Evaluating Neural Networks as a Method for Identifying Students in Need of Assistance

Anon
...
...
...

Anon
...
...
...

Anon
...
...
...

ABSTRACT

Course instructors need to be able to students in need of assistance as early in the course as possible. Recent work has suggested that machine learning approaches applied to snapshots of small programming exercises may be an effective solution to this problem. However, these results have been obtained using data from a single institution, and prior work using features extracted from student code has been highly sensitive to differences in context.

This work provides two contributions: first, a partial reproduction of previously published results, but in a different context, and second, an exploration of the efficacy of neural networks in solving this problem. Our findings confirm the importance of two features (the number of steps required to solve a problem and the correctness of key problems), indicate that machine learning techniques are relatively stable across contexts (both across terms in a single course and across courses), and suggest that neural network based approaches are as effective as the best Bayesian and decision tree methods. Furthermore, neural networks can be tuned to be reliably pessimistic, so they may serve a complementary role in solving the problem of identifying students who need assistance.

Keywords

introductory programming; CS1; at-risk students; source code snapshot analysis; educational data mining; learning analytics; replication; reproduction

1. INTRODUCTION

The drop and failure rate of computer programming courses and, in particular, the first programming course (CS1) is high – in the neighborhood of 30% [6]. The decision to drop is complex and is typically the result of a combination of factors [19]. Common factors include the perceived difficulty of the course and lack of support when a student reaches a critical point [1].

... '17 ...

ACM ISBN ...-.-....-....-./../...

DOI: http://dx.doi.org/10.1145/------.-------

The experience of receiving a bad grade on a major component of the course is often the final event that triggers a decision to drop a course [1]. As a result, it is important to provide quality formative feedback as early as possible [33] and, ideally, to identify and to provide targeted support to students in need of assistance early in the course – before a summative assessment is completed.

In this paper, we focus on the problem of identifying students in need of assistance as early in the course as possible. The increasing use of online programming exercises that are assessed in real time provides an opportunity to develop heuristics – learning analytics [8] – that can identify students while the course is in progress. While this topic has been well studied, from Jadud's error quotient [17] to recent work on repeated error density [5], our focus is on prediction using data available either before or early in the course: demographic information and submissions to small programming exercises in the first third of the teaching term.

This work builds directly on recent work that showed that machine learning (ML) methods are effective at predicting student performance, with accuracy in the range of 80-90% [4]. Furthermore, they demonstrated that ML methods are fairly stable, as models generated from data from one term can successfully predict student performance in another term, albeit with lower accuracy (70-80%). However, their approach has only been tested in one course. Recent work has shown that established methods in this problem space are sensitive to changes in context [25], and a recent ITiCSE working group has called for additional work investigating the issue of sensitivity to context changes [16].

Therefore, this paper provides two contributions: first, a partial reproduction of Ahadi et al's earlier results, in a different context, and second, extension of the work using a new type of model (neural networks). The paper is structured around the following three research questions:

> **RQ1** Can the results of Ahadi et al [4] be reproduced in a different context?
>
> **RQ2** Are neural network (NN) models appropriate for the task of identifying students in need of assistance?
>
> **RQ3** Are some of the models explored less sensitive to varying contexts?

2. RELATED WORK

The goal of this work is to explore machine learning methods for identifying students at risk of dropping or failing the course – students in need of assistance. As discussed in

Ihantola et al.'s 2015 ITiCSE working group report on educational data mining, this area is of particular interest to the community [16], and we recommend their work for an overview of educational data mining literature published between 2005 and 2015. In this section, we focus on two particular topics: the use of repeated errors to identify students at risk of failure, and the application of machine learning techniques to model student behaviour.

2.1 Repeated Errors and Context Sensitivity

Recent work in this area has been based on analysis of compilation behaviour and error frequency. Jadud's work on an error quotient (EQ), which demonstrated that student compilation behavior can be used effectively to model students and to predict their performance in a CS1 context, focused on the importance of repeated errors in a sequence of compilations [17]. Later work on EQ added other factors, such as time between compilations and attributes of programmer behaviour [28, 27]. The work culminated in an unsuccessful effort to predict student midterm scores [29].

Perceived shortcomings in EQ sparked a number of later efforts to better model programmer behaviour. Watson and Godwin incorporated the time spent correcting errors into the model, improving performance [32, 31], and Carter et al. incorporated a sense of semantic correctness to create the Normalized Programming State Model (NPSM) [9]. All of these efforts compared their results to EQ. They found that it did not work as well in their contexts and that their new models were more effective.

Petersen et al. focused on the issue of performance across contexts [25]. They evaluated EQ using data from four different courses which varied in terms of language, submission tool, assignment format, and institution. They found that, while repeated errors remained a significant predictor of performance, the performance of EQ varied across contexts and that the most effective weighting of the parameters in the EQ algorithm was highly volatile. This work suggests that metrics need to be tuned to apply to different contexts – or that more stable methods are necessary. In response, Becker proposed a new error-based metric, Repeated Error Density (RED), that they found to be less sensitive to changes in context [5].

2.2 Machine Learning Approaches

Machine learning approaches have been extensively used to model student behaviour and performance in computer programming activities. Many of these approaches have been based on *a priori* factors. For example, Bergin et al. validated a model that used past performance in math domains [7], and El Gamal et al. attempted to isolate variables and extract rules for predicting student performance in programming using features collected prior to the course [10]. Osmanbegovic et al. investigated a range of factors, including socio-demographic variables, results from high school and from a university entrance exam, and attitudes towards studying, across several different models (C4.5, multilayer perception, and naive Bayesian) [24]. In their context, the naive Bayesian model had the highest accuracy.

Another cluster of recent research has examined the use of social interaction and participation data generated during the course, with mixed results. Fire et al. analyzed social network graphs and found the strongest predictor from the graph to be a students "best" or nearest friend [13]. Elbadrawy et al. found online forum and discussion activity to be of low importance, while metrics related to accessing course material were of high importance [11], and Luo et al. [21] found comments by students on the course and labs to be a strong predictor of course grades.

Finally, other efforts have attempted to use factors derived from attempts to solve exercises, like repeated errors and the number of steps required to obtain a correct solution, in models. Mullier classified students according to their responses to tutorial questions [23], and Minaei-Bidgoli et al. investigated the usefulness of correctness, time spent per problem, and number of attempts as inputs to a genetic algorithm approach [22]. Most recently, Liao et al. used ML techniques to model students using in-class clicker data and obtained up to 70% accuracy categorizing students as at-risk as early as week 3 of a 12 week course [20], and Ahadi et al. proposed using machine learning classifiers on program snapshot data like that used for EQ, NPSM, and RED, and found a random forest model to be more effective than Watwin score in their particular experimental context [4].

One goal of our work is to evaluate the context sensitivity of this recent work, as researchers did to EQ, by applying the same techniques used by Ahadi et al. in a new context. In addition, we also seek to extend the work by exploring the use of neural network models that might prove to be either more accurate or more stable. There is precedent for doing so: random forest and neural network models have been found to outperform one another across various domains, depending on features of the problem [18, 12] Richmond et al. likened the tradeoff between these approaches as one of speed (random forest) vs performance (neural network) [26].

3. METHODOLOGY

This project involves the production and evaluation of various machine learning models to classify students by expected performance. The neural networks were built using TensorFlow [3], and all other models were constructed using Weka [14].

3.1 Data Sources

Two sources of data were obtained to perform the comparison across contexts. The first source of data is a Python-based CS1 course at a large, research-intensive North American university (Uni-A). Course topics include variables; conditionals; loops; functions; built-in types (numeric, string, lists, and dictionaries); basic I/O; and basic algorithms. Data was collected in Fall 2015 (F15) and Winter 2016 (W16), with 897 students providing consent for the data to be analyzed in F15 and 519 providing consent in W16.

The dataset consists of course marks and snapshots from 30 small coding exercises presented in an online programming environment [2]. These exercises are completed individually, in an unsupervised environment, with support available on the discussion board and in a CS help room staffed by teaching assistants. Each snapshot represents a "submission." When a student submits the code, it is executed and feedback, in the form of an interpreter error or results of tests, is provided. A similar set of exercises was used in both the F15 and W16 terms, allowing for comparison across terms.

The second source of data is from a Java-based CS1 course at a research-intensive European university (Uni-B). Course

topics are comparable to Uni-A but include more object oriented material: variables, methods, basic objects, conditionals, loops, lists, arrays, basic I/O, and basic algorithms. Data was collected from 263 students in Fall 2014 (F14).

Similar to the Uni-A dataset, the Uni-B set contains course marks and snapshots from 116 small coding assignments presented in an online environment [30]. The exercises are completed individually, in an unsupervised environment, with assistance available in open computer labs staffed by teaching assistants. Finer grained snapshots are available – including "save," "compile," "run," and "test" events – where the "run" and "test" events are roughly comparable to the "submission" events available in the Uni-A dataset.

3.2 Data Processing

The student code behavior snapshots were first translated into the Progsnap format [15]. Progsnap is a standardized format for code snapshot data. Features were extracted from the Progsnap data files and saved in Weka's arff format. The default parameters were used for all Weka models. Datasets were validated on a randomly sampled 70% train, 30% test split and results were averaged over 50 trials.

To generate neural networks, the data was vectorized but not normalized. The model used has the following characteristics: rectified linear units in the internal layers, a cross entropy cost function, 200 nodes per layer, fully connected layers, and randomly initialized parameters. We evaluated neural networks with 1, 2, and 3 hidden layers; the model with 1 layer had the best performance. Recurrent and convolutional neural networks were not explored because the low feature dimensionality of the data made them unsuitable. After filtration, there were 23 features per student.

3.2.1 Features

To compare with Ahadi et al [4], two features were extracted: the number of "Steps" required to complete an assignment and "Correctness." "Steps" were calculated as the number of submission events recorded for each coding exercise. "Correctness" is the fraction of tests passed for each lab exercise. (If a student's final submission passes 3 of 4 tests, the correctness is 0.75.)

Several additional features were explored: "time spent"; "finishing time"; and "pass", "fail", and "error" counts. The "time spent" on an exercise was defined to be the difference between the timestamps of the initial and final submission of an exercise. The "finishing time" is the timestamp of the final submission. The counts indicate the number of submissions for an exercise that returned a particular status: "pass," when the submission passes all tests; "fail," when the execution runs without error but fails at least one test; and "error," when the execution generates an error.

3.2.2 Null Values

Some fields in the data sets contained null values. For example, a student may fail to attempt a particular exercise, resulting in a null value in a feature representing the number of attempts required to successfully complete an exercise. Weka allows null values to be represented in two ways (-1 or '?'), but the two types of null values are handled differently by different algorithms. To handle nulls consistently with TensorFlow, we chose to represent nulls using '-1.'

Null course grades are handled differently. A null course grade indicates that a student dropped the course. Since the objective of the study is to identify students in need of assistance, we made the assumption that students who dropped were in need of assistance and interpreted a null grade as a failing grade.

Semester	Mean	Median	Std Dev
Uni-A F15 Steps	12.7	6.0	21.2
Uni-A W16 Steps	9.3	3.0	20.0
Uni-A F15 Correctness	0.3	1.0	0.9
Uni-A W16 Correctness	0.7	1.0	0.7

Table 1: Average Steps and Correctness at Uni-A

3.2.3 Bias in the Dataset

The models were built to classify students as "passing" (final grade above 50%) or "failing" (final grade below 50% or null). The raw data is heavily biased toward passing (fortunately!), with roughly 75% of students earning a passing grade in the course. This imbalance could cause models trained on the data to be similarly biased, encouraging them to naively classify all students as passes. To prevent this, the training data is balanced by randomly selecting a sample of students from the larger category so that the two categories have the same number of members. A new random sample is selected for each trial that is run.

4. RESULTS

In the tables presented in this section, all data reported is the average over 50 trials for each model. "accuracy" is the overall accuracy of the model: the number of students who were correctly classified. "fail acc" and "pass acc" are the percentages of students who were correctly classified as "fail" and "pass," respectively.

While multiple features were explored (as described in Section 3.2.1), only "Steps" and "Correctness" were found to contribute, so all of the tables in this section only report results from models based on those features. Table 1 describes the average number of steps and correctness of the exercise in the Uni-A datasets. While the specific values are not relevant, the variance between terms is of particular interest, as it signals significant differences even between courses taught in the same context.

Table 2 describes the classification accuracy of the various models when data from the entire term is available. This represents an unrealistic scenario, as final marks are being predicted after the entire term is complete, but it provides an upper bound on the obtainable accuracy. The models are generally comparable: overall accuracy ranges from 80-85 across both contexts. The Naive Bayesian and Random Forest models perform well, as does the best Neural Network (NN) model. However, there is a difference between the NN model and all others: it is distinctly pessimistic, accurately classifying more failing students but mis-categorizing relatively more passing students.

Table 3 describes results from more realistic scenarios: prediction of a student's final classification early in the term. The first set of data, from Uni-A, illustrates a scenario when prediction is performed one-third of the way through the term (at week 4 in our 12 week term), shortly before the first midterm. In comparison to Table 2, overall accuracy is noticeably lower: by 13%, on average, for all models. However, the accuracy of categorizing failing students remains

Model	Uni-A (F15): all data			Uni-B (F14): all data		
	Accuracy	Fail Acc	Pass Acc	Accuracy	Fail Acc	Pass Acc
bayes.NaiveBayes	**85.29**	80.56	89.90	82.10	73.54	**90.30**
Neural Network	84.91	**89.48**	81.17	83.20	**86.96**	79.96
trees.RandomForest	84.77	78.73	90.66	**84.48**	81.91	86.94
rules.DecisionTable	81.25	77.77	84.63	80.00	80.74	79.29
trees.DecisionStump	82.80	72.62	**92.70**	80.10	80.54	79.66
rules.PART	81.51	81.08	81.92	80.38	79.57	81.16
trees.J48	81.33	81.78	80.90	82.19	82.10	82.28

Table 2: Classification accuracy (using steps and correctness) using all available data. Top performers bolded.

Model	Uni-A (F15): first 4 weeks			Uni-A (F15): first week			Uni-B (F14): first week		
	Accuracy	Fail Acc	Pass Acc	Accuracy	Fail Acc	Pass Acc	Accuracy	Fail Acc	Pass Acc
bayes.NaiveBayes	**72.90**	61.01	85.00	61.29	36.56	87.08	**70.10**	51.97	**87.08**
Neural Network	72.04	**85.50**	59.74	**62.74**	**85.02**	40.87	66.40	**80.62**	50.21
trees.RandomForest	71.10	66.47	75.80	60.95	56.95	65.11	68.29	60.24	75.83
rules.DecisionTable	68.04	58.19	78.06	57.46	32.10	83.92	67.90	64.76	70.85
trees.DecisionStump	67.66	48.21	**87.42**	54.28	13.48	**96.84**	65.05	49.80	79.34
rules.PART	66.62	62.03	71.29	61.16	44.14	78.91	61.62	61.22	61.99
trees.J48	67.83	64.76	70.95	60.60	46.50	75.31	62.19	60.24	64.02

Table 3: Classification accuracy (using steps and correctness) using early-term data. Top performers bolded.

Model	Uni-A (F15 -> W16): first 4 weeks		
	Accuracy	Fail Acc	Pass Acc
bayes.NaiveBayes	70.33	63.81	76.59
Neural Network	70.43	**77.05**	63.52
trees.RandomForest	70.14	71.69	68.66
rules.DecisionTable	69.90	68.39	71.36
trees.DecisionStump	**71.81**	60.89	**82.28**
rules.PART	67.05	69.36	64.83
trees.J48	67.00	72.96	61.29

Table 4: Classification accuracy of early prediction (using steps and correctness) across terms. Top performers bolded.

relatively high for neural net models: only 4% is lost.

The next two sets of data, from both Uni-A and Uni-B, illustrate a scenario when prediction is performed after the first week of the term: in essence, as early as possible, if data from within the course is to be used. For Uni-A, an additional 5-10% accuracy is lost relative to the four week scenario, but again, the neural net model successfully classifies a significant fraction of failing students, albeit at the cost of even more false positives (misclassified passing students).

Finally, Table 4 presents the result of using a model trained on the previous term to classify students in a new term. This is the most realistic scenario, where historical data is being used to classify students early in a term. Comparing to the first dataset in Table 3, we see that overall accuracy drops slightly for most models but remains comparable. The DecisionStump model is an anomaly, as performance increases almost 4%. Otherwise, the relative performance of the models remains the same, with the Bayesian, random forest, and neural net models performing the best.

4.1 RQ1: Are the results of Ahadi et al. reproducible?

In large part, yes. We did not examine the demographic features explored by Ahadi et al. (age, major, gender, etc.) [4], since only major was found to be relevant and that feature does not apply directly at Uni-A. At Uni-A, the students are in the first term of their first year, so they have no previous academic history and will not be accepted into a major until the beginning of their second year. However, both of Ahadi et al.'s snapshot-based features – steps and correctness – were effective, and none of the other snapshot-based features we explored, including time spent and more detailed data on the results of steps, contributed to more effective models.

The accuracy and relative performance of our models are also comparable to the results observed by Ahadi et al [4]. With the addition of the demographic features mentioned earlier, Ahadi et al. observed accuracies ranging from 73-90% when predicting the final course grade, in comparison to our 80-85%. Furthermore, the most effective classifier they observed, random forest, is among the higher performing classifiers we observed. However, in our context, the Bayesian classifiers performed relatively better, and our observations suggest that it may not be possible to identify a single best classifier across contexts.

One significant difference between our Uni-A data and the data used by Ahadi et al. is the number of exercises included in the set. Our data set has less than half the number of exercises. It may be possible that we are not observing the variation between classifiers seen by Ahadi et al. because there is insufficient data for the classifiers to differentiate themselves.

4.2 RQ2: Are neural networks effective at identifying students in need of assistance?

Yes. The neural network model achieved performance comparable to the best Bayesian, rule learner, and decision tree methods evaluated. The neural net was evaluated with 1,2 and 3 internal layers. Additional layers were not found to change performance. The most effective model was that with the least layers, which is equivalent to a simple regression model. This is likely due to there being little to be

gained from inter feature dependencies.

The particular model identified may have an advantage in practice. Compared to the other methods evaluated, the neural net model was consistently pessimistic. Although it achieved comparable overall accuracy when compared to the other models, it favored fail classifications while the Weka models favored pass classifications. In practice, this means that instructors utilizing these models to identify students in need of assistance will successfully identify a larger fraction of the truly at-risk students (more true positives and less false negatives) at the cost of identifying a relatively larger number of students who will eventually pass the class (more false positives). Furthermore, this pessimism is relatively stable: the neural net model continues to identify truly at-risk students, even early in the course or across terms. This pessimism is potentially advantageous, if the goal of the instructor is to identify as many at-risk students as possible.

4.3 RQ3: Are some models less sensitive to changes in context?

The differences are not very large, but almost all of the methods observed were fairly stable. Since the features in the two sets are not comparable, we cannot use models trained in one context at the other. Instead, we can only compare the relative performance of the models in both contexts. Overall, the neural net, naive Bayes, and random forest models generally have a higher accuracy than the other models, with the neural net model having the highest overall average ranking. However, it also had the highest overall fail accuracy by a wide margin, and that result was clearly stable across contexts.

We also used data from one semester (F15) as training data and data from the other semester (W16) as test data (Table 4). The result in this case is very positive: while most models trained in one term were less accurate when applied to the next term, the penalty was not large for any of the models. Furthermore, to make this comparison, only exercises common to both data sets could be retained (so that the resulting model would not be based on irrelevant features). In our case, this reduced the number of features available, so the model in Table 4 was produced with less data. In a situation where all of the features are preserved across terms, the relative performance loss may be even less.

5. CONCLUSIONS

This study provides welcome news to researchers – and instructors – interested in identifying at-risk students early in the term. We explored a range of Bayesian, decision tree, rule learner, and neural network models and found that the performance of most of the models was fairly stable across contexts and, more importantly, across terms in the same course. We also confirmed that features identified as important in previous work [4] – the number of "steps" required to complete exercises and the "correctness" of particular exercises – remained the relevant features in a new context.

This work is the first that we are aware of to use neural networks to identify at-risk students from code snapshots. We found that shallow (1-level) neural networks performed as well as more complex networks and that neural network approaches performed as well as the best Bayesian and decision tree models we explored. In addition, the neural network produced had the potential benefit of being pessimistic – reducing the number of at-risk students "missed" – and this pessimism was a stable feature across contexts and terms.

We anticipate several avenues for future work. First, since many of the models achieved similar performance, there may not be much to be gained from particular models. Instead, future gains in accuracy may be obtained by extracting more effective features or simply from using more data (such as feeding the raw code itself into the model). Second, since some models are more pessimistic and others are more optimistic, it may be possible to use a hybrid approach to place students on a continuum from "definitely at risk" to "definitely passing."

6. ACKNOWLEDGEMENTS

We gratefully acknowledge the support of [redacted], who provided permission to use data from Uni-B.

7. REFERENCES

[1] ... Anonymized for review. In *Proceedings of ...*, 20XX.

[2] ... Anonymized for review. In *Proceeding of ...*, 20XX.

[3] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] A. Ahadi, R. Lister, H. Haapala, and A. Vihavainen. Exploring machine learning methods to automatically identify students in need of assistance. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*, pages 121–130. ACM, 2015.

[5] B. A. Becker. 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*, pages 296–301. ACM, 2016.

[6] J. Bennedsen and M. E. Caspersen. Failure rates in introductory programming. *ACM SIGCSE Bulletin*, 39(2):32–36, 2007.

[7] S. Bergin and R. Reilly. Predicting introductory programming performance: A multi-institutional multivariate study. *Computer Science Education*, 16(4):303–323, 2006.

[8] J. Bichsel. Analytics in higher education: Benefits, barriers, progress, and recommendations, 2012. Accessed August 24, 2016.

[9] A. S. Carter, C. D. Hundhausen, and O. Adesope. The normalized programming state model: Predicting student performance in computing courses based on programming behavior. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*, pages 141–150. ACM, 2015.

[10] A. El Gamal. An educational data mining model for predicting student performance in programming

course. *International Journal of Computer Applications*, 70(17), 2013.

[11] A. Elbadrawy, S. Studham, and G. Karypis. Personalized multi-regression models for predicting students performance in course activities. *UMN CS*, pages 14–011, 2014.

[12] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems. *J. Mach. Learn. Res*, 15(1):3133–3181, 2014.

[13] M. Fire, G. Katz, Y. Elovici, B. Shapira, and L. Rokach. Predicting student exam's scores by analyzing social network data. In *International Conference on Active Media Technology*, pages 584–595. Springer, 2012.

[14] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[15] D. Hovemeyer and J. Spacco. Progsnap specification. Accessed August 26, 2016.

[16] P. Ihantola, A. Vihavainen, A. Ahadi, M. Butler, J. Börstler, S. H. Edwards, E. Isohanni, A. Korhonen, A. Petersen, K. Rivers, M. A. Rubio, J. Sheard, B. Skupas, J. Spacco, C. Szabo, and D. Toll. 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, pages 41–63, New York, NY, USA, 2015. ACM.

[17] M. C. Jadud. Methods and tools for exploring novice compilation behaviour. In *Proceedings of the Second International Workshop on Computing Education Research*, ICER '06, pages 73–84, New York, NY, USA, 2006. ACM.

[18] N. Jaques and J. Nutini. A comparison of random forests and dropout nets for sign language recognition with the kinect.

[19] P. Kinnunen and L. Malmi. Why students drop out CS1 course? In *Proceedings of the Second International Workshop on Computing Education Research*, pages 97–108, 2006.

[20] S. N. Liao, D. Zingaro, M. A. Laurenzano, W. G. Griswold, and L. Porter. Lightweight, early identification of at-risk cs1 students. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ICER '16, pages 123–131, New York, NY, USA, 2016. ACM.

[21] J. Luo, S. E. Sorour, K. Goda, and T. Mine. Predicting student grade based on free-style comments using word2vec and ann by considering prediction results obtained in consecutive lessons. *International Educational Data Mining Society*, 2015.

[22] B. Minaei-Bidgoli and W. F. Punch. Using genetic algorithms for data mining optimization in an educational web-based system. In *Genetic and evolutionary computation conference*, pages 2252–2263. Springer, 2003.

[23] D. Mullier, D. Moore, and D. Hobbs. A neural-network system for automatically assessing students. In *World conference on educational multimedia, hypermedia and telecommunications*, pages 1366–1371, 2001.

[24] E. Osmanbegović and M. Suljić. Data mining approach for predicting student performance. *Economic Review*, 10(1), 2012.

[25] A. Petersen, J. Spacco, and A. Vihavainen. An exploration of error quotient in multiple contexts. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, pages 77–86. ACM, 2015.

[26] D. L. Richmond, D. Kainmueller, M. Yang, E. W. Myers, and C. Rother. Mapping stacked decision forests to deep and sparse convolutional neural networks for semantic segmentation. *arXiv preprint arXiv:1507.07583*, 2015.

[27] M. M. Rodrigo, E. Tabanao, M. B. Lahoz, and M. C. Jadud. Analyzing online protocols to characterize novice java programmers. *Philippine Journal of Science*, 138(2):177–190, 2009.

[28] M. M. T. Rodrigo, R. S. Baker, M. C. Jadud, A. C. M. Amarra, T. Dy, M. B. V. Espejo-Lahoz, S. A. L. Lim, S. A. Pascua, J. O. Sugay, and E. S. Tabanao. Affective and behavioral predictors of novice programmer achievement. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '09, pages 156–160, New York, NY, USA, 2009. ACM.

[29] E. S. Tabanao, Ma, and M. C. Jadud. Predicting At-risk Novice Java Programmers Through the Analysis of Online Protocols. In *Proceedings of the Seventh International Workshop on Computing Education Research*, ICER '11, pages 85–92, New York, NY, USA, 2011. ACM.

[30] A. Vihavainen, T. Vikberg, M. Luukkainen, and M. Pärtel. Scaffolding students' learning using test my code. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, pages 117–122. ACM, 2013.

[31] C. Watson and F. W. Li. Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, pages 39–44. ACM, 2014.

[32] C. Watson, F. W. Li, and J. Godwin. Predicting performance in an introductory programming course by logging and analyzing student programming behavior. In *In Advanced Learning Technologies*, pages 31–323. IEEE, 2013.

[33] M. Yorke. Formative assessment and its relevance to retention. *Higher Education Research & Development*, 20(2):115–126, 2001.