Using Machine Learning Techniques to Predict Introductory Programming Performance

Susan Bergin[1], Aidan Mooney[2], John Ghent[3] and Keith Quille[4]

[1, 2, 4] Department of Computer Science, Maynooth University, Maynooth, Co. Kildare, Ireland

[3] Sytorus, The Capel Building, Dublin 7, Ireland

susan.bergin@nuim.ie, [2]aidan.mooney@nuim.ie, [3]john.ghent@sytorus.com, [4]keith.quille.2014@mumail.ie

ABSTRACT

Learning to program is difficult and can result in high drop out and failure rates. Numerous research studies have attempted to determine the factors that influence programming success and to develop suitable prediction models. The models built tend to be statistical, with linear regression the most common technique used. Over a three year period a multi-institutional, multivariate study was performed to determine factors that influence programming success. In this paper an investigation of six machine learning algorithms for predicting programming success, using the pre-determined factors, is described. Naïve Bayes was found to have the highest prediction accuracy. However, no significant statistical differences were found between the accuracy of this algorithm and logistic regression, SMO (support vector machine), back propagation (artificial neural network) and C4.5 (decision tree). The paper concludes with a recent epilogue study that re-validates the factors and the performance of the naïve Bayes model.

Keywords: *Learning to Program, Programming Predictors, Machine Learning, Naïve Bayes.*

1. INTRODUCTION

It is well established in the Computer Science Education (CSEd) community that students have difficulty with learning to program and this can result in high drop-out and failure rates [4, 13, 22]. Identifying struggling students at an early stage is not easy as introductory programming modules often have a high student to lecturer ratio (100:1 or greater) and early assessment may not be a reliable indicator of overall performance. Early assessment is also troublesome as carrying out authentic assessment with manual correction can be a slow process. By the time feedback is available to students it may be too late for students to withdraw from the course or for instructors to implement interventions to prevent struggling students from failing. This is a cause of great concern for educators and has led to a body of research in the area [1, 2, 3, 4, 14, 22]. Coupled with this, it can be difficult to interpret the findings of previous studies as they tend to have very specific parameters, for example they use different programming languages, have different educational settings, different assessment structures and varying student profiles. This can bias the findings, creating undesired weightings on certain parameters and in turn limiting the true universality of the study. In addition, many of the studies are based on a small sample size and no validation studies have been carried out to verify the findings. These problems are compounded by the fact that some studies provide only anecdotal evidence that lacks scientific rigour whilst the best studies have only ever attempted to use statistical techniques, such as correlation and regression to predict performance. While these techniques are well regarded, they are restricted by underlying assumptions and thus may not yield the most powerful models.

A review of the literature indicates that no longitudinal study had attempted to evaluate the use of different machine learning algorithms to predict introductory programming performance. For such a study to have considerable research value it would need to (1) be multi-institutional to promote the generalizability of the predictive models developed, (2) use factors that have been determined as part of longitudinal research using replicated studies to verify the most effective factors that are truly universal and timeless, resilient to biasing even when student profiles or landscape vary and, (3) develop machine learning models that are accessible, understandable, and usable by the CSEd community. Research that satisfies this set of criteria is presented in this paper. A longitudinal study to determine factors that influence programming success is described. Next, six different machine learning algorithms for predicting programming performance are presented. Our rationale for selecting the algorithms is discussed and a detailed evaluation on our results is presented. The paper concludes with a description of a very recent study that used the approach described here on a new cohort of students, demonstrating the same high level of accuracy and providing further validating evidence on the value of the model developed here.

2. FACTORS THAT IMPACT PROGRAMMING SUCCESS

For over ten years, the authors have carried out numerous studies investigating early identifiable factors that could influence programming performance. Ethical approval for this work was attained and the agreed protocol was carefully adhered to during all stages of the research process, including, the voluntary nature of participation, the right to withdraw at any stage and the steps taken to protect and anonymize participant data. The study reported on in this paper involved four third-level institutions (post high-school) in the Republic of Ireland. These institutions varied significantly in classification (University, College and Community College), academic entry requirements and student demographic. In total 123 students enrolled on introductory programming modules voluntarily consented to participate in this study. The overall aim of each module at each institution was to provide students with introductory programming skills and the content of each module was similar.

The study examined 25 factors that could influence introductory programming performance. Each of these factors can be identified at the start of a module when students have had minimal exposure to programming concepts. This is important so that early interventions can be put in place. The factors examined can be broadly grouped into three categories as outlined in Table 1. Detailed data preprocessing procedures were implemented prior to data analysis. Data pre-processing involved several steps including data screening; tests of uni-dimensionality; missing data analysis; and tests of sample representativeness. In addition Principal Component Analysis (PCA) was implemented to reduce the dimensionality of the dataset. PCA takes a set of data points and constructs a lower dimensional linear subspace that maximizes the variability of the training set. PCA essentially performs an orthonormal transformation on the input data such that the variance of the input data is accurately captured using only a few of the resulting basis vectors. These basis vectors are calculated in such a way that the squared difference between the input data and the data as reconstructed from the principal components is minimized. Components that satisfied the Kaiser criterion (eigenvalues greater than 1.0) were retained for use in the predictive model [19]. Numerous models were developed and three significant factors emerged: final Mathematics examination result at second level, number of hours playing computer games while taking the programming course and the first principal component derived from the programming self-esteem instrument. A detailed review of the study and the significant predictors found is provided in [4].

Table 1: Predictors of Programming Performance

Category	Brief Description
Background factors	Previous academic experience, for example, Mathematics, Science and Language grades achieved in second level exit examinations; previous experience of computer applications, game playing, internet usage and programming; number of hours spent studying and working at a part-time job etc.
Perceived comfort level factors at the start of the module	Assessed by three instruments: (1) Nine questions on comfort-level taken from a study by [5] that examined a student's perception of their level of understanding compared to the rest of the class, their ease at asking and answering programming questions, their general understanding of programming concepts and their ability to design and complete assignments, (2) a programming-self esteem questionnaire, developed and validated by the primary author based on the Rosenberg Self-Esteem (RSE) questionnaire [15], and (3) a shortened version of the Computer Programming Self-Efficacy Scale [14] which asked students to judge their capabilities in a wide range of programming tasks and situations.
Motivation and use of learning strategies	As measured by the Motivated Strategies for Learning Questionnaire (MSLQ). The MSLQ is a self-report instrument used to measure college students' motivation and use of learning strategies [11].

3. MACHINE LEARNING ALGORITHMS

This study utilizes significant predictors that emerged as a result of a longitudinal study carried out by the authors. The predictors are thus in contrast to previous related programming studies in that they are useful predictors at the very start of a module and do not require students to have experienced detailed aspects of the coursework, nor do they require the predictive models to include in-course examinations. An accurate computational model built using these attributes would be particularly useful, as it would facilitate the development of early interventions to assist struggling students. In so far as was possible, the authors sought to implement a blend of algorithms using diverse machine learning techniques to determine their effectiveness at predicting performance on an introductory programming module. It is important that the models developed can be interpreted and utilized by interested educators to predict incoming student performance and thus the machine learning algorithms selected should not require highly specialized knowledge. Thus, the goals of this paper are to: (1) determine the effectiveness of six machine learning algorithms for predicting introductory programming performance and (2) determine if in a brand new setting the algorithm detected as most effective can produce a similar level of performance.

3.1 Review of Classifiers

Learning to accurately classify is a common problem in machine learning and data analysis. Several different machine learning algorithms have been proposed and in this paper six different types of algorithms are evaluated, including, logistic regression, k-nearest neighbor, backpropagation, C4.5, naïve Bayes and support vector machines. Java implementations of these algorithms from the Waikato Environment for Knowledge Analysis, WEKA, as outlined in [20], were used in this study.

Logistic regression is a statistical technique to predict a discrete outcome, such as group membership from a set of variables. The dependent variable does not need to be linearly related to the independent variables, homoscedasticity is not required nor do the variables need to be normally distributed. The independent variables can be continuous, discrete or dichotomous. It is a particularly useful technique when there is a non-linear relationship between the dependent variable and one or more of the independent variables [19]. The standard representation for logistic regression is given by Equation 1:

$$P_i = \frac{1}{1+e^{-b_0+b_iX_i}} \tag{1}$$

K-Nearest Neighbor (KNN) is an instance-based learning technique. This type of learning is 'lazy' as it defers generalization until the classification stage. The nearest neighbor algorithm is based on the principal that the properties of any particular instance are likely to be similar to those instances within its neighborhood. Each new instance is compared with existing ones using a distance metric and the new instance is classified based on the majority class of the nearest K neighbors [10, 20].

Backpropagation is a learning algorithm that can be used to train multi-layer feedforward networks. In the backpropagation learning process one of the training instances is applied to the network, and the network produces some output based on the current state of its weights (initially the output will be random). This output is compared to the target output and an error signal is calculated. The total error, E, over all of the network output units is defined as:

$$E = \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd} - o_{kd})^2 \tag{2}$$

where D is the set of training examples, outputs is the set of output units in the network, t_{kd} and o_{kd} are the target and output values for the k^{th} output unit for training example d [10]. The error value is propagated backwards through the network, and changes are made to the weights in each layer. Weights can be updated after every input-output case and therefore no separate memory is required for the derivatives.

An alternative approach, which is used in this paper, is to accumulate $\frac{\partial E}{\partial w}$ over all of the input-output cases before changing the weights. Each weight is then changed by an amount proportional to the accumulated $\frac{\partial E}{\partial w}$ and to the learning rate η:

$$\Delta w = -\eta\frac{\partial E}{\partial w} \tag{3}$$

The whole process is repeated for each of the training instances and the cycle is repeated until the overall error value drops below a pre-determined threshold.

Naïve Bayes is a non-parametric probabilistic model based on an assumption of conditional independence among variable attributes. Although this assumption is often violated, naïve Bayes classifiers have been shown to work surprisingly well and have highly competitive prediction performance even when compared with some state-of-the art classifiers [9, 10]. A naïve Bayes classifier is denoted by Equation 4.

$$v_n b = \arg\max_{vj\in v} P(v_j)\prod_{i=1..k} P(a_i \mid v_j)$$

(4)

Decision Trees involve the recursive partitioning of a dataset. An attribute is selected to place at the root node and a branch is created for each possible value. This process is repeated recursively for each branch, using only those instances that reach the branch. If all instances at a node belong to the same class no further partitioning is performed. C4.5 is a popular decision tree algorithm, based on ID3 but contains several improvements, such as handling continuous attributes and measures for choosing an appropriate attribute selection scheme [16, 10, 20].

Support Vector Machines (SVMs) are a relatively new generation of learning system based on advances in statistical learning theory [17]. The principal idea behind linear SVMs is the optimal hyperplane. During the generation of a discriminant function, standard techniques such as the Perceptron will stop as soon as the last sample is classified without error. This provides a quick but potentially poor solution as it leaves the separation surface very close to the last sample classified. This will classify all the data in the training set correctly but may provide poor generalisation. To counteract this problem the linear SVM learning algorithm is modified so that the hyperplane is positioned in an optimal location between the two classes. To do this a conceptual margin is used. The margin is the perpendicular distance between the closest vector to the hyperplane and the hyperplane itself. The optimal hyperplane is the one that maximises the margin [8]. Suppose we have a dataset $(x_1,y_1), \ldots, (x_m,y_m) \in$ **X**×{±1} where **X** is some space from which the x_i have been sampled. The optimal hyperplane can be found by solving the dual form Lagrangian:

$$W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (x_i \bullet x_j) \tag{5}$$

which are subject to the constraints

$$\alpha_i \geq 0 \quad \forall_i \qquad \text{and} \qquad \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{6}$$

The solution to Equation 5 is a set of α values [8]. Further details of the construction of Equation 5 can be found in [17]. Although non-linear SVMs exist, given their increased complexity, only a linear SVM using Sequential Minimal Optimization (SMO) is implemented in this study [12].

4. MEASUREMENT TECHNIQUES

Three measurement techniques were employed in this study, specifically, overall classifier accuracy, sensitivity and specificity. The simplest form of evaluation is classification accuracy: the proportion of instances correctly predicted. Using Table 2 for illustration, the computation of this measure is given by Formula 7.

Table 2: Sample Confusion Matrix

		Predicted Class	
		Yes	***No***
Actual Class	***Yes***	TP	FN
	No	FP	TN
TP = True Positive, FP = False Positive			
TN= True Negative, FN = False Negative			

$$\frac{TP + TN}{TP + TN + FP + FN} \tag{7}$$

Sensitivity is a measure of the proportion of actual positive instances that are correctly classified, given by Formula 8.

$$\frac{TP}{TP + FN} \tag{8}$$

Specificity is the proportion of actual negative instances correctly classified, as illustrated by Formula 9.

$$\frac{TN}{TN + FP} \tag{9}$$

5. RESULTS

All algorithms were implemented using 10-times 10--fold stratified cross validation. This involves splitting the data into 10 parts, with each part representing the same proportion of each class. Each part is held out in turn and the learning scheme is trained on the remaining 9 parts, then the error rate is calculated on the holdout set. Thus the procedure is executed 10 times on different training sets. This whole procedure is repeated 9 more times and the results are averaged for the 100 testing datasets. The advantage of this method is that all instances can be used for training and testing thus it reducing bias in partitioning data and increasing overall confidence in the generalizability of the models. Accuracy, sensitivity and specificity measures for the algorithms are given in Table 3.

Based upon the accuracy measure, the most successful algorithms in descending order are naïve Bayes, SMO, logistic regression, backpropagation, C4.5 and 3-NN. Although, overall accuracy is important in this study the sensitivity measure is also valuable. While ideally, we would like to predict the performance of all students accurately; misclassifying strong students as weak is far less detrimental than misclassifying weak students as strong. In the latter case suitable interventions may not be put in place to prevent weak students from failing but providing good students with extra attention unnecessarily is at worst a waste of resources. In order of importance based on the sensitivity measure, the algorithms of choice are naïve Bayes, SMO, C4.5, logistic regression, backpropagation and 3-NN. In terms of the specificity measure, although not as critical a measure in this study, the best algorithms, in order, on this measure were naïve Bayes, logistic regression, SMO, backpropagation, 3-NN and C4.5.

Table 3: Comparison of Classifier Performance

Algorithm	Accuracy	Sensitivity	Specificity
Naïve Bayes	78.3%	87%	66%
SMO	77.5%	87%	63%
Logistic Regression	76.5%	84%	65%
Backpropagation	75.5%	84%	63%
C4.5	74.5%	85%	63%
3NN	71.6%	77%	58%

6. DISCUSSION

A review of the accuracy, sensitivity and specificity measures in Table 3 indicated that naïve Bayes and SMO were the top performers with many of the algorithms having highly comparable results. Given such similar results selection of the choice of algorithm to use is not obvious. As interested parties may have a preference for the choice of algorithm they would like to implement it is important to know if the use of a particular algorithm(s) would result in a statistically lower performance. To test the hypotheses that there would be statistically significant differences between the algorithms based on the accuracy, sensitivity and specificity measures, ANOVA tests with Tukey post-hoc analysis were implemented [19].

With regard to the overall accuracy measure an ANOVA test revealed that there were statistical differences between the algorithms, F (5, 594) = 4.134, $p < 0.001$. Post-hoc analysis revealed that there were no statistical differences between naïve Bayes, logistic regression, SMO, backpropagation and C4.5. However, 3-NN was found to have statistically significant lower accuracy than naïve Bayes, logistic regression and SMO but no statistically significant differences were found between it and C4.5 or backpropagation. Similarly, an ANOVA test revealed that there were significant statistical differences between the algorithms based on the sensitivity measure, F (5, 594) = 6.496, $p < 0.001$. Post-hoc analysis found this difference to be between 3-NN and all the other algorithms, with 3-NN having significantly lower sensitivity. No other differences were found. In terms of the specificity measure, although not as critical a measure in this study, no statistically significant differences were found between the algorithms.

Using the sensitivity measure to choose an algorithm, it would appear that any algorithm except for 3-NN is reasonable. However, naïve Bayes achieves the best results. In addition, an ANOVA test based upon the training times of each of the algorithms indicates that statistically significant differences exist, F (5, 594) = 3282.24, $p < 0.001$. Post-hoc analysis reveals that logistic regression, SMO and backpropagation have statistically significant higher training times than naïve Bayes and C4.5. This provides further evidence on the selection of naïve Bayes to predict introductory programming performance.

7. EPILOGUE STUDY

In the academic year 2014 to 2015 students enrolled on an introductory programming module, in a community college, participated in a study to verify the effectiveness of the naïve Bayes model at predicting programming performance and at the same time validating the study on a modern cohort of students. This was a significant piece of work due to the significant changes in the information technology landscape since the original study was undertaken in 2005/2006. Students were asked to answer questions based on the three factors: that is their mathematics result (high school exit examination), the number of hours spent playing computer games and ten questions to measure their programming self-esteem. All questions were taken directly from the original study and were not changed in any way. All of the 26 students who completed the module participated in the study. The study was carried out at the start of the academic year after a very brief introduction to programming had been given (variable declaration, printing and selection statements).

The full set of students (n = 26), were used as the training instances to validate the naïve Bayes model using 10-fold cross validation. The model achieved an overall prediction accuracy of 80.76% (6 students were misclassified). This was compared to the original study's prediction accuracy of 80.32%. A Welchs T-test, showed that there was no statistical difference between the accuracies produced in the two studies, with values of: T value of 0.7858 and a p value of 0.4342.

A follow up experiment was created in which the original participants' data was used as the training set and the new study data was used as the test set, this was to investigate that the model was truly timeless and enduring with the accuracies not diminishing as the environment evolved. This experiment produced an accuracy of 76.92%. The slight dip in accuracies could be caused by the institution representation in the original study. A community college only had a 7% representation in the original study, whereas in the new study it was 100% of the students. However, this study further confirms the effectiveness of the naïve Bayes model at predicting programming performance and that the factors identified are still valid.

8. CONCLUSIONS

This paper makes two significant contributions. First, it describes a longitudinal study to identify the most appropriate machine learning algorithm for predicting novice performance. Typically, models built to predict programming performance are statistical, with linear regression the most common technique used and thus the investigation of a suite of machine learning techniques to solve this problem is notable. The second contribution of this paper is that it provides a recent validation study. This is important as (1) it provides an opportunity to verify the performance of the naïve Bayes model and (2) it provides evidence on the generalizability of the findings. Studies of this nature are vital as too many one-off studies exist, making it difficult to interpret and use their findings.

The work described in this paper provides a baseline for further studies on the application of machine learning techniques to predict programming performance. Although the accuracy of the developed models is very high, 20% of students are still misclassified and further work to identify other significant factors and to optimise the models is warranted.

REFERENCES

[1] Bergin, S., & Reilly, R. (2005). Programming: Factors that influence success. ACM SIGCSE Bulletin, 37 (1), 411-415.

[2] Bergin, S., & Reilly, R. (2005). The influence of motivation and comfort-level on learning to program. Proceedings of the 17th Workshop on Psychology of Programming, PPIG'05, 293-304.

[3] Bergin, S., & Reilly, R. (2005). Examining the role of Self-Regulated Learning on Introductory Programming Performance. Proceedings of the 2005 International Workshop on Computing Education Research, ICER, 81-86.

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

[5] Cantwell Wilson, B. & Shrock, S. (2001). Contributing to success in an introductory computer science course: A study of twelve factors. ACM SIGCSE Bulletin 33(1), 184-188.

[6] Chih-Chung Chang, & Chih-Jen Lin (2001). LIBSVM: a library for support vector machines. Available: http://www.csie.ntu.edu.tw/cjlin/libsvm.

[7] Dietterich, T.G. (1997). Machine-learning research: four current directions. AI Magazine, 18, 4, 97-136.

[8] Ghent, J. (2005) A Computational Model of Facial Expression. PhD thesis, National University of Ireland Maynooth, Co. Kildare, Ireland.

[9] Michie, D., Spiegelhalter, D.J. & Taylor, C.C. (1994). Machine Learning, Neural and Statistical Classification. Ellis Horwood.

[10] Mitchell, T. (1997). Machine Learning. McGraw-Hill.

[11] Pintrich, P., Smith, D., Garcia, T. & McKeachie, W.(1991b). A manual for the use of the motivated strategies for learning questionnaire. Technical Report 91-B-004. The Regents of the University of Michigan.

[12] Platt, J.C. (1998) Fast Training of Support Vector Machines using Sequential Minimal Optimization. Advances in Kernel Methods - Support Vector Learning, B. Schoelkopf, C. Burges, and A. Smola, eds., MIT Press.

[13] Price, Kellie, and Suzanne Smith (2014). Improving student performance in CS. Journal of Computing Sciences in Colleges 30, (2) 157-163.

[14] Ramalingham, V. & Wiedenbeck, S. (1998). Development and validation of scores on a computer programming self-efficacy scale and group analyses of novice programmer self-efficacy. Journal of Educational Computing Research, 19(4), 367-381.

[15] Rosenberg M. (1965). Society and the adolescent self-image. Princeton, NJ: Princeton University Press.

[16] Russell, S. & Norvig, P. (2003). Artificial Intelligence: A Modern Approach. Pearson Education, Inc. Second Edition.

[17] ScholKopf, B & Smola, A. (2002) Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, MIT Press.

[18] Seewald, A.K. (2002). How to Make Stacking Better and Faster While Also Taking Care of an Unknown Weakness. Proceedings of the Nineteenth International Conference on Machine Learning 554-561.

[19] Tabachnick, B.G., & Fidell, L.S. (2001). Using Multivariate Statistics. Allyn and Bacon. Fourth Edition.

[20] Witten, I.H. & Frank, E. (2005). Data Mining: Practical machine learning tools and techniques. 2nd Edition. Morgan Kaufmann, San Francisco.

[21] Wolpert, D.H. (1992). Stacked generalization. Neural Networks, 5, 241-259.

[22] Yadin, A. (2011). Reducing the drop out rate in an introductory programming course. ACM Inroads, 2 (4), 71-76.