most used ways to visualize performance outcomes when evaluating classification results is the confusion matrix (see Table 1). It is a table that has each class once on both vertical and horizontal axis. As it lists the full classification outcome, it also makes it straightforward to visualize potential imbalances in datasets and to see how the predictions are distributed among different classes.

Table 1 uses abbreviations TP for the number of true positives (positive instances classified correctly), TN for the number of true negatives (negative instances classified correctly), FP for the number of false positives (negative instances classified incorrectly as positive ones), and FN for the number of false negatives (positive instances classified incorrectly as negative ones).

The confusion matrix can be used to calculate various types of performance metrics. One commonly used metric is the F1-score (Equation (3)), which is a weighted average of precision (Equation (1)) and recall (Equation (2)).

$$Precision = \frac{TP}{TP + FP}, \tag{1}$$

$$Recall = \frac{TP}{TP + FN}, \tag{2}$$

$$F1 = \frac{2TP}{2TP + FP + FN}. \tag{3}$$

2.2 Classification Methods

Next, we give a brief overview to the classification methods applied in this work. The methods are chosen as a representative sample of the classifiers that have recently been suggested to perform well for the task of classifying students. At the same time, we also seek a balance with the quantity of classifiers, as our first and foremost goal is not to provide a full classifier comparison but to evaluate the feasibility of transfer-learning methods for our task.

2.2.1 Support Vector Machines. Support vector machine (SVM) is a classifier that builds a set of margins from the training data that best discriminate between the outcomes [11]. In the case of a linearly separable data, linear classifiers can be used. A linear classifier is a geometric model that uses a linear function to construct the decision border from the margins [11]. However, if the data are not linearly separable, then kernel methods can be used as a part of the support vector machine. Kernel methods transform the data points to a new space, where the instances are linearly separable.

2.2.2 Random Forests, Decision Trees, and Bagging. Decision trees are treelike graphs that represent sequences of classification rules. Every node contains a rule that splits the tree into subtrees, and the leaf node defines the final class. Random forest is a collection of such decision trees combined using a technique called bagging.

Bagging, short for bootstrap aggregating, creates multiple random samples with replacement from the training data [39]. These random samples are then used to train a set of classifiers that are used in the final model. When the final model is used for predicting the outcomes for new data, the actual prediction is based on a vote over all the classifiers in the model.

Random forests use bagging on decision trees to prevent some of the inherent problems with decision trees. In decision trees, small changes in training data may result in very different models as a change in top-most rules can alter the structure of the whole subtree [39]. Thus, decision trees are also prone to over-fitting. As shown by Breiman [4], random forests avoid the problem of over-fitting and improve in accuracy when more trees are added.

2.2.3 Boosting. Boosting, like bagging, is based on the idea of using a set of classifiers for building the model. In Boosting, classifiers complement each other and avoid repeating already-known knowledge. If a classifier performs poorly with specific values in the data, then another classifier should compensate the lack of performance and vice versa [39].

Our focus here is on a variant of boosting that is also used in transfer-learning algorithms described later. AdaBoost [12] (short for Adaptive Boosting) is one of the most widely used boosting algorithms. It is based on the idea of iteratively assigning weights on training data instances. When learning a new classifier higher weight is given for instances that are misclassified by the existing set of classifiers, so that the new classifier best complements the ones trained earlier. AdaBoost performs weighting also on another level: The predictions are averaged over the set of classifiers and more weight is given for classifiers with high accuracy [33].

AdaBoost is not a learning algorithm itself, but a method that uses a collection of some learning algorithms. The only requirement for the learning algorithm AdaBoost is used with is that it should support using weights on training instances.

2.3 Transfer Learning

Next, we discuss what transfer learning is, what separates it from traditional supervised machine learning, and when and why it is used. Then, we introduce two transfer-learning algorithms, TrAdaBoost and TransferBoost.

2.3.1 Overview. Traditional machine-learning methodologies are based on the assumption that the training and testing data come from the same distribution. This can lead to a situation where even small changes in the distribution can render the original model unusable for future uses or, at least, lower its performance significantly. Moving away from this assumption is the central goal of transfer learning.

The aim of transfer learning is to extract knowledge from related tasks, called source tasks, to be used for improving the predictive model of the task we are trying to learn, a target task. Transfer learning is applied, for example, in situations where a large collection of data exists for a suitable source task but there is not enough data from the target task available to create an accurate model, or when collecting a new dataset is too expensive and time consuming [26].

For the knowledge transfer to be successful, there has to be some common ground between the source and target task. For example, in the educational context, the courses usually stay the same, but students and lecturer might change between course iterations. The traditional approaches for modeling the outcomes for a new iteration are based on either re-using a model trained on previous iterations, or training the model from scratch on data collected during the early stages of the new iteration. The latter approach suffers from limited data whereas the former may perform badly if the data differs too much from the previous iterations. Transfer learning addresses this challenge by providing mechanisms that use both the previous (source data) and current (target data) course instances to train the models, and even allows using multiple previous iterations as separate source datasets that need not be assumed identical. Studies show that it is almost always beneficial to use multiple sources of data instead of just a single source [10, 41].

Many of the existing transfer-learning algorithms are based on AdaBoost, which was briefly discussed previously. The main difference in transfer-learning adaptations is that instead of creating a model that contains multiple classifiers from the same data, multiple datasets are used to build the classifiers.

2.3.2 TrAdaBoost and TransferBoost. In instance-transfer, which is a type of transfer learning, previously acquired data instances are used, and transfer is conducted through reweighing, including or excluding the data instances. The goal is to choose those data instances that are relevant for the target domain, and to give less weight to those instances that are unlikely to be relevant for the target domain [10].

TrAdaBoost [9] and TransferBoost [10] are both based on instance-transfer. TrAdaBoost uses AdaBoost for the target data, and then weights the classifiers built from the source data to reduce the weight of misclassified source instances to focus more on the target domain [10]. As a part of reducing the weights, TrAdaBoost discards a proportion of the classifiers to ensure convergence.

Eaton and desJardins argue that discarding a proportion of the classifiers may degrade the empirical performance of the classifier [10]. TransferBoost is a generalization of the AdaBoost algorithm that supports multiple data sources. Similarly to TrAdaBoost, TransferBoost tries to weight the samples that match to the distribution of the target domain. The main difference to TrAdaBoost is that TransferBoost assigns a importance weight for each data sample. The importance weights are an estimate for how well the source data can be transferred over to the new domain. They call this measure *transferability* [10]. By introducing this additional importance weight they show that by choosing the dataset and the classifier weights cleverly (for details, see References [10]), the training loss can be bounded to a reasonably low upper-bound even when all the classifiers are included, thus overcoming the problem of having to discard classifiers.

2.4 Predicting Course Outcomes

Next, we briefly visit recent literature that is related to predicting programming course outcomes. In most of the cases, outcome prediction focuses on course grades or on whether the student passes the course, but in some cases, researchers may also focus on the outcomes of individual assignments. For a more extensive review, see, e.g., References [1, 18].

One of the cornerstones of the recent work on using process data for predicting students' course outcomes is the error quotient algorithm [19]. Error quotient is based on the idea that students who are likely to perform poorly struggle to fix consecutive syntax errors, while students who are likely to perform well do not struggle as much. It is one of the few algorithms that has been studied in multiple contexts [20, 29].

Error quotient and its relatives (see, e.g., References [2, 5, 38]) are approaches where the data used for the outcome prediction has required significant insight into the programming process and the internals of the used programming language. This is not necessary, however, as one can also use higher level information such as the number of actions or attempts that a student makes on assignments as a feature [1, 6]. It is also possible to extend this thinking outside the programming environment and, for example, use data extracted from students' in-class behavior such as clicker answers. This was recently proposed by Reference [31] and later extended by Reference [21].

While there exists much work on the topic, so far, there has been little work on methods that study whether specific methods work in different contexts [18]. Some of the exceptions include the work by Petersen et al., who explored the performance of the error quotient algorithm in three different contexts [29], and the study in Reference [6], who explored the applicability of neural networks to predicting cross-semester course outcomes.

3 METHODOLOGY

3.1 Research Questions

The overall theme in our research is predicting whether students will fail or pass an introductory programming course. For this task, we focus on the applicability of data from previous and somewhat dissimilar course instances for the outcome prediction, and study whether transfer-learning methods would be beneficial applicable for the task. Our research questions are as follows:

(1) How do traditionally used machine-learning methods perform in identifying at-risk students over course instances?
(2) Can this performance be improved with the use of transfer-learning methods? If so, then how large is the impact?

With the first research question, we both revisit a previous study from Ahadi et al. [1], as well as form a baseline that shows how machine-learning methods fare when predicting course outcomes in somewhat dissimilar contexts. We study two separate course outcome possibilities, where we first seek to identify those who pass the course and then seek to identify those with overall course points over the course median.

This is then followed by the second research question, where transfer-learning methods are applied to the same data, with the purposes of determining the applicability of transfer-learning methodologies to our problem.

3.2 Context and Data

We study the question of transfer learning in a setup consisting of two courses, mimicking a typical scenario for which transfer learning is expected to be useful. We have collected a large amount of training data from a past course and want to apply the model for a new version of the course for which we can only access a small amount of training examples, obtained, for example, by running a quick pilot with only a few students (or a near-parallel course, see, e.g., Reference [36]). In practice, we conduct the study in a post hoc so that we have large number of students also for the latter course (see the details below) but simulate this scenario by only using a subset of the students in the latter course for training the model.

The data for this study come from two introductory Java programming courses organized at the University of Helsinki. The main differences in the courses were related to lectures,

Table 2. Summary of the Course Differences

	Course 1 (source)	Course 2 (target)
Number of students in dataset	247	101
Of whom passed the course	193	66
Pass-rate	78%	65%
Percentage CS majors	30%	5%
Grading	Linear 0–5	pass/fail
Minimum course points to pass	Over 50%	Over 70%
Exam cutoff	Yes	Yes
Lectures	Yes	No
Individual assignments	Yes	Yes
Weekly (unguided) lab-hours	45	20
Pair-programming	No	Yes

course activities, and grading, but both courses were 7 weeks in length and covered the typical introductory programming course contents such as variables, input and output, lists, methods, objects and elementary sorting and searching algorithms. The material for the second course had extra assignments and additional explanations, but no significant changes were done in the material between the courses.

The first course had weekly lectures that used active learning methodologies as a part of the lectures. The course had 45 weekly lab hours where students could come and work on course assignments under guidance from course assistants and instructors. The second course had no lectures but had 20 weekly lab hours where students could come to work on the course assignments. Both courses had weekly individual assignments, and the second course also had weekly pair programming assignments.

The first course was graded on a linear scale from 0 to 5, where 0 would mean failing the course (less than or equal to 50% of overall points), and 5 would equal the best possible grade (over 90% of overall points). The second course was graded on a fail-pass -scale, where students would have to reach a grade comparable to 3 from the first course to pass the course (over 70% of overall points). In both courses, roughly 1/3 of the course points came from a written exam, and 2/3 of the course points came from completing the course assignments. Finally, an exam cutoff was in place in both courses, which meant that the exam had to be completed with at least half of the overall points for the students to pass the course. The exam was similar in both courses.

The data from the first course have information on 247 students (78% pass rate), while the data from the second course has information on 101 students (65% pass rate). From the first course, 30% of the participants had CS as their major subject, while in the second course, only 5% of the participants had CS as their major. The differences between the two courses are summarized in Table 2.

As the students worked on the individual assignments, their working process was recorded using Test My Code [37], which is an IDE plugin that the courses use for automatic assessment. The plugin also helps students' download course-specific programming assignments and records the students' working process (key presses, timestamps, assignment details) as the students work on the assignments.

At the beginning of the course, the students completed a survey that asked for their background details. The details include year of birth, gender, information on whether they are currently working, and previous programming experience. As pair programming was done in labs, where only one student is logged in on a computer at a time, no data from pair programming was available for this analysis.

3.3 Data Preprocessing

The data from the students' working process were transformed into a form that contains assignment-specific information on events (actions in the IDE), states in which the source compiles, and whether the student was able to complete the assignment. The use of such data was motivated by recent efforts to use programming process data to predict course outcomes (see, e.g., References [1, 18]); information on the proportion of compiling states was included based on Reference [19]. This programming process data wwere then aggregated to the week level to allow better generalizability, as the quantity of assignments can vary between courses.

The working process data were augmented with the background survey details. The background survey was constructed in house and inspired by the surveys in References [1, 38]. Some details such as whether the student has Computer Science as a major were included due to our specific context—in the studied context, students pick their major before entering the University, and as such, choosing Computer Science as a major could indicate motivation. Similarly, information of part-time job and employment was motivated in Reference [3]—while their work did not show a significant difference between groups that have a part-time job and do not have a part-time job, our experiences suggested that this could differ in our context. When constructing the survey, we sought to identify a balance between the quantity of included questions and their approximated predictive value.

As the survey included the option of typing in free-form text in places where numbers were expected (largest program written in lines, hours spent on programming, working hours per week), manual cleaning was conducted to transform phrases such as "maybe 100" and "thousands of hours" into numeric versions. Here, vague statements such as the "maybe 100" were encoded to match the exact number, while plurals were mapped to double the given number, i.e., "thousands of hours" was mapped to 2000. Then, the non-numeric background features such as gender and major were transformed into binary features. Finally, missing background values were mapped to zeros as the used algorithm had no support for missing values. Even though the background features are only a small subset of all possible features, this can create bias, as also in the binary case the actual responses do take values of zero and 1.

Two separate datasets were constructed. In the first dataset, the course outcomes were shown as pass/fail, where the values from 1 to 5 (in the first course) were considered to be passing grades and 0 the failing grade. In the second course, the grading was already a pass/fail, so no changes were done to it. In the second dataset, the students were categorized into above median (or equal) and below median. In the first dataset, the data are biased towards passing students (see Table 2), while the second dataset is balanced.

When possible, the features were normalized using min-max normalization [27]. An overview of the features used is shown in Table 3.

3.4 Classifier Evaluation

The applicability of these features for predicting course outcomes was assessed using a set of classifiers. First, three classifiers (Support Vector Machine, Random Forest, and AdaBoost) are compared to assess the performance of traditional machine-learning methods. Two versions of Support Vector Machine are used, one with a linear kernel and one with a radial bases function kernel. Support Vector Machine and Random Forest have been previously shown to perform well in the task

Table 3. Final Feature Set Used for the Analysis
$N \in \{1, 2, 3, 4, 5, 6, 7\}$

Background features
Has CS as a major
Has previous Java experience
Programming experience: most lines
Programming experience: hours
Gender
Year of birth
Working hours per week
Is currently working
Course result
Features from programming process
Total events on week N assignments
Compiling states on week N assignments
Completed exercises on week N assignments

of identifying at-risk students [1], while AdaBoost is used as a comparison, as it is used as the baseline for the transfer-learning comparisons.

In all cases, feature selection was first performed using a ready-made feature selection algorithm that chooses the most important features based on the results of the used classifier and tries to optimize the number of correctly classified instances. The implementation can be found in the Python library called Scikit-learn [28] under the name SelectFromModel. The same type of classifier was used both in feature selection as well as in classification.

From the transfer-learning algorithms, two AdaBoost-based variants, TrAdaBoost and TransferBoost, were used. The AdaBoost classifier was used to do feature selection on these. In addition, a second version of TrAdaBoost was created that retained all the classifiers to be used in the classification step. The alternate version was implemented due to the arguments by [10] that suggested that leaving out a proportion of the classifiers that TrAdaBoost uses can reduce the overall performance.

All the classifiers were evaluated multiple times and the parameter values that gave the best results for this dataset on average were chosen. The best result here is determined by the F1 metric. AdaBoost and the transfer-learning variants were learned with 100 iterations using Decision tree with maximum depth of one as the base learner. Random Forest was trained using maximum amount of features set to 1 and maximum depth 5 and with 10 estimators. SVMs were trained with regularization parameter set to two, the first one using a linear kernel and the other one using a radial basis function kernel.

In all of our tests, we assumed that there are 100 samples (students) from the source dataset drawn from the larger course, and a smaller number of values ($n = 5, 10, 25$) that corresponds to the number of samples from the target data, which were drawn from the smaller course. In all of the cases where the transfer-learning algorithm had the small number of samples from the target data, the same data were available also for the traditional machine-learning methods.

The samples from the smaller course that remained ($n = 96, 91, 76$) after picking a small number of the values into the training data were used for validating the chosen models. The tests were conducted using 10-fold cross-validation conducted multiple times over the dataset with stratification to form a coherent picture on the performance of the classifiers. The evaluation results are

Table 4. Performance of the Traditional Machine-Learning Methods Compared When Using the Original Fail-Pass Labeling

Classifier	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6
			Source=100, Target=5			
SVC^1	**0.70 0.99 (0.82)**	0.74 0.93 (0.82)	**0.79 0.93 (.86)**	0.79 0.92 (0.85)	0.79 0.93 (0.85)	0.80 0.94 (0.86)
SVC^2	**0.70 1.00 (0.82)**	0.70 1.00 (0.82)	0.72 0.99 (0.83)	0.79 0.96 (0.86)	**0.80 0.95 (0.87)**	0.79 0.96 (0.86)
RandomForest	0.71 0.97 (0.82)	**0.75 0.95 (0.83)**	0.79 0.93 (0.85)	**0.81 0.94 (0.87)**	**0.82 0.94 (0.87)**	**0.82 0.96 (0.88)**
AdaBoost	0.72 0.90 (0.80)	0.75 0.86 (0.80)	0.78 0.92 (0.84)	0.78 0.94 (0.85)	0.78 0.94 (0.85)	0.78 0.96 (0.86)
			Source=100, Target=10			
SVC^1	0.71 0.99 (0.82)	**0.75 0.93 (0.83)**	0.80 .92 (0.86)	0.80 0.91 (0.85)	0.80 0.92 (0.86)	0.81 0.93 (0.87)
SVC^2	**0.71 1.00 (0.83)**	**0.71 1.00 (0.83)**	0.74 0.98 (0.84)	0.79 0.95 (0.86)	**0.81 0.95 (0.87)**	0.79 0.96 (0.86)
RandomForest	0.72 .97 (0.83)	**0.75 0.94 (0.83)**	**0.80 0.93 (0.86)**	**0.82 0.93 (0.87)**	**0.83 0.93 (0.87)**	**0.83 0.95 (0.88)**
AdaBoost	0.73 .87 (0.79)	0.77 0.84 (0.80)	0.80 0.90 (0.84)	0.80 0.90 (0.85)	0.80 0.92 (0.85)	0.82 0.94 (0.87)
			Source=100, Target=25			
SVC^1	**0.70 .98 (0.82)**	0.74 .93 (0.82)	0.81 0.92 (0.86)	0.81 0.91 (0.85)	0.81 0.92 (0.86)	0.82 0.93 (0.87)
SVC^2	**0.70 1.00 (0.82)**	0.70 1.00 (0.82)	0.76 0.97 (0.85)	**0.80 0.94 (0.87)**	0.82 0.94 (0.87)	0.80 0.95 (0.87)
RandomForest	**0.72 0.95 (0.82)**	**0.75 0.94 (0.83)**	**0.80 0.93 (0.86)**	**0.82 0.93 (0.87)**	**0.83 0.93 (0.88)**	**0.83 0.95 (0.89)**
AdaBoost	0.73 0.85 (0.79)	0.78 0.84 (0.80)	0.80 0.89 (0.84)	0.81 0.91 (0.86)	0.81 0.91 (0.86)	0.83 0.93 (0.87)

SVC^1 refers to SVM classifier with linear kernel, SVC^2 refers to SVM with radial basis function kernel. Labels are assigned so that students that passed the course are given positive labels and the rest have negative labels. For each week, results are reported using Precision, Recall, and F1-score.

presented using Precision, Recall, and F1-score, and we focus on the predictive capability during the first 6 weeks of the 7-week course.

4 RESULTS

The results of the traditional machine-learning classifiers are summarized in Table 4 for the dataset with pass and fail division and in Table 5 for the dataset with median division. The results of evaluating AdaBoost with transfer-learning variants of AdaBoost are summarized in Tables 6 and 7, where the first table contains data with pass and fail division and the second table data with median division.

The tables also include details on the source and target sample size. In all tables, the remaining data from the target domain has been used for the validation. For each source and target sample size pair, the entry with highest F1-score is highlighted. We do not include the full contingency table to maintain readability of the results but only a few additional metrics (precision and recall) that contain information about the other metrics, for example, proportion of false negatives is $1 - Recall$. Next, these results are visited in more detail.

4.1 Performance of Traditional Machine-Learning Methods

When predicting whether students will pass the course, the traditional machine-learning models, here SVM, Random Forest, and AdaBoost, perform reasonably well when using only students' background variables and information on events, compiling states, and completed exercises for each week. For these classifiers, the average F1-scores for weeks 1 to 6 range from 0.82 to 0.89, as shown in Table 4. Here, when considering the overall quantity of the weeks where a specific algorithm performs better than others, the Random Forest classifier performs the best.

However, when predicting whether the student will be above median in overall scores, we observe that the task is more challenging. For the traditional classifiers, the average F1-scores for

Table 5. Performance of the Traditional Machine-Learning Methods Compared When Using the Median Split Fail-Pass Labeling

Classifier	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6
			Source=100, Target=5			
SVC[1]	0.57 0.64 (0.59)	0.63 0.62 (0.62)	0.71 0.77 (0.73)	0.66 0.79 (0.72)	0.69 0.80 (0.74)	0.72 0.83 (0.77)
SVC[2]	**0.56 0.73 (0.61)**	**0.58 0.71 (0.63)**	**0.68 0.89 (0.77)**	**0.63 0.91 (0.74)**	**0.67 0.88 (0.76)**	**0.70 0.91 (0.79)**
RandomForest	0.61 0.62 (0.60)	0.61 0.42 (0.47)	0.73 0.64 (0.67)	0.71 0.66 (0.68)	0.74 0.69 (0.70)	0.74 0.75 (0.74)
AdaBoost	0.59 0.60 (0.59)	0.56 0.42 (0.46)	0.70 0.67 (0.67)	0.69 0.73 (0.70)	0.67 0.72 (0.68)	0.70 0.73 (0.71)
			Source=100, Target=10			
SVC[1]	0.56 0.67 (0.60)	0.62 0.65 (0.63)	0.70 0.79 (0.74)	0.66 0.81 (0.72)	0.69 0.81 (0.74)	0.72 0.82 (0.76)
SVC[2]	0.56 0.74 (0.63)	**0.57 0.80 (0.67)**	**0.67 0.90 (0.77)**	**0.62 0.90 (0.74)**	**0.66 0.89 (0.76)**	**0.68 0.91 (0.78)**
RandomForest	**0.60 0.69 (0.64)**	0.66 0.56 (0.59)	0.73 0.67 (0.70)	0.71 0.73 (0.72)	0.73 0.77 (0.75)	0.71 0.80 (0.75)
AdaBoost	0.59 0.63 (0.61)	0.59 0.53 (0.55)	0.70 0.73 (0.71)	0.67 0.76 (0.71)	0.69 0.75 (0.71)	0.71 0.78 (0.74)
			Source=100, Target=25			
SVC[1]	0.58 0.68 (0.62)	0.62 0.68 (0.64)	0.71 0.80 (0.75)	0.67 0.82 (0.74)	0.71 0.81 (0.75)	0.73 0.81 (0.77)
SVC[2]	0.57 0.75 (0.64)	**0.59 0.81 (0.68)**	**0.69 0.89 (0.77)**	**0.65 0.90 (0.75)**	**0.68 0.88 (0.76)**	**0.70 0.90 (0.79)**
RandomForest	**0.62 0.69 (0.65)**	0.67 0.66 (0.66)	0.73 0.73 (0.73)	0.73 0.76 (0.74)	0.73 0.76 (0.74)	0.75 0.79 (0.77)
AdaBoost	0.62 0.65 (0.63)	0.65 0.60 (0.62)	0.72 0.73 (0.73)	0.71 0.75 (0.73)	0.72 0.76 (0.74)	0.74 0.77 (0.75)

SVC[1] refers to SVM classifier with linear kernel, SVC[2] refers to SVM with radial basis function kernel. Labels are assigned so that top half of the best performing students have positive label and the lower half has negative label. For each week, results are reported using Precision, Recall, and F1-score.

Table 6. Performance of the Transfer Machine-Learning Methods Compared When Using the Original Fail-Pass Labeling

Classifier	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6
			Source=100, Target=5			
AdaBoost	0.72 0.90 (0.80)	0.75 0.86 (0.80)	0.78 0.92 (0.84)	**0.78 0.94 (0.85)**	0.78 0.94 (0.85)	0.78 0.96 (0.86)
TrAdaBoost[1]	**0.74 0.94 (0.83)**	**0.76 0.93 (0.84)**	**0.83 0.93 (0.88)**	**0.82 0.89 (0.85)**	**0.83 0.90 (0.86)**	**0.85 0.93 (0.89)**
TrAdaBoost[2]	**0.74 0.94 (0.83)**	0.76 0.92 (0.83)	**0.84 0.93 (0.88)**	0.82 0.88 (0.84)	**0.83 0.90 (0.86)**	**0.85 0.93 (0.89)**
TransferBoost	0.73 0.93 (0.82)	0.75 0.88 (0.81)	0.78 0.95 (0.85)	0.78 0.93 (0.84)	0.79 0.94 (0.85)	0.80 0.94 (0.86)
			Source=100, Target=10			
AdaBoost	0.73 0.87 (0.79)	0.77 0.84 (0.80)	0.80 0.90 (0.84)	0.80 0.90 (0.85)	0.80 0.92 (0.85)	0.82 0.94 (0.87)
TrAdaBoost[1]	**0.75 0.94 (0.83)**	**0.78 0.90 (0.83)**	**0.84 0.93 (0.88)**	**0.82 0.91 (0.86)**	0.84 0.90 (0.86)	**0.85 0.93 (0.89)**
TrAdaBoost[2]	**0.75 0.94 (0.83)**	**0.78 0.90 (0.83)**	**0.84 0.93 (0.88)**	0.82 0.88 (0.85)	**0.83 0.91 (0.87)**	**0.86 0.93 (0.89)**
TransferBoost	0.74 0.92 (0.82)	0.77 0.89 (0.82)	0.79 0.93 (0.85)	0.79 0.93 (0.85)	0.80 0.93 (0.86)	0.81 0.96 (0.88)
			Source=100, Target=25			
AdaBoost	0.73 0.85 (0.79)	0.78 0.84 (0.80)	0.80 0.89 (0.84)	0.81 0.91 (0.86)	0.81 0.91 (0.86)	0.83 0.93 (0.87)
TrAdaBoost[1]	**0.74 0.94 (0.83)**	**0.79 0.89 (0.83)**	**0.84 0.92 (0.88)**	**0.83 0.90 (0.86)**	**0.84 0.90 (0.87)**	**0.86 0.92 (0.89)**
TrAdaBoost[2]	**0.75 0.93 (0.83)**	**0.79 0.89 (0.83)**	**0.84 0.93 (0.88)**	**0.83 0.90 (0.86)**	**0.84 0.91 (0.87)**	**0.86 0.93 (0.89)**
TransferBoost	0.73 0.91 (0.81)	0.76 0.88 (0.81)	0.78 0.94 (0.85)	0.78 0.94 (0.85)	0.79 0.94 (0.86)	0.79 0.96 (0.87)

TrAdaBoost[1] is the original implementation and TrAdaBoost[2] is the modified version as described in Section 3.4. Labels are assigned so that students that passed the course are given positive labels and the rest have negative labels. For each week, results are reported using Precision, Recall, and F1-score.

Table 7. Performance of the Transfer Machine-Learning Methods Compared When Using the Median Split Fail-Pass Labeling

Classifier	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6
			Source=100, Target=5			
AdaBoost	**0.59 0.60 (0.59)**	0.56 0.42 (0.46)	0.70 0.67 (0.67)	0.69 0.73 (0.70)	0.67 0.72 (0.68)	0.70 0.73 (0.71)
TrAdaBoost[1]	0.58 0.62 (0.58)	**0.58 0.67 (0.61)**	**0.71 0.71 (0.71)**	**0.69 0.75 (0.71)**	**0.74 0.73 (0.72)**	0.70 0.80 (0.73)
TrAdaBoost[2]	**0.58 0.62 (0.59)**	0.58 0.65 (0.60)	**0.71 0.71 (0.71)**	**0.69 0.74 (0.71)**	**0.74 0.74 (0.72)**	**0.70 0.82 (0.75)**
TransferBoost	**0.59 0.61 (0.59)**	0.61 0.41 (0.44)	0.72 0.67 (0.69)	0.69 0.73 (0.70)	0.73 0.69 (0.69)	0.72 0.70 (0.69)
			Source=100, Target=10			
AdaBoost	**0.59 0.63 (0.61)**	0.59 0.53 (0.55)	**0.70 0.73 (0.71)**	**0.67 0.76 (0.71)**	0.69 0.75 (0.71)	0.71 0.78 (0.74)
TrAdaBoost[1]	0.57 0.64 (0.59)	**0.57 0.69 (0.61)**	0.70 0.71 (0.70)	**0.68 0.75 (0.71)**	0.74 0.75 (0.73)	**0.70 0.83 (0.76)**
TrAdaBoost[2]	0.57 0.62 (0.58)	**0.58 0.67 (0.61)**	**0.70 0.72 (0.71)**	**0.68 0.76 (0.71)**	**0.76 0.74 (0.74)**	0.69 0.83 (0.75)
TransferBoost	**0.62 0.63 (0.61)**	0.61 0.43 (0.47)	0.72 0.70 (0.70)	0.67 0.77 (0.70)	0.72 0.73 (0.70)	0.71 0.73 (0.71)
			Source=100, Target=25			
AdaBoost	**0.62 0.65 (0.63)**	**0.65 0.60 (0.62)**	**0.72 0.73 (0.73)**	**0.71 0.75 (0.73)**	**0.72 0.76 (0.74)**	0.74 0.77 (0.75)
TrAdaBoost[1]	0.57 0.66 (0.60)	**0.57 0.71 (0.62)**	0.70 0.74 (0.71)	0.69 0.76 (0.71)	0.76 0.72 (0.73)	**0.70 0.85 (0.76)**
TrAdaBoost[2]	0.57 0.68 (0.61)	**0.58 0.70 (0.62)**	0.69 0.73 (0.71)	0.68 0.77 (0.72)	0.76 0.73 (0.73)	**0.71 0.84 (0.76)**
TransferBoost	0.61 0.62 (0.60)	0.62 0.42 (0.47)	0.72 0.69 (0.69)	0.68 0.78 (0.72)	0.69 0.79 (0.73)	0.72 0.78 (0.73)

TrAdaBoost[1] is the original implementation and TrAdaBoost[2] is the modified version as described in Section 3.4. Labels are assigned so that top half of the best performing students have positive label and the lower half has negative label. For each week, results are reported using Precision, Recall, and F1-score.

weeks 1 to 6 range from 0.61 to 0.79, as shown in Table 5. Contrary to the previous case when the task was to predict whether the student will pass the course or not, where Random Forest was the top performing model, classifying the students based on whether they fall onto the top half of the course population seems to work best with the SVM with a non-linear kernel.

4.2 Traditional Machine Learning versus Transfer Learning

To assess whether transfer-learning methodologies perform better in the task than traditional machine-learning methods, we compare the TrAdaBoost, TransferBoost, and AdaBoost algorithms. This comparison is justified due to TrAdaBoost and TransferBoost being based on AdaBoost. Two variants of TrAdaBoost are considered, where one performs predictions only using a proportion of all the constructed classifiers, while the second one uses all the classifiers.

When predicting whether students will pass the course, transfer-learning-based models almost always perform at least marginally better than the traditional model, as shown in Table 6. Here, both TrAdaBoost implementations perform comparably, while the TransferBoost algorithm performs the worst of the transfer-learning algorithms. When comparing the results of the transfer-learning-based models to the traditional models, the transfer-learning-based models perform on par, if not marginally better.

However, when predicting whether the student will be above median in overall scores, the performance is rather poor, as can be seen in Table 7. This can be partially explained by the poor performance of the AdaBoost algorithm for the task (it performs the worst of the selected set of traditional machine-learning methods). Transfer-learning methods seem to have worse performance than the traditional machine-learning methods.

5 DISCUSSION

5.1 Determining Students' Performance

Both traditional machine-learning methods and transfer-learning methods show reasonable performance in determining whether the student will pass the course. This result is in line with previous studies that have identified the possibility of assessing students' performance based on various process- and background-based metrics [1, 19, 22, 32, 34, 38]. However, when comparing our study to the previously mentioned ones, the difference in context was significantly larger, and our study had rather generalized features.

Changing the outcome variable from pass to above median significantly degraded the classifier performance. This result can be partially explained by the imbalanced dataset (approximately 65% of the students in the target data had passed the course): Predicting the majority class is easier, while when the data are balanced, the task may become harder.

Moreover, in our case, the majority of the students who pass the course pass it clearly. That is, there are only a few borderline cases. Median score, however, depends more significantly on the course grading scheme. In the first course, the students had to gain 50% of the overall course points to pass the course, while in the second course, the students had to gain approximately 70% of the overall course points to pass. We suspect that this simple difference was sufficient to fool a majority of the classifiers, as most of the data for the classifiers came from a course where students were effectively expected to work less.

5.2 More Data (Often) Yields Better Results

In general, we observed that increasing data (here, weeks) increases the predictive performance of the algorithms, as expected. This does not, however, mean that additional data always leads to better results, as data quality also matters [42]. In our case, the initial features were selected so that they would try to encode the students' working process in a reliable but generalizable manner. Further feature selection was applied to prune out features that are not useful to guarantee high quality of input data. It is notable, however, that feature selection methods provide only an approximation for the optimal choice; there are 2^n possible feature combinations to be selected from n features, and the choice is often done in a greedy fashion.

In our case, the total number of features in the 6 week dataset was 26, while some other approaches such as that in Reference [1] opt for a more fine-grained approach. In their case, each assignment is considered as a separate feature. It is also possible to build predictive models using just a few variables. Here, the models are typically based on expert feedback, such as in the case of the Error Quotient [19], Normalized Programming State Model [5], and the Repeated Error Density [2]. Naturally, this is not only limited to the domain of programming process data, but information can be drawn from other sources as well (see, e.g., Reference [31]).

5.3 Transfer Learning May Beat Traditional Machine Learning, But...

When comparing near-identical machine-learning algorithms, AdaBoost and TrAdaBoost, the transfer-learning-based implementation almost always had better performance than the non-transfer-learning-based implementation. The improvement was particularly notable when the amount of training data for the target context was very limited, either due to the small number of students or because of evaluating the predictions already after 1–2 weeks, and diminished as the amount of available data increased as expected.

However, when considering all the algorithms that we included in this study, the assumption that transfer-learning algorithms would be better did not hold. The Random Forest algorithm, which has been observed to perform well for the task of identifying at-risk students also in the past (see Reference [1]), showed near-similar performance to that of the TrAdaBoost algorithm. No notable difference between the results of those algorithms was identified, possibly because Random Forest as an ensemble method is effective in avoiding overfitting even to small datasets.

An interesting future work would be to consider transfer-learning extensions of Random Forests. While some preliminary results on transfer learning with Random Forests exist, see, e.g., Reference [14, 35], we could not identify a stable implementation that we could have used for this analysis.

5.4 Notes on Comparing Results between Studies

Directly comparing the performance-specific results in this article to those published in previous works is not meaningful. While we can uniformly state that it is possible to determine students' performance based on their process data, it is not currently possible to assess which one of the methods performs the best—or even if one such method exists.

Here, we chose to make our feature representation as generalizable as possible (e.g., through combining all weekly exercises into a single variable), while still showing that specific features add power to the model. Other studies have typically used at least partially different feature representations as well as different target variables. For example, the authors of Reference [1] sought to determine whether the student belongs to the upper median or the lower median in both a specific exam question and the overall course performance, while the error quotient extension in Watwin [38] used a combination of different metrics, including a programming project and a mid-term exam with context-specific weights as the target outcome.

We repeat the note from Reference [18], where the authors posit that one of the grand challenges in educational data mining and learning analytics in programming is related to replication. There is a need for a repository, where researchers could share their data and results for study and verification purposes:

> [One of the challenges is to] have researchers and practitioners commit to building and maintaining a multi-language, multi-institution, multi-nation learning process data and experiment result database that contains student metadata, logs from systems that the students use, and other details, which could then be used as a reflection point for novel results from a specific context.

5.5 Limitations of Our Study

This study has several limitations, which we address next. First, the main limitation, which was already partially covered in the previous section, is related to the generalizability of our findings. As we used data from two similar but separate courses for the analysis, we cannot state whether the performance levels presented here would generalize to other contexts. This is an issue that, in general, seems to be typical in this line of research (see, e.g., Reference [29])—at the same time, we believe that the general trend of transfer-learning methods marginally improving the performance over traditional machine-learning methods would hold across contexts.

Second, as the contexts are quite similar, we do not yet know how these approaches would perform when the differences between the contexts are more different. It would, for example, be meaningful to conduct a similar study with datasets from different universities across continents.

Third, we made an assumption that the researchers could have some labeled data from the target domain at their disposal. While this assumption is possible in some contexts due to the use of, e.g., "alpha" and "beta" courses, it is likely that the data in these courses would marginally differ from the actual implementations. That is, we do not know how well the supposed labels from the target domain would represent the actual target. At the same time, there exists a variety of transfer-learning methods under the label unsupervised domain adaptation that do not require the labels from the target data [13]. Such methods try to learn the distributions of the target data in an unsupervised way and generalize the source data to the target data.

Finally, we expected that the source and target domains would have equal feature sets and made this a reality through the generalization of the features. While this is not necessarily a limitation as it is possible to do so through normalization of variables, it likely affects the overall performance of the model. If the students' progress in each individual assignment would be assessed separately, then it would likely also improve the performance of the model over a generalization that considers the progress as a whole. One approach to handle this would be the use of heterogeneous domain adaptation [8], which tries to learn a mapping between the source and the target domain.

6 CONCLUSIONS

In this work, we studied methodologies for predicting students' course outcomes in introductory programming courses. More specifically, we studied the applicability of traditional machine-learning methods (SVM, Random Forest, AdaBoost) for predicting the outcomes of a similar but separate course and then analyzed how transfer-learning methods (TrAdaBoost and Transfer-Boost) perform in the same task. The data used for the analysis came from two separate introductory programming courses with different student population, teaching strategies, assessment, and activities. At the same time, the material that both courses used was very similar.

Our results (1) confirm previous studies and reproduce the result that students' course outcomes can be predicted using data from their learning process, (2) confirmed that increase in data in general improves the predictive performance, and (3) showed that transfer-learning methods can improve performance over similar machine-learning methods. At the same time, our experiment expected small amounts of data with information on outcomes from all data sources—a possible solution for this would be the use of unsupervised domain adaptation [13], which could open up the possibility of needing only the students' process data from a context for initial outcome predictions.

We see our results as one of the first steps towards having cross-context methodologies for predicting course outcomes. Such approaches have been called for in a recent ITiCSE working group report [18], and also recently suggested in Reference [6], as they could lead to better understanding of both the factors that contribute to students' performance in programming courses and provide models that could be transferred from one context to another without a significant drop in predictive power.

As a part of our future work, we are looking to further analyze the possibility of having cross-context methodologies for predicting students' course outcomes. Moreover, we are also studying practical interventions where these results could be used to help the students who are expected to perform poorly.

REFERENCES

[1] Alireza Ahadi, Raymond Lister, Heikki Haapala, and Arto Vihavainen. 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 (ICER'15).* ACM, New York, NY, 121–130. DOI: http://dx.doi.org/10.1145/2787622.2787717

[2] Brett A. Becker. 2016. A new metric to quantify repeated compiler errors for novice programmers. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'16).* ACM, New York, NY, 296–301. DOI: http://dx.doi.org/10.1145/2899415.2899463

[3] Susan Bergin and Ronan Reilly. 2005. Programming: Factors that influence success. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'05).* ACM, New York, NY, USA, 411–415. DOI: https://doi.org/10.1145/1047344.1047480

[4] Leo Breiman. 2001. Random forests. *Mach. Learn.* 45, 1 (2001), 5–32.

[5] Adam S. Carter, Christopher D. Hundhausen, and Olusola Adesope. 2015. The normalized programming state model: Predicting student performance in computing courses based on programming behavior. In *Proceedings of the 11th Annual International Conference on International Computing Education Research*. ACM, 141–150.

[6] Karo Castro-Wunsch, Alireza Ahadi, and Andrew Petersen. 2017. Evaluating neural networks as a method for identifying students in need of assistance. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE'17)*. ACM, New York, NY, USA, 111–116. DOI: http://dx.doi.org/10.1145/3017680.3017792

[7] Suleyman Cetintas, Luo Si, Yan Ping Xin, and Casey Hord. 2010. Automatic detection of off-task behaviors in intelligent tutoring systems with machine-learning techniques. *IEEE Trans. Learn. Technol.* 3, 3 (2010), 228–236.

[8] Wenyuan Dai, Yuqiang Chen, Gui-Rong Xue, Qiang Yang, and Yong Yu. 2008. Translated learning: Transfer learning across different feature spaces. In *Advances in Neural Information Processing Systems*. 353–360.

[9] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. 2007. Boosting for transfer learning. In *Proceedings of the 24th international Conference on Machine Learning*. ACM, 193–200.

[10] Eric Eaton and Marie desJardins. 2011. Selective transfer between learning tasks using task-based boosting. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI'11)*. AAAI Press, 337–342.

[11] Peter Flach. 2012. *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press.

[12] Yoav Freund and Robert E. Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.* 55, 1 (1997), 119–139.

[13] Yaroslav Ganin and Victor Lempitsky. 2015. Unsupervised domain adaptation by backpropagation. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning (ICML'15)*, Francis Bach and David Blei (Eds.). Vol. 37. JMLR.org, 1180–1189.

[14] Norberto A. Goussies, Sebastián Ubalde, and Marta Mejail. 2014. Transfer learning decision forests for gesture recognition. *J. Mach. Learn. Res.* 15 (November 2014), 3667–3690.

[15] Isabelle Guyon and André Elisseeff. 2003. An introduction to variable and feature selection. *J. Mach. Learn. Res.* 3 (2003), 1157–1182.

[16] Mark A. Hall. 1999. *Correlation-Based Feature Selection for Machine Learning*. Ph.D. Dissertation. The University of Waikato.

[17] Wilhelmiina Hämäläinen and Mikko Vinni. 2006. In *Proceedings of the Intelligent Tutoring Systems: 8th International Conference (ITS'06)*. Springer, Berlin, 525–534. DOI: http://dx.doi.org/10.1007/11774303_52

[18] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H. Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, Miguel Ángel Rubio, Judy Sheard, Bronius Skupas, Jaime Spacco, Claudia Szabo, and Daniel Toll. 2015. Educational data mining and learning analytics in programming: Literature review and case studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports (ITICSE-WGR'15)*. ACM, New York, NY, 41–63. DOI: http://dx.doi.org/10.1145/2858796.2858798

[19] Matthew C. Jadud. 2006. Methods and tools for exploring novice compilation behaviour. In *Proceedings of the Second International Workshop on Computing Education Research (ICER'06)*. ACM, New York, NY, 73–84. DOI: http://dx.doi.org/10.1145/1151588.1151600

[20] Matthew C. Jadud and Brian Dorn. 2015. Aggregate compilation behavior: Findings and implications from 27,698 users. In *Proceedings of the 11th Annual International Conference on International Computing Education Research (ICER'15)*. ACM, New York, NY, 131–139. DOI: http://dx.doi.org/10.1145/2787622.2787718

[21] Soohyun Nam Liao, Daniel Zingaro, Michael A. Laurenzano, William G. Griswold, and Leo Porter. 2016. Lightweight, early identification of at-risk CS1 students. In *Proceedings of the 2016 ACM Conference on International Computing Education Research (ICER'16)*. ACM, New York, NY, 123–131. DOI: http://dx.doi.org/10.1145/2960310.2960315

[22] R. Lindén, T. Rajala, V. Karavirta, and M.-J. Laakso. 2016. Utilizing learning analytics for real time identification of students-at-risk on an introductory programming course. In *Proceedings of the Annual International Conference on Education and New Learning Technologies (EDULEARN'16)*. Barcelona, Spain.

[23] Ioanna Lykourentzou, Ioannis Giannoukos, Vassilis Nikolopoulos, George Mpardis, and Vassili Loumos. 2009. Dropout prediction in e-learning courses through the combination of machine learning techniques. *Comput. Educ.* 53, 3 (2009), 950–965.

[24] Merriam-Webster. 2016. Learning. Retrieved January 2016 from http://www.merriam-webster.com/dictionary/learning.

[25] Loris Nanni and Alessandra Lumini. 2009. Ensemble generation and feature selection for the identification of students with learning disabilities. *Exp. Systems with Appl.* 36, 2, Part 2 (2009), 3896–3900. DOI: http://dx.doi.org/10.1016/j.eswa.2008.02.065

[26] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.* 22, 10 (2010), 1345–1359.

[27] S. Gopal Krishna Patro and Kishore Kumar Sahu. 2015. Normalization: A preprocessing stage. In *International Advanced Research Journal in Science, Engineering and Technology* 2, 3 (2015). DOI: https://doi.org/10.17148/IARJSET.2015.2305

[28] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in python. *J. Mach. Lear. Res.* 12 (November 2011), 2825–2830.

[29] Andrew Petersen, Jaime Spacco, and Arto Vihavainen. 2015. An exploration of error quotient in multiple contexts. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research (Koli Calling'15)*. ACM, New York, NY, 77–86. DOI: http://dx.doi.org/10.1145/2828959.2828966

[30] Chris Piech, Mehran Sahami, Daphne Koller, Steve Cooper, and Paulo Blikstein. 2012. Modeling how students learn to program. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*. ACM, 153–160.

[31] Leo Porter, Daniel Zingaro, and Raymond Lister. 2014. Predicting student success using fine grain clicker data. In *Proceedings of the 10th Annual Conference on International Computing Education Research (ICER'14)*. ACM, New York, NY, 51–58. DOI: http://dx.doi.org/10.1145/2632320.2632354

[32] Nathan Rountree, Janet Rountree, Anthony Robins, and Robert Hannah. 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)*. ACM, New York, NY, 101–104. DOI: http://dx.doi.org/10.1145/1044550.1041669

[33] Robert E. Schapire. 2013. Explaining adaboost. In *Empirical Inference*. Springer, 37–52.

[34] Simon, Sally Fincher, Anthony Robins, Bob Baker, Ilona Box, Quintin Cutts, Michael de Raadt, Patricia Haden, John Hamer, Margaret Hamilton, Raymond Lister, Marian Petre, Ken Sutton, Denise Tolhurst, and Jodi Tutty. 2006. Predictors of success in a first programming course. In *Proceedings of the 8th Australasian Conference on Computing Education (ACE'06)*, Vol. 52. Australian Computer Society, Inc., Darlinghurst, Australia, 189–196.

[35] Masamitsu Tsuchiya, R. Yumiba, Y. Yamauchi, T. Yamashita, and H. Fujiyoshi. 2015. Transfer forest based on covariate shift. In *Proceedings of the IAPR Asian Conference on Patten Recognition (ACPR'15)*, Vol. 2. 4.

[36] Arto Vihavainen, Matti Luukkainen, and Jaakko Kurhila. 2012. Multi-faceted support for MOOC in programming. In *Proceedings of the 13th Annual Conference on Information Technology Education (SIGITE'12)*. ACM, New York, NY, USA, 171–176. DOI: http://dx.doi.org/10.1145/2380552.2380603

[37] Arto Vihavainen, Thomas Vikberg, Matti Luukkainen, and Martin Pärtel. 2013. Scaffolding students' learning using test my code. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 117–122.

[38] Christopher Watson, Frederick W. B. Li, and Jamie L. Godwin. 2014. No tests required: Comparing traditional and dynamic predictors of programming success. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE'14)*. ACM, New York, NY, 469–474. DOI: http://dx.doi.org/10.1145/2538862.2538930

[39] Ian H. Witten, Eibe Frank, and Mark A. Hall. 2011. *Data Mining: Practical Machine Learning Tools and Techniques* (3rd ed.). Morgan Kaufmann, San Francisco, CA.

[40] Tung-Kuang Wu, Shian-Chang Huang, and Ying-Ru Meng. 2008. Evaluation of ANN and SVM classifiers as predictors to the diagnosis of students with learning disabilities. *Exp. Syst. Appl.* 34, 3 (Apr. 2008), 1846–1856. DOI: http://dx.doi.org/10.1016/j.eswa.2007.02.026

[41] Yi Yao and Gianfranco Doretto. 2010. Boosting for transfer learning with multiple sources. In *Proceedings of the 2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'10)*. IEEE, 1855–1862.

[42] Michael Yudelson, Steve Fancsali, Steve Ritter, Susan Berman, Tristan Nixon, and Ambarish Joshi. 2014. Better data beats big data. In *Proceedings of the 7th International Conference on Educational Data Mining (EDM'14)*.

Received September 2016; revised October 2017; accepted October 2017