L.O. 4309

Statistical and Machine Learning Models to Predict Programming Performance

by

Susan Bergin, BSc(Mgmt), M.Comp.Sci.

NUI MAYNOOTH
Ollscoil na hÉireann Má Nuad

Dissertation submitted in partial fulfillment of the requirements for candidate for the degree of

Doctor of Philosophy

Department of Computer Science,
National University of Ireland, Maynooth, Ireland.

Supervisor: Professor Ronan Reilly

May 2006

Contents

List of Figures

List of Tables

List of Acronyms

ANN	Artifical Neural Network
BP	Backpropagation
FN	False Negative
FP	False Positive
K-NN	K-Nearest Neighbour
PBL	Problem Based Learning
PCA	Principal Component Analysis
RBF	Radial Basis Function
SRL	Self- Regulated Learning
SVM	Support Vector Machine
TN	True Negative
TP	True Positive

Declarations

I confirm this is my own work and the use of all material from other sources has been properly cited and fully acknowledged. Part of the work in this thesis has been presented in publications listed in Section 1.5

National University of Ireland Maynooth Susan Bergin

May 2006

Acknowledgements

First and foremost I would like to extend my sincere gratitude to my supervisor Professor Ronan Reilly. No matter how busy he was, he always made time for me and was a constant source of support and encouragement. I feel privileged to have had such a great mentor and hope that we get to work together again in the future.

The research documented in this thesis was funded in part by the Higher Education Authority. I'm grateful for this financial assistance. I'd also like to thank the students who participated in this research and each of the institutions who facilitated this work. I'm especially grateful to Rosemary, Diarmuid, Enda, Margaret and Karel.

I would also like to thank the technical and administrative staff at the Department of Computer Science. James, Michael, Misha and Patrick thank you for all your technical help over the years. Lilly, Ann and Louise, thanks for all your support and also for the many chats! I wish you all the very best for the future. Des Noonan warrants a very special note of thanks. I'm very grateful to him for his continuous assistance and support during my time at NUIM. Thanks Des.

It is often said that doing a PhD is a very lonely experience, but this was not true for me as I had such a great network of friends. Laura and Phelim, thanks for the copious amount of tea and chats we've enjoyed over the years. Peter, thanks for making labs so much more bearable, you're a great teacher, never forget that! All credit to Marky-Mark, his football league and his love of fine food and drink. His banter cheered up many a dull lunchtime! Des, thanks for making conference trips so much more fun, we had some good laughs! Thanks also for proofing this thesis. Johnny-John-John, thanks for all your help on SVMs, for proofing this thesis but mostly for being such a good friend. I look forward to having you as my new neighbour (mostly so you can walk Walt!). Thanks also to Jacq for the wonderful chats and laughs we have had over the past few years. You've been a great friend. I look forward to visiting you and Tom in Cambridge! As for Ado, where do I begin to thank you? The list is just too long. Thanks so much for everything, you are a top class person and I hope we will always be great friends. As for my non-NUIM friends thanks for always being there for me... I'm so lucky to have all of you as friends.

I would also like to thank my family. I have, in my opinion, the best parents in the world. They afforded me so much throughout my life. I'm delighted to dedicate this thesis to them. Thanks also to my brother and great friend Brian. His constant encouragement has gotten me to where I am today. Thank you. Finally, I'd like to thank my husband Andy. Without his constant unwavering support, I would never have completed this thesis. However did I ever get so lucky?

For Mam and Dad

Abstract

This thesis details a longitudinal study on factors that influence introductory programming success and on the development of machine learning models to predict incoming student performance. Although numerous studies have developed models to predict programming success, the models struggled to achieve high accuracy in predicting the likely performance of incoming students. Our approach overcomes this by providing a machine learning technique, using a set of three significant factors, that can predict whether students will be 'weak' or 'strong' programmers with approximately 80% accuracy after only three weeks of programming experience.

This thesis makes three fundamental contributions. The first contribution is a longitudinal study identifying factors that influence introductory programming success, investigating 25 factors at four different institutions. Evidence of the importance of mathematics, comfort-level and computer game-playing as predictors of programming performance is provided. A number of new instruments were developed by the author and a programming self-esteem measure was shown to out-perform other previous comparable comfort-level measures in predicting programming performance.

The second contribution of the thesis is an analysis of the use of machine learning (ML) algorithms to predict performance and is a first attempt to investigate the effectiveness of a variety of ML algorithms to predict introductory programming performance. The ML models built as part of this research are the most effective models so far developed. The models are effective even when students have just commenced a programming module. Consequently, timely interventions can be put in place to prevent struggling students from failing.

The third contribution of the thesis is the recommendation of an algorithm, based on detailed statistical analysis that should be used by the computer science education community to predict the likely performance of incoming students. Optimisations were carried out to investigate if prediction accuracy could be further increased and an ensemble algorithm, StackingC, was shown to improve prediction performance.

The factors identified in this thesis and the associated machine learning models provide a means to predict accurately programming performance when students have only completed preliminary programming concepts. This has not previously been possible.

Chapter 1

Introduction

1.1 Background

Student retention on third level Computer Science (CS) and Information Technology (IT) courses is a significant problem. Students find computer programming difficult and struggle to master the core concepts. Identifying struggling students is difficult as introductory programming modules tend to have a very high student-to-lecturer ratio (100:1 or greater) and often lecturers do not know how well students are doing until after the first assessment. This assessment may not take place until six or eight weeks after the module has commenced and, given the typically high number of students, marking the assessments can take a considerable length of time. Even if the assessment is indicative of likely overall performance on the module, it may be too late for students to withdraw from the course or for instructors to intervene to prevent struggling students from failing. This is a cause of great concern for computer science educators and has led to a body of research in the area.

Although many previous studies of programming predictors have interesting results it can be difficult to apply the results to other educational settings with different parameters, for example, the programming language taught and the assessment structure used. Furthermore, previous studies have often been carried out when students have completed a considerable part of an introductory programming module. The factors examined are often dependent upon the students having considerable experience with the module material and consequently it is difficult to know how predictive the same factors would be if measured earlier on the module.

A model that could predict likely programming performance in the first few weeks of a module would considerably help to alleviate this problem. To build such a model would require (1) the identification of early predictors of performance on an introductory programming module and (2) the appropriate implementation of a scientifically sound, robust modelling technique.

This thesis details a model that satisfies the above criteria [BR05c], [BR05b], [BR05a], [BR06a], [BR06c], [BR06d], [BR06b]. A study of early identifiable factors that influence performance on an introductory programming module is presented and numerous prediction models using these factors are developed. A recommendation of the most effective algorithm for predicting future student programming performance is provided.

1.2 Goals and Contributions

The goals of this thesis are to:

1. *Identify early predictors of performance on an introductory programming module.*

2. *Investigate the effectiveness of different machine learning techniques for predicting programming performance. The models should predict whether students are likely to be 'weak' or 'strong' programmers.*

3. *Recommend the use of the most suitable scientifically sound technique that should be used to predict future student performance.*

Moreover, there are three fundamental contributions to these objectives in this thesis, notably, (1) the identification of early predictors of programming performance, (2) a first study on the use of a variety of machine learning algorithms to predict introductory programming performance and (3) the recommendation of a suitable algorithm to predict the likely performance of incoming students. Each of these contributions is discussed in the following sections.

Identification of early predictors of programming performance

First, the research outlined in this thesis is based on a longitudinal study identifying factors that influence success on introductory programming modules. Although numerous studies have taken place, most studies are carried out when students have experienced a considerable part of an introductory programming module and thus it is unclear if the identified predictors would be accurate indicators if measured at an earlier stage in the course. In this thesis over 25 factors were examined at four different institutions. The study provides evidence on the importance of mathematics, programming self-esteem and computer game-playing as predictors of programming performance. The study also determined numerous factors that did not contribute further to the developed prediction models. These factors included prior programming experience, number of hours a student spends working at a part time job, encouragement from others to study programming, preference to work alone or in a group when solving problems and number of hours using application software, emailing or surfing the web before and during the early stages of the course. Second, a new instrument, (the Programming Self-Esteem Scale), developed by the author, was found to out-perform a measure referred to in this thesis as the Cantwell-Wilson and Shrock comfort-level measure and a shortened version of the Computer-Programming Self-Efficacy scale as a measure for predicting programming performance. Third, an investigation on the usefulness of self-regulated learning (SRL) for predicting performance was carried out. While SRL was not found to contribute further to the predictiveness of the model, except for the self-efficacy measure, the importance of particular aspects of SRL in learning to program were identified and as such provides justification for future research in the area.

Analysis of the use of machine learning algorithms to predict performance

In this thesis, six distinct machine learning (ML) algorithms are described and analysed to determine how effective each technique is at predicting programming performance. The algorithms examined are naïve Bayes, support vector machines, logistic regression, k-nearest neighbour, backpropagation networks and a decision tree (C4.5). Typically, linear regression is used to predict programming performance and while this is a well regarded statistical technique it is restricted by underlying assumptions, including normal distribution and linear relationship requirements. When these assumptions are not satisfied, subsequent work-arounds, where appropriate, can lead to interpretation problems and other difficulties. ML algorithms tend to have less stringent requirements and thus may be used to solve problems where linear regression fails. Following a detailed review of the published literature, it appears that this thesis is a first attempt to utilise a variety of ML algorithms to predict introductory programming performance. While similar techniques have been investigated in other academic domains, no comprehensive study of the suitability of a range of ML techniques has taken place within this domain. This work provides the foundation for future work on the application of artificial intelligence techniques to this problem and encourages computer science educators to consider a broader suite of tools.

Each algorithm was implemented using 10-times 10-fold stratified cross validation. Although this approach is more computationally intensive than the commonly used 'hold out' method, all examples in the dataset are used for training and testing and thus confidence on the generalisability of the results is increased. In addition the stratification process improves the representativeness of each fold as the process seeks to represent the same proportion of each class in a fold as is in the original full dataset [WE05].

The models developed as part of this thesis are the most effective models developed to predict programming performance. Comparable results have never been achieved before. The models are effective even when students have just commenced a programming module. Consequently, timely interventions can be put in place to prevent struggling students from failing. The models also identifies strong students and thus additional resources or alternative streams could be provided to further develop their skills. In addition, the fact that the models are based on three years of studies and involve students from multiple institutions enhances the likely generalisability of the findings. Indeed just recently, Fincher and Petre [FP04], two leading researchers in the CSEd field, argued that repetition and generalisation are the key drivers for research that can be considered valid, relevant and important.

Recommendation of a suitable algorithm to predict the likely performance of incoming students

Inspection of the predictions made by each of the machine learning algorithms indicated that naïve Bayes was the most effective algorithm for predicting programming performance. To confirm this, detailed statistical analysis was carried out to determine if there were any statistically significant differences between the prediction accuracy of each of the algorithms and also between the training times of each algorithm. The analysis confirmed naïve Bayes as the most successful algorithm for predicting programming performance.

Although the examination of six machine learning techniques and the subsequent statistical analysis was a significant contribution to the field, optimisations were also carried out to investigate if the results could be further improved. Although several techniques were implemented with varying performance, one particular ensemble algorithm, StackingC, was shown to improve the results achieved by the naïve Bayes model. However, the improvements are not sufficient to justify the additional work required to implement StackingC. Therefore, naïve Bayes is still recommended for predicting incoming student programming performance.

1.3 Thesis Overview

Chapter two provides a review of previous research on factors that influence programming. The studies are summarised and the results are outlined. In addition, self-regulated learning, a topic that is currently receiving considerable research attention, is introduced and its potential for predicting academic performance is discussed. In Chapter three the materials and methods used in this research are presented. First, a pilot study carried out in the academic year 2003-2004 is described. The results of this study are outlined and discussed and the main study is then introduced. A detailed review of the instruments used, the participants and data pre-processing is provided. Chapter four provides an overview of the six machine learning techniques used to model the data. Each algorithm is described and an evaluation of its strengths and weaknesses in relation to our research is provided. In addition, an overview of Principal Component Analysis (PCA), a dimensionality reduction technique is outlined. Chapter five describes the results achieved by each of the techniques. Further optimisations are proposed and implemented and the subsequent results are outlined. In Chapter six the overall results are discussed and analysed. An epilogue study, which was carried out to verify the findings is described and the results are discussed. Chapter seven provides conclusions and suggests possible directions for future work.

1.4 Motivation

Computer science courses have a notoriously high attrition rate worldwide and Ireland is no exception. In a study carried out in 2001 by the Higher Education Authority (HEA) in Ireland [MFK01] the field of computer studies was found to have the highest rate of non-completion. Over one-quarter (26.9%) of students who commenced in this field failed to complete with a further 14% of students graduating late. A considerable cause of failure is CS1, typically an introductory programming module. A recent multi-institutional, multi-national study [MAD^{+}01] found that after completing an introductory programming course students scored an average of 21% on a short lab based assessment, which the module lecturers anticipated the students would pass. At the Department of Computer Science, NUI Maynooth, on average 34% of students failed on their first attempt of the first year programming module over the four academic years starting in 2001. In addition 21% of students who registered for the module did not complete it but typically transferred out of computer science.

Numerous studies over the past 30 years have attempted to examine factors that influence programming performance while no computer science education conference is complete without at least one session dedicated to issues in learning how to program. However identifying struggling students is still a difficult task. Often the first programming assessment does not take place until after the first six weeks of the module and when a large number of students take the course, correction can take a considerable amount of time. Even when early assessments are indicative of overall performance, it may be too late for students to withdraw from the course or for instructors to intervene to prevent struggling students from failing. An early measure of how a student is likely to perform on an introductory programming module is required. Previous studies have considered a wide range of factors that could influence performance. These factors include: previous academic and computing experience, various cognitive and behavioural factors and level of comfort felt on an introductory programming module. However, it is difficult to know how to apply the findings of these studies or to feel confident that the findings would hold true in a further study as typically, the studies are one-off and little if any attempt has been made to replicate the findings.

Two important longitudinal studies (PhD theses) in the area were carried out by Cantwell-Wilson in 2000 [CW00] and Ventura in 2003 [Ven03]. Cantwell-Wilson examined 12 factors and a multiple regression model was able to account for 44% of the variance in midterm results. Similarly, Ventura's PhD thesis (2003) developed several multiple regression models for predicting performance on an objects-first programming module. The most significant regression model developed that did not include variables directly related to the module, for example, the number of continuous assessment exams taken and the percentage of lab assignments submitted, was able to account for 53% of the variance in course results [Ven03]. Although multiple regression analysis is a well respected technique, it makes a number of assumptions including a normal distribution of the residuals and linearity of relations among the variables. This can be problematic if these assumptions are not satisfied. Other techniques exist, such as Artificial Neural Networks (ANNs), that do not require normal distributions and can handle non-linearly related data, often produce results superior to those of multiple linear regression models. However, no studies to date have used machine learning techniques to predict introductory programming performance. While predictions of reasonably accurate student results could be useful, in terms of retention it is of less importance than knowing whether students are likely to struggle with the material or not. As such, being able to classify students as 'weak' or 'strong' programmers would be a significant contribution to the field.

Over the past six years the author has been involved in running the introductory programming module within the Department of Computer Science, NUI Maynooth. During this time it was observed that many students struggle to learn how to program and the author has been directly involved in the development of several initiatives to assist them. These initiatives have included the introduction of Problem-Based Learning (PBL) workshops, the development of an intranet application to provide a centralised repository of first year information, the development of an intranet-based programming assessment system, weekly programming clinics and tutorials. Although it is too early to say definitively, initial evidence suggests that these initiatives have been beneficial [OBG+04]. However, such initiatives are limited if it is not known early on which students need extra resources or material in order to help them to learn how to program. Thus, the research addressed in this thesis is of great importance to the author in terms of both its research value and its application to her work on improving retention.

1.5 Publications

Part of the work in this thesis has been presented in the publications listed in this section.

1. Bergin. S, Reilly. R. "Using Machine Learning Techniques to Predict Introductory Programming Performance", *Applied Artificial Intelligence*, submitted May 2006.

2. Bergin. S, Reilly. R. "Predicting programming performance using machine learning techniques", *NUIM Postgraduate Symposium*, March 2006.
3. Bergin. S, Reilly. R. "A computational model to predict introductory programming performance", *NUIM Technical Report Series*, January 2006.
4. Bergin. S, Reilly. R. "Predicting Introductory Programming Performance: a multi-institutional multivariate study", *Computer Science Education*, submitted January 2006.
5. Traynor. D, Bergin. S, Gibson, P. "Automated Assessment in CS1", *Australian Computing Education Conference (ACE'06)* 2006.
6. Bergin. S, Reilly. R. "Examining the role of Self-Regulated Learning on Introductory Programming Performance", *Proceedings of the first International Computer Science Education Research Workshop, ICER05*, pg 81–86, 2005.
7. Bergin. S, Reilly. R. "The influence of motivation and comfort-level on learning to program", *Proceedings of the 17th Workshop on Psychology of Programming, PPIG'05*, 2005.
8. Bergin. S, Reilly. R. "Programming: Factors that influence success", *Proceedings of the 36th SIGCSE technical symposium on computer science education, SIGCSE '05*, pages 411–415, 2005.
9. Bergin. S, Reilly. R, "Assessing programming Aptitude: A web-based adaptive testing system with remedial instruction", *16th Doctoral Workshop of the Programming Psychology Interest Group (PPIG'04), Carlow IT*, April 2004.

10. Bergin. S, Reilly. R, "An adaptive web-based testing system to assess programming aptitude". *NUIM postgraduate symposium* March 2004.

Chapter 2

Literature Review

Predictors of performance on introductory programming can broadly be classified into the following categories: previous academic and computer experience, cognitive factors and psychological factors with emphasis on perceived comfort-level on the course. An overview of studies within each of these categories is provided and further supplementary information is presented in Table 2.1.

2.1 Previous Academic and Computing Experience

Previous academic experience and programming experience have often been cited as predictors of programming success. Numerous studies have found that both mathematical ability and exposure to mathematics courses is related to performance on introductory computer science modules. Similarly, performance in and experience of other academic subjects have been shown to be important. Finally, several studies have ascertained that prior programming experience and non-programming computer experience are useful predictors of programming performance.

2.1.1 Previous Academic Experience

Numerous studies have investigated the role of mathematics in learning to program. Leeper and Silver [LS82] developed a regression model that included the number of high school mathematics units completed and SAT mathematics score that could account for 26% of the variance of overall results on an introductory programming course. SAT mathematics was found to have one of the strongest correlations with performance, $r = 0.37$. Similarly, Byrnes and Lyon [BL01] in a recent Irish study, found a significant correlation between Leaving Certificate mathematics, $r = 0.353, p < 0.01$, and the overall results students achieved on a first year programming and logical methods module taught using BASIC. In a study by Evans and Simkins [ES89] the number of high school mathematics courses a student had taken prior to the course was found to be a significant predictor of homework assignment performance. However, multiple regression models developed using this variable and others accounted for at most 23% of the variance. In a study of 12 factors that influence performance on an introductory computer science course, Cantwell-Wilson and Shrock [CWS01] found mathematics background to be the second most important predictor. Hostetler [Hos83] also found a student's mathematics background to be a significant predictor of performance. A regression model was able to classify 61 of 79 students (77.2%) into high and low aptitude groups based on the overall grade a student achieves on an 'Introduction to Computers and Their Application to Business' course using Fortran. Honour-Werth [HW86] found a significant correlation between the number of high school mathematics courses taken, $r = 0.252, p < 0.05$ and student performance on an introductory computer science course. Similarly, Konvalina [KWS83] found that students who completed a first technical computer science course had significantly more mathematical background before taking the course than students who withdrew from the course. Stein [Ste02] in a study of 160 students on a programming course through Java found that students who studied calculus do at least as well as students who studied discrete mathematics. Thus, he concluded that having some form of mathematical maturity is more important for programming than experience with specific mathematical topics and this conclusion appears to be well supported by the findings of the studies presented here.

Ability in and experience of academic subjects other than mathematics has also been previously investigated. Leeper and Silver [LS82] developed a regression model using units of high school English, mathematics, science, a foreign language, SAT verbal score, SAT mathematics score and high school rank that accounted for 26% of the variance of results on an introductory programming course. Byrnes and Lyon [BL01] found a significant correlation between Leaving Certificate science scores ($r = 0.572, p < .01$) with the overall results students achieved on a first year programming and logical methods module through BASIC. However, neither Leaving Certificate English nor performance in a foreign language were found to be good predictors of performance.

2.1.2 Prior Computing Experience

Evans and Simkins [ES89] found that prior BASIC experience was a predictor of performance in an entry level business computer class while Cantwell-Wilson and Shrock [CWS01], similarly found that a previous formal class in programming had a positive relationship with performance on a introductory computer science course using C++. Konvalina [KWS83] in a study of 382 students found that students who withdrew from an introduction to computer science course ($n = 154$) had significantly lower computer science experience than students who completed the course ($n = 228$). In a study of 75 students on an introductory programming course, Hagan [HM00] found that students with previous programming experience had a significantly higher mean score than students with no previous experience on all assessments except the final examination. In addition, the study found that the more programming languages a student knows prior to taking the course, the higher their performance. Holden [HW04] found that prior experience (independent of the programming language) is an advantage in the first course on an 'Introduction to Programming' sequence but not in later courses. It appears that while prior experience is initially an advantage, as an introductory programming course progresses it loses its importance. In Ireland, programming is not an examination subject on the second level curriculum and consequently few students commence introductory programming modules in third level with any formal training. As the studies presented in this section all took place in the United States of America, it is difficult to know how relevant their findings are to an Irish study of programming factors. Byrnes and Lyon [BL01] attempted to examine the role of prior programming performance in an Irish study but found that so few students in their study had previous programming experience it was not possible to draw any conclusions.

2.2 Cognitive Factors

The role of various cognitive factors in learning to program has also been researched. Previous studies have investigated various cognitive factors, including cognitive style and abstract reasoning ability, and provide useful insights into the role of cognition in learning to program.

Hostetler [Hos83] investigated the performance of 79 students studying an 'Introduction to Computers and Their Application to Business' class using Fortran. The study found that diagramming (analysing a problem and ordering the solution into a logical sequence) and reasoning (translating ideas and operations from text based problems into mathematical notations) tests on the Computer Programmer

Aptitude Battery (CPAB) along with a students' grade point average (GPA) were the most important predictors of success. A multiple regression model, using these three variables with mathematics background and a personality factor (a measure of how carefree an individual perceives themselves to be), correctly classified 61 of 79 students (77.2%) into high- and low-aptitude groups based on the overall grade a student achieves.

Kurtz [Kur80] designed and carried out a test on the formal reasoning abilities of 23 students on an introductory programming course using Fortran. Based on performance, students were classified at three intellectual development levels: late concrete, early formal and late formal. The levels of late concrete and late formal were found to be strong predictors of low and high performance respectively. A regression model using the formal reasoning levels accounted for 63% of the variance in grades. Honour-Werth [HW86] used Kurtz's measure of intellectual development [Kur80] and found significant correlations between the measure and performance on an introductory programming course using Pascal, $r = 0.232, p < .05$. They also found a correlation between performance and cognitive style $r = 0.317, p < .01$. Barker and Unger [BU83] implemented a shortened version of Kurtz's test [Kur80], reducing the time required to take the test from 80 to 40 minutes and the number of questions from 15 to 11. The test was administered to 353 students. ANOVA testing indicated that there was a significant difference between the mean score of the late concrete group and the means of the early and late formal groups. A regression model using level of intellectual development accounted for 12% of the variance in the final course grade. Gibbs [Gib00] measured the relationship between cognitive style (field dependent and field independent) and programming performance. In a study of 50 students on a first programming course using a constructivist learning environment, no significant correlations were found between cognitive style and programming achievement. Mayer [MDV86] investigated the relationship between learning to program and 'learning to think'. Eight thinking skills were assessed and three measures, word–problem translation (translating word problems into equations), word–problem solution (giving the correct numerical answer to word problems) and following directions, were found to be significant predictors of performance accounting for 50% of the variance in results. Although this result is considerable, the study only involved a small sample ($n = 50$) and has not been validated through further studies. Austin [Aus87] found that quantitative and algorithmic reasoning abilities and iconic pattern recognition abilities were important indicators of programming success. The latter was found to be important for both program reading and writing while the former was only found to be important for program reading.

In summary, a wide variety of cognitive factors have been investigated including logical ability, reasoning ability, arithmetic ability, intellectual development, and cognitive style. It appears that certain cognitive factors play a role in learning to program but typically the strength of this relationship is not particularly strong and in general accounts for very little of the variance in student performance.

2.3 Psychological Factors

Recently researchers have examined the relationship between students' expectations of an introductory computing module and their actual experience of it. Cantwell-Wilson and Shrock [CWS01] in a recent longitudinal study found that the most important predictor of students' performance on an introductory computer science course was comfort-level measured by the degree of anxiety a student felt about the course. The measure includes level of comfort asking and answering questions in class, labs and during office hours; designing programs without help, understanding programming concepts, completing lab assignments and perceived understanding of material compared to classmates. A multiple regression model, $F(12, 92) = 6.13, p = 0.0001$, with three significant variables: comfort-level, mathematics background and attribution of success or failure to luck resulted in an adjusted R square of 44%. Ventura [Ven02] examined predictors of a graphical design-centric objects-first Java course and found that student effort (as measured by the number of hours spent using the labs) and comfort-level were the strongest predictors of success. A stepwise regression model using these two predictors and SAT mathematics score accounted for 52.9% of the variance in course grade. Holden [HW04] measured the comfort-level of students using computers at the start of a programming course. No relationship between this measure and performance were found. It is important to note that the measure was specific to computer usage and not a measure of programming comfort-level. Newsted [New75] in a study of 131 introductory Fortran students found that two of the most important predictors of performance were perceived ability and time spent working on the course and working with other students. While perceived ability was found to be positively related to performance, the number of hours spent working on the course was negatively related to performance. This latter finding is interesting when compared to Ventura's [Ven02] finding on the positive relationship between the number of hours the students spent using the labs and programming performance and requires further investigation. Hagan [HM00] found a significant difference between the confidence levels felt by students, with and without previous programming experience, that they would pass an introductory programming course, $F = 4.558, p < 0.05$. In addition, Rountree *et al.* [RRR02] in a study of 472 students found that the grade a student expected to achieve in an introductory module was the most important indicator of performance. Goold and Rimmer [GR00] identified that 'dislike of programming' is related to performance on an introductory programming course. A multiple regression model composed of (in order) dislike for programming, gender, average on other modules and a measure referred to as 'raw secondary score' which incorporated student performance in English and their best (other) three subjects taken for final examination in secondary school accounted for 43% of the variance in student results. The authors did not find expected grade or mathematics to be predictors of performance.

The relationship between students' mental models of and self-efficacy in programming (an individuals' judgements of their capabilities to perform various programming tasks) and their performance has also been investigated [RLW04] [Wie05]. Recently Ramalingham *et al.* [RLW04] investigated the effects of students' self-efficacy for programming and their mental models of programming and found that self-efficacy is influenced by a students' mental model of programming and also by previous computer and programming experience. Significant differences were found in the pre- and post-self-efficacy measures. A path analysis model with pre- and post-programming self-efficacy, programming mental model and previous experience as predictor variables accounted for 30% of the variance in the final course grade. Similarly, in a study by Wiedenbeck [Wie05] measures of pre- and post-self-efficacy of programming were shown to be important predictors of performance. A measure of previous computer and programming experience was found to be a strong positive predictor of pre-self-efficacy. However, pre-self-efficacy was found to have a negative relationship with performance and the author suggests that perhaps some students are over confident at the start of the course. Knowledge organisation was measured using a program recall task and found to have a positive relationship with performance. A regression model composed of previous experience, pre- and post-self-efficacy and knowledge organisation was able to account for 30% of the variance in final grade.

It appears that how students feel before and during an introductory programming course ('comfort-level') is a very important predictor of performance on an introductory programming course. Research in this area has been very fruitful and further studies on this topic would be well justified.

2.4 Miscellaneous Factors

The previous sections have outlined the main body of research on factors that influence introductory programming success. Other factors have also been considered, albeit often in once-off studies that do not fall into the previous categories. In particular two factors have been investigated in several studies and have been found to relate to programming performance. The factors are (1) the number of hours a student spends playing computer games and (2) the number of hours spent working at a part-time job. Cantwell-Wilson and Shrock [CWS01] found that the number of hours students played computer games was negatively related to performance on an introductory computer science course. Similarly, Evans and Simkins, [ES89] found the number of hours students spent playing electronic games (video and computer games were studied) to have a negative relationship with performance on an introductory Basic course. They also found that the number of hours a student spent working at a part-time job negatively relates to performance. Honour-Werth [HW86] found a significant correlation between hours working at a part-time job $r = 0.203, p < .1$ and performance on an introductory computer science course using Pascal.

Researchers	Language	n	Reference Criterion	Significant Predictors
[Aus87]	Pascal	76	Composite score based on program reading, program writing and lab performance.	A model composed of high school composite achievement, quantitative and algorithmic reasoning abilities, vocabulary and general information abilities, self-assessed mathematics ability and measures of an introverted/analytic style and extroverted level was able to account for 64% of the variance in results.
[BU83]	C++ and C	353	Final grade of students in 15 class sections learning two different programming languages.	Intellectual development (ID) test successfully predicted advanced students. A regression model based on ID level accounted for 12% of the variance in final course grade.
[BL01]	BASIC	110	Overall result on 'Programming and Logical Methods' for first year Humanities students (70% written examination and 30% weekly assignments).	Significant correlations found for: Leaving Certificate (LC) mathematics result ($r = 0.353, p < .01$), LC science result ($r = 0.572, p < .01$) with programming performance.
[CWS01]	C++	105	Midterm course grade on 'CS202 Introduction to Computer Science' for introductory computer programming students.	$F(12, 92) = 6.13, p = 0.0001$, adjusted R square = 44%, three significant variables: comfort-level, mathematics background and attribution of success/failure to luck. Secondary analysis indicated that the number of hours playing computer-games prior to the course had a negative effect on grade while experience of a prior formal programming class had a positive effect.

continued on next page

continued from previous page

Researchers	**Language**	**n**	**Reference Criterion**	**Significant Predictors**
[ES89]	BASIC	117	Six outcome variables on an entry level business computer class (homework score, 2 BASIC exams, 2 midterm exams and a final examination).	Various predictors dependent upon outcome variable found (for example, number of high school mathematics courses, prior BASIC experience, hours playing video or computer games). Multiple linear regression modelling accounted for at most 23% of the variance on each outcome variable.
[Gib00]	BASIC	50	Two achievement tests on a first programming course of computer science students. The first test was on designing and the second test was on coding.	Within a constructivist learning environment, cognitive style (field dependent and field independent) was not found to influence programming achievement (on either tests).
[GR00]	C	39	Result on an introductory programming concepts course for students enrolled in a computer-related degree (60% examination and 40% assignments).	A multiple regression model, composed of (in order) dislike for programming, gender, average on other modules and raw secondary score accounted for 43% of the variance in scores.
[HM00]	Java	97	Five outcome variables on a first programming course in an undergraduate computing degree (2 tests 30%, 2 assignments 30% and examination 40%).	Significant difference between the performance of students with and without prior programming experience on all assessments except the final examination. Further analysis indicated that the more programming languages a student knew prior to taking the course, the higher the performance. They also found a significant difference between the confidence level of students with and without prior experience on their expectation to pass the class ($F = 4.558, p < 0.05$).

continued on next page

continued from previous page

Researchers	Language	n	Reference Criterion	Significant Predictors
[HW04]	Java	159	Three in-class exams on the first course of an 'Introduction to Programming' sequence.	Prior experience (independent of language) is an advantage in the first course in the sequence but not in later courses.
[HW86]	Pascal	58	Student grade on 'Computer Science I' (First class in the ACM Curriculum). Grade included scores on homework assignments 30%, examinations 40%, quizzes 10% and a programming project 20%.	Significant correlations found for: high school mathematics ($r = 0.252, p < .05$), hours working at a part-time job ($r = 0.203, p < .1$) , Piagetian intellectual development ($r = 0.232, p < .05$) and cognitive style ($r = 0.317, p < .01$) with performance.
[Hos83]	Fortran	79	Overall grade on an 'Introduction to Computers and Their Application to Business'. The grade consists of scores on programming assignments, two one-hour examinations and a three-hour final examination.	A multiple-regression model, using diagramming and reasoning score on the Computer Programmer Aptitude Battery (CPAB), GPA, mathematics background and a personality factor (sober or happy-go-lucky trait), correctly classified 61 of 79 students (77.2%) into high and low aptitude groups.
[KWS83]	Basic	382	Final examination on an 'Introduction to Computer Science' course based on CS1 in Curriculum '78.	Found significant differences between the mathematics background of students who withdraw from the course and students who complete the course, in that students who completed the course had significantly more mathematical background before taking the course.
[Kur80]	Fortran	23	Final grade on an 'Introduction to Programming' course for computer science majors and non-majors. Grade composed of scores on programs 55%, quizzes 10%, midterm 11.7% and final examination 23.3%.	Level of formal reasoning is a strong predictor of performance (specifically poor and outstanding performance), accounted for 63% of the variance in grades.

continued on next page

continued from previous page

Researchers	Language	n	Reference Criterion	Significant Predictors
[LS82]	Not specified	92	Grade on an Introductory Programming Course.	Regression model accounted for 26% of the variance using number of high school english, mathematics, science and foreign language units completed, SAT verbal and SAT mathematics score and high school rank. Strongest correlations found for SAT verbal and SAT mathematics score.
[MDV86]	Basic	57	Basic examination score.	Regression model accounted for 50% of the variance using two problem translation skills: word problem translation and word problem solution and a procedure comprehension skill: following directions.
[New75]	Fortran	131	Grade on an Introductory Programming Course.	Regression model using perceived ability, college GPA and time spent working on the course accounted for 41% of the variance. The first two variables were found to be positively related to performance but time spent working on the course was found to be negatively related.
[RLW04]	C++	75	Final grade on an introductory programming course for CS majors and non-majors.	Used path analysis to determine that mental models affect success directly and strengthen self-efficacy. Self-efficacy is influenced by previous performance. Mental model, self-efficacy and previous programming and computer experience accounts for 30% of the variance in final course grade.

continued on next page

continued from previous page

Researchers	Language	n	Reference Criterion	Significant Predictors
[RRR02]	Java	472	Final mark on a first year programming course. The mark was composed of 30% bi-weekly programming assignments, 20% mid-semester examination and 50% final examination.	The strongest indicator of success was the grade a student expected to get on the course.
[Ste02]	Java	160	Performance on CSII, a programming course.	Students who study Calculus do at least as well as students who study discrete mathematics. Thus, some form of mathematical maturity is important for programming.
[Ven02]	Java	499	Course grade on a graphical design-centric objects-first course. Grade based on weighted average of homework and quizzes (10%), lab average (40%) and exam (50%).	A stepwise regression model using a log of percent lab usage, comfort-level and SAT mathematics score accounted for 52.9% of the variance in course averages.
[Wie05]	C++	120	Final grade of non-majors on an introductory programming course.	A measure of previous computer and programming experience along with pre and post measures of self-efficacy and knowledge organisation accounted for 30% of the variance in final grade. Previous experience is a strong predictor of pre-self-efficacy.

Table 2.1: *Previous research on factors that influence programming performance*

2.5 Self-Regulated Learning

Although numerous studies of factors that influence programming performance have been carried out, a comprehensive understanding of the factors has yet to be realised and thus further predictors are required. Recently, self-regulated learning (SRL) has become an important topic in education and psychology. Zimmerman [Zim86] defines SRL as the degree to which learners are meta-cognitively (the control of cognition through planning, monitoring and regulating [Pin89]), motivationally and behaviorally active participants in their own academic learning. Furthermore, Pintrich and DeGroot [PD90] propose that a complete model of self-regulated learning should incorporate cognitive and meta-cognitive strategies, referred to as a *skill* component, and motivational components, referred to as *will* components.

A well recognised model of self-regulated learning was developed by Pintrich and his colleagues [PSGM91] and includes *skill* and *will* components of self-regulated learning. The *skill* component includes cognitive strategies, meta-cognitive strategies and resource management strategies. The *will* component is composed of various motivations, including intrinsic goal orientation and task value [Pin99].

Cognitive strategies include rehearsal, elaboration and organisational strategies. Rehearsal strategies include the recitation of information to be learned and mnemonic techniques for memory tasks. These strategies are assumed to help learners to attend to and select important information from lists or texts, but may not reflect a very deep level of processing. Elaboration strategies involve paraphrasing, summarising, creating analogies and generative note taking. These strategies help learners to integrate and connect new information with prior knowledge. Organisational strategies include clustering, outlining and selecting the main ideas from texts. These strategies help learners to select appropriate information and to develop connections between different pieces of information [Pin99], [PSGM91].

Meta-cognitive strategies include planning, monitoring and regulating cognition. Planning includes setting goals, skimming a text before reading and analysing tasks. These activities help to activate relevant aspects of prior knowledge, making the comprehension of the material easier. Monitoring includes tracking one's attention when reading or listening and self-testing using questions. Regulation concerns the continuous modification of one's cognitive activities. For example, a student monitors her attention while reading an article to make certain that she understands its content. When she realises through her monitoring activities that she has not comprehended a portion of the text, she will go back and reread the difficult part of the article. This rereading of text is a regulation strategy [Pin99], [PSGM91]. Finally, resource management strategies refer to strategies students use to manage their time, their effort, their environment and other people, including their interaction with other students and teachers to seek help.

A considerable number of studies [PBV00], [PB90], [PD90], [ZMP90] have consistently found a significant positive correlation between academic achievement and self-regulated learning among elementary, high school, and college students. In addition, numerous studies [PRP99], [PG91], [Pin89], [PB90], have found that intrinsic goal orientation and high task value in a topic (beliefs about the importance of, interest in and utility value of the task) are strongly positively correlated with the use of cognitive and meta-cognitive strategies and also with academic performance.

While numerous studies have been carried out to determine factors that relate to programming success the findings are largely inconclusive and suggest that perhaps more evidence of potential factors needs to be gathered. It appears that computer science education researchers have yet to examine, in detail, the role of SRL in learning to program and, more specifically, if SRL is a useful predictor for programming performance. Given the significant findings on the role of self-regulated learning in other academic domains it would appear well justified to examine its role in learning to program.

2.6 Summary

Over the past 30 years numerous studies have investigated predictors of programming success. Ability in and exposure to mathematics has been confirmed as a factor by several studies. A science background also appears to have a positive relationship with introductory programming performance, however the role of languages including english and foreign languages is less clear. Prior programming experience has also been found to be a predictor, however, previous studies have taken place in countries where students can study programming for examination at national level. This is not the case in Ireland and therefore it is difficult to know how to apply the findings to such a setting.

Several researchers have focused on the role of cognitive factors in learning to program. Numerous factors including arithmetic ability, logical ability and reasoning ability have been found to relate to success in programming. However, the strength of the relationship appears to be weak. Psychological factors have also been examined and research on the role of students experience on and expectations of an introductory programming course has proved very fruitful. Numerous studies have found that perceived comfort-level on the module is the best overall predictor of programming performance. It would appear that further work in this area would be well-justified.

Finally, although numerous studies on factors that influence programming performance have been carried out the smallest set of predictors that can account for the most significant amount of variance in results has yet to be established. A considerable number of studies have found a significant positive correlation between academic achievement and Self-Regulated Learning (SRL) among elementary, high school, and college students. Computer science educational researchers have yet to examine, in detail, the role of SRL in learning to program but such research would appear to be well justified.

Chapter 3

Methodology: Subjects, Instruments, Institutions and Studies

In this chapter the methods employed to gather data for this research are outlined. First, a pilot study undertaken in the academic year 2003-2004 is described. The predictor variables investigated, the methodology used, the analytical techniques and the main findings are discussed. The research methodology employed for a subsequent detailed study carried out in the academic year 2004-2005 is then presented. An overview of the participants involved, the instruments used and data pre-processing is provided.

3.1 Pilot Study

During the academic year 2003-2004 a pilot study was carried out, at the Department of Computer Science NUI Maynooth, to investigate factors that could influence success on an introductory programming module. The introductory programming module at the university is composed of a one-and-a-half hour Problem-Based Learning (PBL) workshop, a one-and-a-half hour laboratory session and three one-hour lectures per week over two semesters [OBG+04]. Students in Ireland do not study programming for national examination in secondary school and the majority of students taking this module have recently completed second level education.

The selection of factors for this study were based on the findings of the literature review, as outlined in Chapter 2. However, factor selection was constrained for a number of reasons, including availability of participants, length of completion time needed for each instrument and the stage in the academic year. Given these restrictions the relationship between 15 factors and performance on the introductory module was investigated. The factors fall into four broad categories:

1. Previous academic and computer experience: as measured by performance in the Irish Leaving Certificate (LC) examinations in mathematics and science subjects and self-reported computer experience.

2. Specific cognitive skills: as measured by an in-house cognitive test.

3. Personal information: gender, age, work-style preference (preference to work alone or as part of a group), encouragement from others and the number of hours per week spent working at a part-time job.

4. Experience on the module: students perception of how well they are doing and how comfortable they feel with the module material.

Performance on this module is based on continuous assessment (30% of the overall mark) and a final examination (70% of the overall mark). The measure of performance reported upon in this study is the overall module mark.

3.1.1 Instruments

Two instruments were used to collect data: a questionnaire and a custom-made cognitive test. The questionnaire collected data on the following items: (1) LC mathematics grade, (2) LC physics grade, (3) LC biology grade, (4) LC chemistry grade, (5) highest LC science grade, (6) comfort level on the module (likert-scale questions based on a questionnaire previously used by Cantwell-Wilson and Shrock [CW00] that measured a student's perceived understanding of programming concepts, difficulty designing programs without help, difficulty in completing lab assignments, and their ease at asking and answering programming questions), (7) perceived understanding of the module material (based on a single likert-scale question again based on a questionnaire by Cantwell-Wilson and Shrock [CW00]: 'How do you rate your level of understanding of the programming module?'), (8) prior programming experience, (9) prior non-programming computer experience, (10) work-style preference, (11) encouragement from others to study computer science, (12) number of hours per week working at a (part-time) job. The cognitive test was developed within the Department of Computer Science at NUI Maynooth [1] and comprised items involving numerical and letter sequencing, arithmetic reasoning, procedural ability and problem translation skills. In addition, information on gender, age and overall module results was available for all students taking the module. Both instruments were completed in the second semester of the module and data collection for both was paper-based.

3.1.2 Participants

Students enrolled in the first year 'Introduction to Programming' module voluntarily participated in the pilot study. For each of the studies outlined in this thesis,

[1]Developed by Jacqueline McQuillan, Department of Computer Science, NUI Maynooth.

students were provided with an information sheet about the research and signed a consent form agreeing to participate. Permission for the research activities documented in this thesis was also granted by the Ethics Committee at NUI Maynooth. Ninety-six students completed the module in the academic year 2003-2004. In total 80 students (49 male, 31 female) completed the cognitive test and 30 (19 male, 11 female) students completed the survey.

3.1.3 Initial Analysis

A review of the literature, as documented in Chapter 2, indicated that scientific analysis in this area is typically based on statistical techniques such as correlation and regression. To maintain consistency with previous studies and to allow findings to be directly compared, initial analysis in this thesis used the same techniques. However, subsequent analysis employed techniques tailored to our research requirements, that is, techniques that are suited to predicting weak and strong programming students.

Statistical Techniques

Initial analysis of the data employed Pearson correlation coefficients, multiple linear regression and t-tests for independent samples. In order to satisfy some of the underlying assumptions of these techniques equality of variance and normality tests were also performed. Each of these techniques are briefly described in this section.

Pearson's Product Moment Correlation Coefficient measures the strength of the linear relationship between two variables (X and Y). It is based on the assumption that both variables are interval or ratio based and are sampled from populations that follow a normal (Gaussian) distribution. This type of correlation is usually signified by r and can take on values from -1.0 to 1.0, where -1.0 is a perfect negative correlation, 0.0 is no correlation, and 1.0 is a perfect positive correlation [Hin95]. The formula for calculating r is given by Equation 3.1 where N is the sample size.

$$r_{xy} = \frac{\sum XY - \frac{\sum X \sum Y}{N}}{\sqrt{(\sum X^2 - \frac{(\sum X)^2}{N})(\sum Y^2 - \frac{(\sum Y)^2}{N})}} \tag{3.1}$$

Multiple regression is used to assess the relationship between a continuous dependent variable and several linear combinations of independent variables. It can also be used to establish the relative predictive importance of the independent variables. Regression requires that a number of underlying assumptions are satisfied, including, the absence of outliers (very high or very low independent variable values), a linear relationship between the predicted dependent variable and each of the independent variables exists (although transformations can be applied to satisfy this assumption), equality of variance and that the errors of prediction are normally distributed around each and every predicted dependent value score [TF01]. The regression equation is given by:

$$\hat{Y} = A + B_1X_1 + B_2X_2 + ... + B_kX_k \tag{3.2}$$

where $\hat{Y}$ is the predicted value on the dependent variable, the X values represent the independent variables, the B values are the regression coefficients, representing the amount the dependent variable $\hat{Y}$ changes when the corresponding independent variable changes one unit; A is the intercept with the y-axis, representing the value of $\hat{Y}$ when all the independent variables are 0.

The most common method for fitting a regression line is the least-squares method. This method calculates the best-fitting line by minimising the difference between the sum of the squares of actual values Y and the predicted values $\hat{Y}$, that is $\sum(Y-\hat{Y})^2$ [Hin95].

Equality of variance tests check if the variability in scores on a continuous variable is approximately the same at all values of another continuous variable. Levene's test [Lev60] can be used to measure equality of variance and is more robust than other similar tests as it is not as sensitive to departures from normality [TF01]. Levene's test can consider the distances of the observations from their sample median rather than their sample mean making the test more robust for smaller samples. Given a variable X with a sample size of N, divided into g subgroups, where n_i is the sample size of the ith subgroup, the test statistic is given by:

$$F_L = \frac{[\frac{\sum_i n_i(z_{i.}-z_{..})^2}{g-1}]}{[\frac{\sum_i \sum_j (z_{ij}-z_{i.})^2}{\sum_i (n_i-1)}]} \tag{3.3}$$

where $z_{i.} = \sum_j \frac{z_{ij}}{n_i}$, $z_{..} = \sum_i \sum_j \frac{z_{ij}}{\sum_i n_i}$, $z_{ij} = |x_{ij} - \tilde{x}_i|$ and $\tilde{x}_i$ is the median of the ith subgroup.

The **Independent Samples T-test** compares the mean scores of two groups on a given variable. It is based on the assumptions that the dependent variable is normally distributed and that the two groups have approximately equal variance on the dependent variable. Once satisfied the t-test can be calculated using:

$$t = \frac{(\bar{x_1} - \bar{x_2})}{\sqrt{[\frac{\sum x_1^2 - \frac{(\sum x_1)^2}{n_1} + \sum x_2^2 - \frac{(\sum x_2)^2}{n_2}}{n_1+n_2-2}](\frac{1}{n_1}+\frac{1}{n_2})}} \tag{3.4}$$

where x_1 and x_2 are the values of the given variable for the first and second group and n_1 and n_2 are their respective sample sizes [Hin95].

The **Shapiro-Wilks test of normality**(W) tests the null hypothesis that a sample $x^1, ..., x^n$ comes from a normally distributed population and is recommended for small or medium sized samples [TF01]. The test is given by:

$$W = \frac{(\Sigma_{i=1}^{n} w_i X_i')^2}{\sum_{i=1}^{n}(X_i - \bar{X})^2} \tag{3.5}$$

where n is the number of observations, X is the original data, X' is the ordered data and $\bar{X}$ is the sample mean of the data. The constants w_i are given by:

$$w_1, ..., w_n = MV^{-1}[(M'V^{-1})(V^{-1}M)]^{-\frac{1}{2}} \tag{3.6}$$

where M denotes the expected values of standard normal order statistics for the sample and V is the corresponding covariance matrix. If W = 1 the given data is perfectly normally distributed. When W is significantly smaller than 1, the assumption of normality is not met.

Results

As the students who participated in the study were self-selected and the number of students who completed the survey was low, $n = 30$, it was important to determine if the sample was representative of the class. To this end *a priori* analysis was carried out to verify that no statistically significant difference existed between the mean overall module results of the class and the sample. Test assumptions on normality (Shapiro-Wilks test) and the equality of variance (Levine test) were performed and a t-test on the overall results, ($t(94) = 1.093, p = 0.277$), found no statistically significant differences between the mean scores of the class and the sample. Only one factor was found to have outlying values (that is instances with standardised (z) scores ± 3.29, $p < 0.01$, as measured by a two-tailed test [TF01]). The factor measured the number of hours per week students spend playing computer games while studying on the module, referred to in this study as WHILEGAMES. This factor was initially measured on a scale of 0 to 5, however, only two students were found to have a value greater than or equal to 3 and as

Factor	**r**	**n**
LC mathematics	0.46**	30
LC physics	0.59*	18
LC biology	0.75*	10
LC highest Science	0.48**	28
Comfort-level	0.55**	30
Perceived Understanding	0.76**	30
** Correlation is significant at the 0.01 level (2-tailed).		
* Correlation is significant at the 0.05 level (2-tailed).		

Table 3.1: *Factors identified as having a significant relationship with performance using Pearson Correlations*

such were detected as outlying values. Where the number of outlying values is small, a common solution is to change the outlying value so that it is still extreme but sufficiently reduces the value so it is no longer detected as an outlier [TF01]. Accordingly, the values for the two outliers on the WHILEGAMES factor were subsequently changed from 3 to 2 and were no longer detected as outliers.

In the remainder of this section the findings on the relationship between each of the factors and programming performance is presented, followed by an analysis of the combination of factors that best predict performance.

Factors significantly related to programming performance

A summary of the main significant relationships identified by Pearson Correlations are illustrated in Table 3.1. To establish the relationship between previous academic experience in mathematics and science, the achievable grades for each subject were ranked, with the highest rank given to the highest possible grade and the lowest rank given to the lowest possible grade. LC mathematics, LC physics

Factor	Values
Gender	Male, Female
Age	Under 23, 23+
Work style preference	Individual, Group
Encouragement	Yes, No
Part-time employment	Yes, No

Table 3.2: *Dichotomous values for personal factors*

and LC biology were found to have a statistically significant relationship with performance. Highest science result, which includes other less commonly studied science subjects, was also found to be statistically significant. Comfort-level, a likert-response measure, was also found to be a statistically significant indicator of performance and a strong significant relationship between perceived understanding (likert-scale question) and performance was found.

Factors where no relationship with programming performance was identified

Contrary to the positive relationship identified between LC physics, LC biology, and highest science result with programming performance no relationship was found between LC chemistry and performance. This is a surprising finding and may suggest that some science subjects are more important than others or that some underlying phenomena is affecting the results. Previous computer experience was measured by prior programming experience and previous non-programming computer experience. In both cases student responses were separated into those with previous experience and those without previous experience. T-tests for independent samples were used to examine the differences between the overall module results of each group. Before each t-test was carried out assumptions of normality and equality of variance were confirmed. No significant differences were found between students with or without previous programming experience or between students with or without non-programming computer experience with performance on the module. In addition, a weak correlation of $r = 0.31$, $p < 0.01$ was found between performance on the cognitive test and performance on the module.

Gender, age, work-style preference, encouragement by others and part-time employment were treated as dichotomous variables for analysis purposes and the possible values of each factor are given in Table 3.2. Students were grouped according to the responses they provided for each of the factors. T-tests for independent samples were used to examine if differences existed between the overall module results on each of the factor values, for example, the mean overall module result for male students was compared to the mean overall module result for female students. Assumptions of normality and equality of variance were again confirmed. In each instance, the t-tests revealed no statistically significant differences.

Regression modelling

To investigate whether the various factors studied were predictive of performance on the module a number of regression analyses were conducted. Two significant models emerged. The first was designed to determine the earliest indicators of programming performance. Consideration was given to gender, previous academic experience, cognitive test score, previous programming and non-programming computer performance, encouragement from others, work-style preference and hours working at a part-time job. Using a stepwise regression method a significant model emerged with $F(2, 27) = 7.113$, $p < 0.01$ with an adjusted R square = 30%. Significant values were found for: LC mathematics ($\beta = 0.390$, $p = 0.021$) and gender ($\beta = -0.368$, $p = 0.028$).

A second model included the predictors from the first model in addition to a students' comfort-level with the module and perceived understanding of how they are doing. Using stepwise regression, a significant model emerged with $F(4, 23) = 26.03$, $p < 0.001$, adjusted R square = 79%. Significant regression weights were found for: perceived understanding ($\beta = 0.505$, $p = 0.000$), gender ($\beta = -0.494, p = 0.000$), comfort-level ($\beta = 0.301$, $p = 0.022$), and LC mathematics ($\beta = 0.197$, $p = 0.047$).

3.1.4 Subsequent Analysis

Initial analysis of the data using statistical techniques commonly employed by computer science educators to predict programming performance provided valuable insights into the nature of the relationships between the independent factors and performance on the module. However, this research study is concerned with determining whether students can be regarded as 'weak' or 'strong' programming students and as such requires the application of suitable classification techniques. It was decided, therefore, to classify students who receive a mark below 55.5% as weak programmers and students with a result above this threshold as strong programmers. Although it would have been possible to use the actual pass mark as the threshold ($\sim$ 40%), this was not suitable as institutions can adjust student marks to attain this threshold using discretionary institutional policies. Additionally, students who pass the module but fail to achieve a mark considerably higher than the pass rate (for example 10% to 15% higher) should typically not be regarded as strong students. An analysis of the student cohort marks indicated that a threshold of $\sim$ 55.5% provided a more appropriate separation of strong and weak programmers.

Logistic Regression

Logistic regression is a statistical technique to predict a discrete outcome, such as group membership from a set of variables[2]. The variety and complexity of data sets that can be analysed is considerable. It makes no assumptions about the distributions of predictor variables, that is, the predictors do not need to be normally distributed, linearly related or have equal variance within each group and can be a mix of continuous, discrete and dichotomous variables. It is a particularly useful technique when the distribution of responses on the dependent variable is expected to have a non-linear relationship with one or more of the independent variables. The model produced by logistic regression is non-linear and is denoted by:

$$P_i = \frac{e^{b_O + b_i X_i}}{1 + e^{b_O + b_i X_i}}. \tag{3.7}$$

Like most statistical techniques, logistic regression makes a number of assumptions which must be satisfied. (1) The outcome variable must be discrete but a continuous variable can be converted to a discrete one. (2) Overfitting can occur when there are too few cases relative to the number of predictor variables. The models generated in this study attempt to keep the ratio of input variables to students low to avoid this problem. (3) Outliers can significantly affect results, however, as noted in Section 3.1.3 outliers were detected and appropriately handled. (4) Logistic regression assumes the absence of multicollinearity. Multicollinearity exists when the independent variables (predictors) are highly correlated ($r \geq 0.9$ and above) [Pal05]. Generation of correlation coefficients between the independent variables indicated that this assumption was satisfied in our study. (5) It also assumes that each response comes from a different, unrelated case (student). This assumption is satisfied by the nature of this study.

[2]Logistic regression is considered further in Chapter 4

Results

Logistic Regression analysis resulted in three significant models. A model (Model 1) incorporating LC mathematics and gender was able to classify 70% of students correctly. A second model (Model 2) which also incorporated perceived understanding correctly classified 90% of students. A third model (Model 3) which also considered the total score on comfort-level accurately classified 93% of students. Interestingly, when gender is omitted from the models and the number of hours students spend playing computer games is incorporated instead, 73% of students are classified correctly (Model 4). The number of hours playing games does not improve prediction accuracy when gender is also included and this suggests that playing games is related to gender. Subsequent analysis using a t-test confirmed that there is a statistically significant difference between the game playing of male and female students, with male students associated with playing more games. Interestingly, game playing is negatively associated with performance, that is, the more hours spent playing games the lower the performance. This suggests that in the absence of gender information the number of hours playing games can be used, albeit, with poorer results. Table 3.3 provides information on the overall accuracy of the models, the percentage of students correctly identified as weak and the percentage of students correctly identified as strong.

3.1.5 Discussion

The findings on the relationship between experience in mathematics and science subjects, and programming performance is in line with previous research findings. The strength of the correlations between LC physics scores, LC biology scores and programming performance is interesting, and would suggest that science in general has a significant influence on performance. However, the lack of corre-

Model	**% Weak Students Correctly Classified**	**% Strong Students Correctly Classified**	**% Overall Correctly Classified**
Model 1	79%	55%	70%
Model 2	95%	82%	90%
Model 3	100%	82%	93%
Model 4	90%	46%	73%

Table 3.3: *Percentage of students correctly classified as weak or strong as well as overall classification accuracy achieved by the logistic regression models ($n = 30$)*

lation with LC chemistry appears contradictory and given the small sample size further research is required. Like the Cantwell-Wilson and Shrock study [CWS01], comfort-level was found to be highly correlated with programming performance. The most significant finding however, is the very strong correlation between students' perception of their understanding of the programming module and their programming performance. As this study was carried out in the second semester, further research to identify the point in time that perception of module understanding becomes such a reliable indicator is important. If a similarly high correlation can be found early on in the module then it would be very powerful in diagnosing and subsequently assisting struggling students. Although previous research has found previous programming experience and non-programming computing experience to be indicators of success, no such relationship was found here. This may be partially accounted for by the fact that students cannot study programming or application software for national examination in secondary schools in Ireland. The relationship between performance on the cognitive test and performance on the module was found to be weak, however, subsequent analysis found that a number of items in the test were highly correlated with programming performance. A redesign of the test could result in more significant findings in the future.

Although only some factors were found to have a relationship with programming performance, given the limited sample size in the pilot study it would be hasty to exclude any of the factors from the main study. Therefore, it was decided that all factors would be re-tested in the main study with the exception of the cognitive test. Given the weak correlation found between performance on the module and the cognitive test as well as the existence of several previous studies with findings of a similar weak relationship and the large amount of work that would be necessary to re-design the test it was decided to omit the test from future studies. However, it was decided that given the positive findings on self-regulated learning (SRL) in other academic domains, as outlined in Section 2.5, that the main study should include an instrument to measure SRL.

3.2 Main study

Based on the results of the pilot study and on the literature review outlined in Chapter 2 a detailed study on factors that could influence programming performance was carried out in the academic year 2004-2005. This study is referred to as the 'main study' in this thesis. In this section the methodology employed in the main study is documented. Multiple institutions were involved and a description of the participants and the introductory programming modules they were taking is provided. The instruments developed and employed in this study are described and the section concludes with a description of the *a priori* procedures carried out to prepare the gathered data for analysis.

3.2.1 Participants

The study was carried out at four third-level institutions (post high-school) in the Republic of Ireland in the academic year 2004-2005. A-hundred and twenty-three students enrolled in a first year introductory programming module at the facilitating institutions voluntarily participated in this study. The institutions involved are referred to in this thesis as Institute A, Institute B, Institute C and Institute D and the actual institute names are provided in Table 3.4. The institutes were quite different in that one was a university, two were institutes of technology and one was a college of further education. The overall aim of each module was to provide students with introductory programming skills and the contents of each module were highly similar. An overview of each of the modules is given in Table 3.5. The measure of performance used in this study was the overall module mark. As with the pilot study, students were provided with an information sheet about the research and signed a consent form agreeing to participate. Permission to carry out the research was also granted by each of the participating institutions.

Institute ref.	**Institute name**
Institute A	National University of Ireland Maynooth
Institute B	Institute of Technology Blanchardstown
Institute C	Institute of Technology Carlow
Institute D	Whitehall College of Further Education

Table 3.4: *Participating Institutes*

3.2.2 Instruments

Five instruments were used to collect data: a background questionnaire, the Cantwell-Wilson and Shrock comfort-level measure, a programming self-esteem

Institute	**Language**	**Concepts covered***	**Assessment structure**
Institute A	Java	Variable types, selection statements, iteration, recursion, arrays, methods, sorting, searching, classes and objects.	30% continuous assessment, 70% final examination.
Institute B	Java	Variable types, selection statements, iteration, methods, classes and objects, introduction to applets .	50% continuous assessment, 50% final examination.
Institute C	Pascal	Variable types, selection statements, iteration, arrays, searching, sorting, linked lists and pointers.	40% continuous assessment, 20% practical examination, 40% final examination.
Institute D	VB	Variable types, selection, iteration, arrays, methods, classes and objects.	100% project.
Institute D	Java	Variable types, selection, iteration, arrays, methods, classes and objects.	2 x 30% assignments, 40%theory examination.
* *Not in order*			

Table 3.5: *Module Overview*

questionnaire, a self-efficacy questionnaire and a motivation and learning strategies questionnaire. Copies of each of these instruments are provided in the Appendices.

The background questionnaire collected data on a number of items including previous academic information, for example, Leaving Certificate (LC) mathematics grade, highest LC science grade; prior programming and non-programming computer experience, and various miscellaneous items, including, number of hours playing games before and during the module, number of hours per week working at a part-time job.

A questionnaire on comfort-level based on a larger set of questions used in a study by Cantwell-Wilson and Shrock [CW00] was also employed. The modified questionnaire focussed on the same issues as the original questionnaire but was re-structured so that it could be completed in a shorter length of time. The questionnaire is referred to as the 'Cantwell-Wilson and Shrock [CW00] comfort-level measure' in this thesis and was composed of nine questions that examined a student's perception of their level of understanding compared to the rest of the class (one question), their ease at asking and answering programming questions (five questions), their general understanding of programming concepts, their ability to design the logic of a program without help and complete assignments (three questions).

The Rosenberg Self-Esteem (RSE) questionnaire ([Ros65]) was adapted to apply to programming self-esteem. The RSE scale is perhaps the most widely used self-esteem measure in social science research. The scale consists of 10 questions and has been shown to have generally high inter-item and test-retest reliability. Each of the questions were modified to relate to programming self-esteem and not to self-esteem directly, for example the first question was changed from 'On the whole, I am satisfied with myself' to 'On the whole, I am satisfied with my programming progress'.

The Computer Programming Self-Efficacy Scale [RW98] consists of 33 items that ask students to judge their capabilities in a wide range of programming tasks and situations. As this instrument was administered when students had very limited experience of the programming module, a shortened version of this scale using only seven questions about simple programming tasks was used.

In this study we employed the model of self regulated learning developed by Pintrich and his colleagues, as outlined in [PG91]. This model stresses the learner's use of cognitive strategies and self-regulatory strategies, self-efficacy beliefs (individuals' judgements of their capabilities to perform a task), task value beliefs (the importance of, interest in and value associated with a task) and goal orientation (intrinsic goal orientation and extrinsic goal orientation) [Pin99]. The Motivated Strategies for Learning Questionnaire (MSLQ), a self-report instrument designed by Pintrich *et al.* to measure students' motivation and self-regulated learning in classroom contexts, was used to measure SRL [PSGM91]. To examine the components of SRL the following scales were employed:

- Intrinsic and extrinsic goal orientation scales,
- Task value scale (a student's perceptions of the course material in terms of interest, importance and utility),
- Cognitive strategy usage as measured by a rehearsal strategies scale, elaboration strategies scale and organization strategies scale,
- Meta-cognitive strategy usage as measured by a planning, monitoring and regulating strategies scale
- Self-efficacy for learning and performance scale (incorporates two aspects of expectancy: expectancy for success and self-efficacy).

Data was collected in two study administrations. In the first administration all surveys (background questionnaire, Cantwell-Wilson and Shrock comfort-level measure, programming self-esteem questionnaire, and the Computer Programming Self-Efficacy scale), except the MSLQ, were administered. The first administration was carried out early in the programming module (when the students had completed very early programming concepts - typically variable types, selection statements and sometimes iteration) while the second administration was completed when they were on average one third of the way through the material. It was the intention that both administrations would be completed closer together but this was not possible due to timetabling and other constraints.

3.2.3 Data Pre-Processing

A number of *a priori* procedures were put in place to prepare the gathered data for analysis. The procedures included (1) data screening, (2) testing the representativeness of the sample, (3) missing data analysis and (4) tests of unidimensionality.

Data screening required the examination of encoded data to ensure that it was free of coding errors. Maximum and minimum frequency values were inspected to check that no out-of bounds entries existed. As a more rigorous measure each encoded item was inspected along with totals, by an independent witness and the author, to ensure that all data had been satisfactorily entered and computed.

An *a priori* analysis was carried out to verify no significant differences existed between the mean overall module results of the samples and the total student (introductory programming) population. A t-test confirmed that no significant differences existed between the mean results of students who participated in the first administration of the study and the relevant student population at each institute. However, statistical differences were found between the students who participated in the motivation section ($t(67) = 6.451$, $p = 0.001$) and the learning strategies section ($t(56) = 7.1$, $p = 0.001$) of the MSLQ at institute A and this will have to be taken into account in the analysis. The cause of this statistical difference was a considerably reduced sample size on the second administration at institute A.

Statistical analysis was carried out to determine whether data was missing completely at random (MCAR) or missing at random (MAR) [TF01]. Missing data in both of these instances is generally regarded as ignorable. Independent t-tests indicated that there was no significant difference between the mean module result on any of the first administration factors between students with no missing values and students with missing values except for programming self-esteem, ($t(121) = 3.088$, $p = 0.008$). However, further analysis confirmed that for 7 of the 8 students with missing data, a reason could be found to explain why the data was missing: 3 students were never given the programming self-esteem scale to complete, 3 students failed to complete one of the ten items on the scale and was thus omitted, and 1 student selected the same value for each item on the scale, even for reversed items and their response was deemed invalid. Therefore, only one student's failure to respond is unknown. Given that so few cases were missing, it was reasonable to drop the cases from analysis rather than inferring values which could result in distortions of association and correlation. One question on the background survey concerning the likely number of hours a student would spend studying for the module was omitted from the questionnaire administered at institute C and institute D and therefore 85 students completed the question. A t-test revealed that there was no significant difference between the mean results of students with missing values and without missing values. Only 77 students completed the MSLQ motivation scales. There are two main reasons for this reduced sample size. First, at institute A only 55% of students who completed the first part completed this section. At this institute students had to collect the survey and return it upon completion. The effort involved appeared to deter them. Second, due to timetabling constraints it was not possible to administer the motivation scales at institute C. Similarly, only 82 students completed the MSLQ learning strategy scales. This again was caused by the reduced participation at institute A and although the MSLQ learning strategy scales were administered at institute C only 52% of the original number of participants completed this section. The latter was caused by absenteeism on the day of administration. A t-test found that statistically significant differences existed between the mean results of students who completed the motivation scales and students who did not at institute A, ($t(54) = -4.93$, $p = 0.001$). Similar differences were found on the learning strategy scales at both institute A ($t(54) = -5.541$, $p = 0.001$) and institute C ($t(17) = -2.828$, $p = 0.012$). The findings here are more problematic especially given the earlier results on the representativeness of the sample at institute A for these two parts. A number of options are available to deal with the missing data, including dropping cases from analysis or substituting a value for the missing cases. The approach taken was to perform two separate investigations, the first using data gathered in the first administration and the second on the data gathered in both administrations. Interpretation of the second investigation will need to take into account the reduced sample size and the lack of representativeness.

Where multiple indicator variables were used to measure a construct, tests of unidimensionality were performed. Cronbach's alphas for each of the MSLQ subscales and the subsequent values calculated in this study are given in Table 3.6. In each instance, the alpha values were found to be high. Test of reliability for the Cantwell-Wilson and Shrock comfort-level measure was 0.80 and for the shortened Computer Programming Self-Efficacy Scale was 0.949. Typically, Cronbach's alpha for the Rosenberg Self-Esteem scale are in the range of 0.82 to 0.88, ([Ros65]) and for this study the alpha value was 0.91.

Scale	[PSGM91]	Study values
Intrinsic goal orientation scale	.74	.75
Extrinsic goal orientation scale	.62	.56
Task value scale	.90	.85
Self-Efficacy for learning and performance	.93	.95
Rehearsal scale	.69	.73
Elaboration scale	.76	.58
Organization scale	.64	.63
Planning, monitoring and regulating scale	.79	.83

Table 3.6: *Reliability analysis using Cronbach alpha measure for MSLQ scales as given by Pintrich et al and as found in this study.*

3.3 Summary

In the academic year 2003-2004 a pilot study was carried out at the Department of Computer Science, NUI Maynooth, to determine factors that influenced success on an introductory programming module. An initial examination of the data followed the norm of previous studies and used correlation and linear regression to analyse the data. The results indicated that gender, LC mathematics, perceived understanding and comfort-level were important indicators of programming success. Subsequent analysis using logistic regression, which allows our research question to be directly examined resulted in three significant models. The first model incorporated LC mathematics and gender and correctly classified 70% of students as ‘weak’ or ‘strong’. The second model incorporated perceived understanding as well and classified 90% of students correctly. A third model which also considered the total score on comfort-level accurately classified 93% of students. In addition, a relationship was found between gender and the number of hours spent playing computer games. Omitting gender from the third model and including the number of hours students spend playing computer games instead resulted in 73% of students correctly classified.

A detailed description of the research methods employed in the main study, carried out in the academic year 2004-2005, was also presented. The students participating in the study were described and descriptions of the introductory programming modules investigated were outlined. The instruments used to gather data were documented and finally, the procedures used to prepare the data for analysis were outlined including data screening, testing the representativeness of the sample, missing data analysis and tests of unidimensionality.

Chapter 4

Machine Learning Techniques

Initial linear regression modelling, as outlined in Chapter 3, was valuable in identifying factors that can be used to classify students as weak or strong programmers. Learning to classify accurately is a common problem in machine learning and data analysis. Many machine learning algorithms have been proposed for classification and in this chapter, six different types of algorithms are evaluated and reviewed. The algorithms evaluated are k-nearest neighbour, C4.5, naïve Bayes, logistic regression, support vector machines and backpropagation networks.

4.1 Nearest Neighbour Learning

Nearest neighbour is a supervised (learning from examples) non-parametric method. Non-parametric modelling is different to parametric modelling in that the model structure is not specified *a priori*, but is instead determined from the data. To classify an unknown pattern, the class of the pattern nearest the unknown pattern is selected using a distance metric where all instances are assumed to correspond to points in the n-dimensional space $\Re^n$. With this approach, learning is directly based on the training examples and generalisation is only performed when a new instance must be classified. As no learning takes place until the classification stage this type of learning is often regarded as 'lazy' learning. Typically, nearest neighbours are determined using standard Euclidean distance, as given in Equation 4.1 [Mit97], where one instance has attribute values $a_1^{(1)}, a_2^{(1)}, \ldots, a_n^{(1)}$ (where n is the number of attributes) and the other instance has values $a_1^{(2)}, a_2^{(2)}, \ldots, a_n^{(2)}$.

$$\sqrt{(a_1^{(1)} - a_1^{(2)})^2 + (a_2^{(1)} - a_2^{(2)})^2 + \ldots + (a_n^{(1)} - a_n^{(2)})^2}. \tag{4.1}$$

If the Euclidean distance formula is used directly on attributes that are measured on different scales, the effects of some attributes might be completely overshadowed by other attributes with larger scales of measurement. Consequently it is usual to normalise all attribute values to lie between 0 and 1. One such approach to normalise attribute values is given by:

$$a_i = \frac{v_i - min(v_i)}{max(v_i) - min(v_i)} \tag{4.2}$$

where v_i is the actual value of attribute i and the maximum and minimum values are calculated over all instances in the training set [WE05].

An extension to this approach is to choose the majority class of k nearest neighbours (K-NN) as illustrated in Figure 4.1.

4.1.1 Evaluation of Nearest Neighbour Learning

K-NN has many advantages as a classifier. For example, it is relatively straightforward to implement and requires no training time once the training set is stored [WE05], [RP03]. However, there are a number of drawbacks to this method. As each training example is considered for classification K-NN can be slow and has high storage costs. This can be improved by eliminating redundant samples in stable regions of attribute space. To reduce the computation and time required to classify a new example, techniques such as *kd*-tree can be implemented for

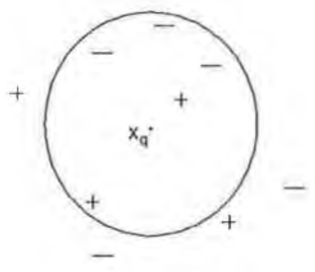


Figure 4.1: *k-nearest neighbour example: In the figure positive training examples are illustrated using + and negative training examples are shown by −. The choice of k is critical, in that, the query instance x_q will be classified as positive using a k value of 1 but as negative using a k value of 5*

indexing the stored training examples [WE05] [Mit97]. The value selected for k is very important. If the neighbourhood is too small it won't contain any data points but if it is too large it may contain all the data points. For fixed k, the size of the neighbourhood varies - where data are sparse, the neighbourhood is large but where data are dense the neighbourhood is small [RP03]. K-NN is sensitive to high dimensionality amongst the attributes. In high dimensional spaces with many irrelevant attributes for classification, instances that are similar on the relevant attributes may be distant from each other in the large dimensional space, however previous studies have found that selecting $k \geq 3$ helps to eliminate this [RP03]. A refinement of K-NN is to weight the contribution of each of the nearest neighbours according to their distance to the new example, giving greater weight to the closer neighbours. Such an approach tends to be more robust to noise but with the trade-off that the classifier will run more slowly [Mit97].

4.2 Decision Tree Learning

Decision trees are one of the most widely used learning methods and have been successfully applied to a wide range of tasks [Mit97]. New instances are classified by sorting them down the tree from the root node according to the values of the attributes tested in successive nodes. Each branch descending from a node corresponds to one of the possible values for this attribute. The process continues until a leaf node is reached which provides the classification of the instance [RP03], [Mit97], [WE05]. A sample decision tree is given in Figure 4.2.

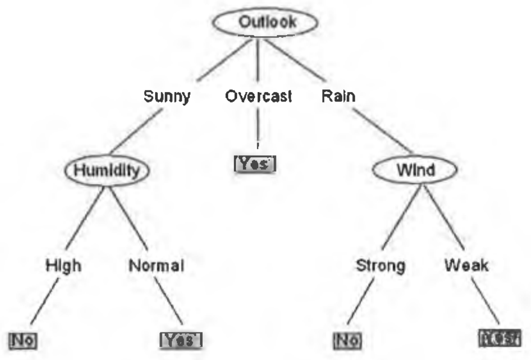


Figure 4.2: *Sample decision tree: ovals represent attribute nodes, branches correspond to one of the possible attribute values and squares represent classification values. In this example the target is to decide to play tennis or not based on the weather.*

To construct a decision tree, an attribute is selected to place at the root node and a branch for each possible value of the attribute is created. This divides the example set into subsets, one for every value of the attribute. The process can be repeated recursively for each branch using only those instances that actually reach the branch. If at any time all instances at a node have the same classification, no further development of that part of the tree is required [WE05].

Fundamental to the development of any decision tree is the selection of an optimal way to split the data, that is, selecting the attribute that is most useful for classifying examples. C4.5 is a commonly used decision tree generating algorithm, based upon ID3 [WE05]. It recursively visits each decision node, selecting the optimal split, until no further splits are possible. C4.5 uses the concept of entropy reduction (information gain) to determine the optimal split. Entropy reduction measures how well a given attribute separates the training examples according to their target classification and is based on the concept of entropy, a measure from *Information Theory* that describes how much information is carried by a signal [RP03]. In a binary classification problem, the entropy of a set X, is calculated by:

$$H(X) = -p_p log_2(p_p) - p_n log_2(p_n) \tag{4.3}$$

where p_p is the proportion of positive examples in X and p_n is the proportion of negative examples in X. C4.5 uses entropy as follows: given that a possible split, S, exists, which partitions the training data set T into several subsets, $T_1, T_2, ...T_k$, the entropy of S can be calculated as the weighted sum of the entropies for the individual subsets as follows:

$$H_s(T) = \sum_{i=1}^{k} P_i H_s(T_i) \tag{4.4}$$

where P_i represents the proportion of records in subset i. *Information Gain* measures the expected reduction in entropy caused by partitioning the training data T according to this candidate split S and is denoted by $gain(S) = H(T) - H_s(T)$. At each decision node, C4.5 chooses the split with the greatest information gain [Lar05].

4.2.1 Evaluation of Decision Tree Learning

Decision tree learning is suited to problems where instances are represented by attribute-value pairs and the target function has discrete output values [Mit97]. Decision trees are relatively simple to understand and interpret, and data preparation is straightforward. The presence of correlated attributes generally does not affect the accuracy of a decision tree. Generally once an attribute has been chosen, the correlated attribute will not be selected as there would be no information gain in doing so. They have also been shown to have fast performance even on large datasets [WE05].

Decision Trees can suffer from overfitting, but a number of approaches exist to avoid this. These include: (1) stop growing the tree before it reaches the point where it perfectly classifies the training data, and (2) allow the tree to overfit the data then post-prune the tree. The second approach has been found to be more successful as it can be difficult to decide when to stop growing the tree [WE05].

C4.5 is not restricted to binary splits like other decision tree algorithms such as CART [Lar05], and can thus produce a tree of variable shape. It includes several improvements over its predecessor ID3, including methods for dealing with continuous attribute values, missing values, noisy data and generating rules from trees. If the attribute is numeric, the test at a node usually determines whether its value is greater or less than a predetermined constant, giving a two-way split. Alternative splits can also be used. A numeric attribute is often tested several different times in any given path down the tree from root to leaf, each test involving a different constant [WE05] [Lar05].

4.3 Naïve Bayes

Bayes' theorem provides a way to calculate the probability of a hypothesis based on its prior probability. Assume a hypothesis space H and some observed training data D exist. Then $P(h)$ is the probability that hypothesis h is true, that is, it is the prior probability of h. $P(D)$ is the prior probability of the training data D. $P(D|h)$ is the probability of observing data D given that the hypothesis h holds. Bayes' theorem provides a way to calculate the posterior probability (conditional probability) $P(h|D)$ from prior probability $P(h)$, together with $P(D)$ and $P(D|h)$ and is given by:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}. \tag{4.5}$$

To determine the most probable hypothesis given the observed training data D, the maximum *a posteriori* (MAP) hypothesis is selected. MAP is determined by using Bayes' theorem to calculate the posterior probability of each hypothesis and given by:

$$\begin{aligned} h_{MAP} &= argmax_{h \in H}\, P(h|D), \\ &= argmax_{h \in H}\, \frac{P(D|h)P(h)}{P(D)}, \\ &= argmax_{h \in H}\, P(D|h)P(h). \end{aligned} \tag{4.6}$$

In the final step above $P(D)$ is dropped as it is a constant independent of h [Mit97].

Naïve Bayes is a non-parametric probabilistic model based on Bayes' theorem [MST94]. It makes the assumption that the effect of a variable value on a given class is independent of the values of other variables. This assumption is referred to as conditional independence among variable attributes. The assumption of independence makes it much easier to estimate these probabilities since each attribute can be treated separately, resulting in Equation 4.7.

$$v_{nb} = argmax_{v_j \in V} P(v_j) \prod_{i=1..k} P(a_i|v_j). \tag{4.7}$$

The number of $P(a_i|v_j)$ terms that must be estimated from the training data is equal to the number of attribute values multiplied by the number of target values which is a considerably smaller number than if the assumption of conditional independence was not made [MST94]. Although this assumption is often violated, naïve Bayes classifiers have been shown to work surprisingly well and in some cases out perform more sophisticated methods.

4.3.1 Evaluation of Naïve Bayes

As a classifier, naïve Bayes has many advantages. Unlike many other machine learning techniques, it can handle unknown or missing values [MST94]. It is particularly well suited when the dimensionality of the inputs is high and can handle both continuous or categorical data. It has often been shown that naïve Bayes outperforms more sophisticated algorithms [WE05] particularly when the attributes are in fact conditionally independent given the class [MST94].

A number of difficulties exist in implementing a naïve Bayes classifier. A significant computational cost can be incurred to determine the Bayes optimal hypothesis (linear in the number of possible hypotheses) [Mit97]. As all attributes are treated as equally important and independent of one another, given the class, naïve Bayes can suffer from redundant attributes. For example, if two attributes represent the same underlying phenomena then the probability of this attribute would be squared, giving it a great deal of influence in the final decision [WE05]. This can be reduced by using a careful selection of the most suitable attributes, for example, by using the regression techniques described in the previous chapter.

4.4 Logistic Regression

Logistic regression is a statistical technique that builds a linear model based on a transformed target value [WE05]. It is typically used to predict a dichotomous outcome but can also be used to predict multi-class outcomes. The independent variables can be continuous, discrete, dichotomous, or a mix of any of these [TF01].

Representing a dichotomous predictor variable as 0 or 1 can be advantageous as the mean of such a distribution is equal to the proportion of cases with a value of 1 in the distribution and can be interpreted as a probability. Intuitively, one might consider using multiple linear regression with a binary outcome but this approach has a number of problems. Probabilities have a maximum value of 1 and a minimum value of 0. But a linear regression line can extend upwards towards $+\infty$ and downwards towards $-\infty$ as the value of an independent variable increases or decreases. Thus the model could give probability values less than 0 or greater than 1 [Pam00]. In addition, assumptions of linear regression are violated in that the error residuals cannot be normally distributed or have equal variances if there are only two outcome variables [Men01].

An alternative technique favoured for modelling the problem due to its inherent simplicity is to use the logistic or logit transformation. The logit transformation involves two steps given a probability P_i of experiencing an event and a probability $1 - P_i$ of not experiencing an event with two possible outcomes. First the odds of experiencing an event are calculated using:

$$O_i = \frac{P_i}{1 - P_i}. \tag{4.8}$$

Subsequently the natural log of the odds (logged odds) is calculated, as follows:

$$L_i = \ln[\frac{P_i}{1 - P_i}]. \tag{4.9}$$

A linear relationship between the independent variables and the logit transformation can then be computed:

$$\ln[\frac{P_i}{1-P_i}] = b_0 + b_i X_i. \tag{4.10}$$

Expressing the probability rather than the logit gives:

$$\frac{P_i}{1-P_i} = e^{b_O + b_i X_i}. \tag{4.11}$$

After some mathematical manipulation the standard representation for logistic regression is produced [Pam00]:

$$P_i = \frac{e^{b_O + b_i X_i}}{1 + e^{b_O + b_i X_i}}. \tag{4.12}$$

Maximum Likelihood Estimation (MLE) is used to determine the best coefficient values. The goal of MLE is to determine parameters that most likely give rise to the pattern of observed examples [Men01] and uses the log likelihood function:

$$\ln LF = \sum [Y_i * \ln P_i] + [(1 - Y_i) * \ln(1 - P_i)] \tag{4.13}$$

Typically the procedure employed is to iterate through sets of coefficients seeking larger log likelihoods that better fit the observed data. The process stops when the increase in the log likelihood function from new coefficients is so small that little benefit comes from continuing further [Pam00].

4.4.1 Evaluation of Logistic Regression

Unlike discriminant function analysis, a closely related statistical technique, with logistic regression the predictor variables do not need to be normally distributed, linearly related or have equal variance within each class. It has the capacity to analyse different predictor types, including continuous, categorical and discrete variables, and can handle highly complex data sets. However, logistic regression has a number of weaknesses that must be handled accordingly. For example, it is sensitive to outliers and to multicollinearity amongst the independent variables [WE05], [TF01].

4.5 Support Vector Machines

Support Vector Machines (SVMs) were first introduced in the late 1970's and have received significant attention over the past 15 years. SVMs have been shown to have either equivalent or significantly better generalisation performance than other competing methods on a wide range of classification problems [Bur98]. SVM's are capable of classifying both linearly separable and non-linearly separable data and the techniques involved are outlined in this section.

4.5.1 Linear Classification

Given a training dataset $\{(\mathbf{x_1},\mathbf{y_1}),(\mathbf{x_2},\mathbf{y_2}),...,(\mathbf{x_m},\mathbf{y_m})\}$ where $\mathbf{x_i} \in \Re_\mathbf{d}$ and $y_i \in \{\pm 1\}$, a hyperplane which can successfully separate positive examples from negative examples is desired. The SVM base algorithm will find the optimal hyperplane. This is achieved by re-scaling the hyperplane $H : y = \mathbf{w} \cdot \mathbf{x} + \mathbf{b} = \mathbf{0}$, where $\mathbf{w}$ is normal to the hyperplane and $\| \mathbf{w} \|$ is the Euclidean norm of $\mathbf{w}$, such that

$$\begin{aligned} H_1 &: y = \mathbf{w} \cdot \mathbf{x} + \mathbf{b} = +\mathbf{1} \\ H_2 &: y = \mathbf{w} \cdot \mathbf{x} + \mathbf{b} = -\mathbf{1}. \end{aligned} \tag{4.14}$$

That is, the following constraint must be satisfied:

$$|\mathbf{w} \cdot \mathbf{x} + \mathbf{b}| \leq -\mathbf{1}\,\forall\,\mathbf{i} \tag{4.15}$$

This can be combined into:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1\,\forall\,i \tag{4.16}$$

Let $\mathbf{x}^-$ be any point on the minus plane as given by H_2, and let $\mathbf{x}^+$ be the closest point on the plus plane as given by H_1 to $\mathbf{x}^-$, such that a line from $\mathbf{x}^+$ to $\mathbf{x}^-$ is perpendicular to the planes, as illustrated in Figure 4.3. The margin width, λ, is equal to $|\mathbf{x}^+ - \mathbf{x}^-|$ and can be calculated by projecting the vector $(x^+ - x^-)$ onto the vector normal to the hyperplane $\frac{\mathbf{w}}{||\mathbf{w}||}$, i.e.

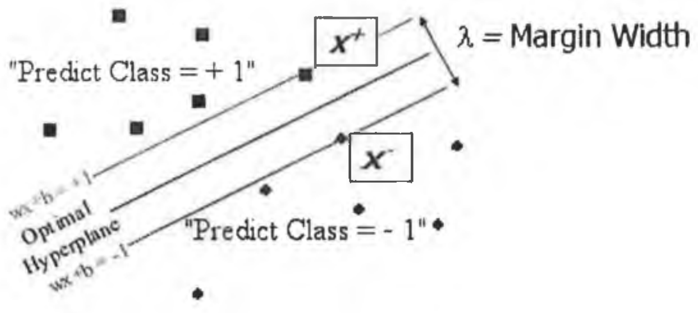


Figure 4.3: *SVM solution: the SVM algorithm will find the optimal hyperplane, that is, the hyperplane which maximizes the margin, to separate the positive and negative examples. x^+ and x^- are support vectors*

$$\lambda = (x^+ - x^-) \cdot \frac{\mathbf{w}}{||\mathbf{w}||} = \frac{2}{||\mathbf{w}||} \tag{4.17}$$

Maximising the margin is equivalent to minimising

$$\lambda = \frac{1}{2}||\mathbf{w}||^2 \tag{4.18}$$

subject to Equation 4.16. The solution for a typical two dimensional case is illustrated in Figure 4.3. The training examples lying on H_1 and H_2 (x^+ and x^- respectively) are called support vectors. All other examples could be removed without changing the solution. Thus, the problem is a constrained optimisation problem which can be solved using the standard Lagrangian approach [AR94], which results in:

$$W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \tag{4.19}$$

subject to the constraints

$$\alpha_i \geq 0 \quad \forall i \qquad and \qquad \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{4.20}$$

The derivation of Equation 4.19 can be found in [SS02]. Equation 4.19 is a quadratic optimisation problem and many effective robust algorithms exist for solving such problems.

The α_i values that are non-zero correspond to the support vectors. If $\alpha_i = 0$ then these points make no contribution to the decision function:

$$f(\mathbf{z}) = sign\left(\sum_{i=1}^{m} y_i \alpha_i (\mathbf{x}_i, \mathbf{z}) + b\right). \tag{4.21}$$

4.5.2 Soft Margins

Enforcing the maximal separation between the hyperplane and the closest points generates a solution known as a *hard margin classifier*. However, a separating hyperplane may not exist and even if it does, it is not always the best solution as an individual outlier in a data set can crucially affect the hyperplane. An algorithm that is robust to a certain fraction of outliers might be preferable. An approach which has been shown to work well is to incorporate a non-negative slack or error variable $\xi_i \geq 0$, where $i = 1, \ldots, m$ such that Equation 4.18 becomes:

$$\lambda = \frac{1}{2}||\mathbf{w}||^2 + C\Sigma_{i=1}^{m}\xi_i \tag{4.22}$$

In Equation 4.22 it can be seen that the constant $C > 0$ determines the trade-off between margin maximisation and training error minimisation. Rewriting in terms of Lagrange multipliers this leads to the problem of maximising Equation 4.19 subject to Equation 4.23:

$$0 \leq \alpha_i \leq C \, \forall \, i \quad and \quad \Sigma_{i=1}^{m}\alpha_i y_i = 0. \tag{4.23}$$

Thus the only difference from the hard margin classifier previously described is that we now have an upper bound of C [Cam02].

4.5.3 Non-Linear Classification

Consider the classification function shown in Figure 4.4(a). A linear SVM as defined earlier is unable to solve this problem. However, the data can be transformed to a higher dimensional space such that the data points will be linearly separable. The higher dimensional space is referred to as the *feature space* F. The effect of mapping non-linearly separable data to feature space can be seen in Figure 4.4(b) and is denoted by Equation 4.24.

$$\Phi : \Re^N \rightarrow F. \tag{4.24}$$

Suppose a 'kernel function' K exists, such that $k(\mathbf{x_i}, \mathbf{x_j}) = (\Phi(\mathbf{x_i})\Phi(\mathbf{x_j}))$. That is, the dot product in the high dimensional space is equivalent to a kernel function of the input space. It is not necessary to be explicit about the transformation given that $k(\mathbf{x_i}, \mathbf{x_j})$ is equivalent to the dot product of some other high dimensional space. If a function satisfies Mercer's condition it can be considered a valid kernel function [Bur98].

When the non-linear mapping is introduced modifications to the objective function are required:

$$W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{4.25}$$

subject to the constraints defined by Equation 4.20. Once an optimal solution is found, the decision function for a new point $\mathbf{z}$ is given by

$$f(\mathbf{z}) = sign\left(\sum_{i=1}^{m} y_i \alpha_i K(\mathbf{x}_i, \mathbf{z}) + b\right). \tag{4.26}$$

Several kernel functions exist including:

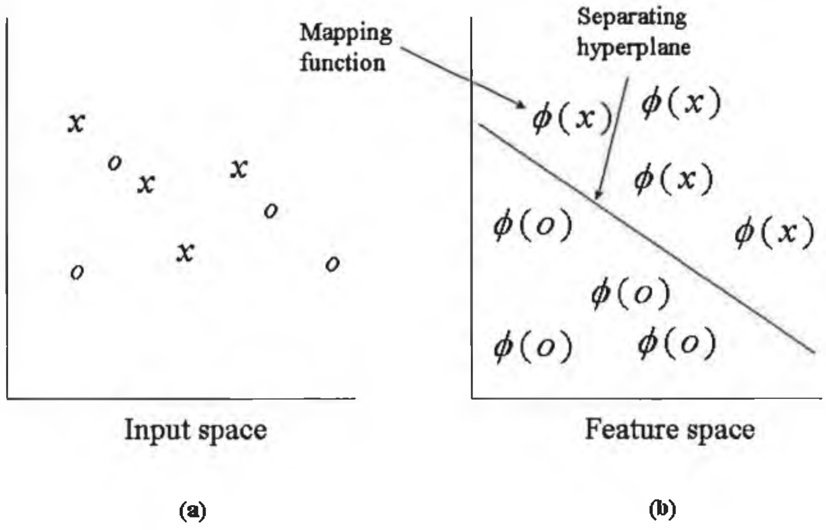


Figure 4.4: *The diagram on the left portrays the original datapoints. It can be seen that the xs can not be linearly separated from the os. Each datapoint is projected into the feature space using the mapping function ϕ and the two sets of data become linearly separable.*

- Polynomial: $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^d$
- Radial Basis Function: $K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma||\mathbf{x}_i - \mathbf{x}_j||^2)$, $\gamma > 0$.

4.5.4 Evaluation of SVMs

SVMs have been shown to have good generalisation performance. Linear SVMs have very fast execution times and do not require any parameter tuning (except the constant C when soft margins are used). In addition, SVMs do not suffer from local minima and can cope well when the relative size of the number of training examples within each class is very different. Since SVMs do not directly try to minimise error, but try to separate the patterns in high dimensional space, the result is that SVMs are fairly insensitive to the relative numbers of each class. For instance, new examples that are far behind the hyperplanes do not change the support vectors.

One of the biggest limitations of the support vector approach is the choice of kernel and consequently the kernel function parameters. The best choice of kernel is still under debate. Even when a particular kernel is chosen, the kernel function parameters must then be selected. A second limitation is that training times can be high if there are large numbers of training examples and execution time can be slow for non-linear SVMs. Finally, from a practical viewpoint, SVMs have a high algorithmic complexity [Bur98], [SS02].

4.6 Backpropagation Networks

Artificial neural networks (ANNs) are simplified models of the central nervous system. Networks are composed of highly interconnected nodes connected by directed links. Each of the links has a numeric weight associated with it which determines the strength of the connection. Typically, the network is presented with sets of correctly classified training examples and they learn to recognise and classify other instances of these classes. The output of the network is determined by passing the sum of the weighted inputs and a term known as the bias to a differentiable transfer function. This function is a mathematical representation of the relationship between input and output. Learning consists of adjusting the weights in the network to minimise error using a learning algorithm [Pat96], [Mit97], [Haw99]. Backpropagation (BP) networks are one of the most commonly used neural networks and have been applied successfully to complex classification problems.

A BP network is typically a layered feedforward neural network as illustrated

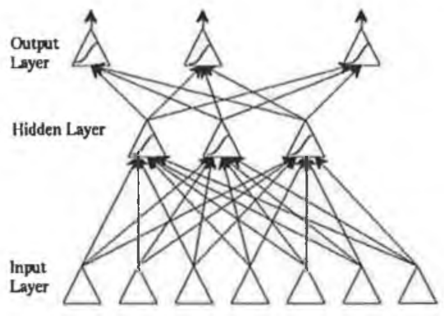


Figure 4.5: *Backpropagation Network: in this example the network is composed of three layers: an input layer, a hidden layer and an output layer. Signals propagate from the input layer to the output layer in a forward direction*

by Figure 4.5. In feedforward networks the input signal propagates in a forward-direction through the layers of the network. It is composed of an input layer, any number of hidden layers and an output layer. Connections within a layer are not allowed but connections can skip intermediate layers. Learning consists of two passes through the layers of the network: a forward pass and a backward pass. In the forward pass an activity pattern, that is, an input vector is applied to the units in the network and its values propagates through the network layer-by-layer until an output (response) is produced by the network. During the backward pass, the weights on the connections are adjusted to minimise the difference between the actual output of the network and the desired output [Haw99]. Although any differentiable function will do, typically, BP networks use the sigmoid function (logistic function) [RHW86] and is given by:

$$o = \frac{1}{1 + e^{-x}}. \tag{4.27}$$

The derivative of the sigmoid function is easily expressed in terms of its output, that is, $\frac{df(x)}{dy} = f(x) - (1 - f(x))$ and is used by the backpropagation learning rule to minimise the squared error between the network output values and the target values for these outputs. The total error, E, over all of the network output units is defined as:

$$E = \frac{1}{2}\sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2 \tag{4.28}$$

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 kth output unit for training example d respectively [Mit97]. To minimise E, the partial derivative of E with respect to each of the weights in the network must be computed ($\frac{\partial E}{\partial w}$). This is simply the sum of the partial derivatives of the error with respect to the weights, for each of the training examples and details of this derivation can be found in Rumelhart *et al* [RHW86]. Once this derivation is known the weights associated with each layer can be modified. This can be done after every input-output case and therefore no separate memory is required for the derivatives. An alternative approach, which is used in this thesis, 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 $\frac{\partial E}{\partial w}$:

$$\Delta w = -\eta \frac{\partial E}{\partial w}. \tag{4.29}$$

The size of the adjustment will depend on η, and on the contribution of the weight to the error of the function. That is, if the weight contributes a lot to the error, the adjustment will be greater than if it contributes in a smaller amount.

4.6.1 Evaluation of Backpropagation networks

Backpropagation networks are relatively simple to implement and prediction accuracy is often reported to be generally high. The network can handle categorical and continuous data types and can learn relationships directly from the data being modelled. They can model numerous different functions, including boolean functions and continuous functions. They are robust, working well even when the training data contains errors.

However, the backpropagation algorithm is only guaranteed to converge to some local minima and not necessarily to a global minima, although in many practical applications the local minima problem has not been shown to be hugely detrimental. Backpropagation can be susceptible to overfitting the training examples, however, several techniques exist to address this problem, including weight decay (decreasing each weight by some small factor during each iteration). BP networks can be slow, however, improved performance can be achieved by adding a momentum term when updating weights and was used in this thesis. The value of the learning rate is important. If it is too small, it can take a long time to converge. If it is too big, the algorithm may continually jump over the optimum weight values and fail to converge [Mit97].

4.7 Principal Component Analysis

Principal Component Analysis (PCA) also known as the Karhunen-Loéve transform, is a statistical technique used to lower the dimensionality of a dataset while retaining as much information as possible. It is widely used in data analysis and compression [DHS01], [TK99]. This method takes a set of data points and constructs a lower dimensional linear subspace that best describes the variation of these data points from their mean. PCA essentially performs an orthogonal transformation on the input data such that the variance of the input data is accurately captured using the resulting principal components. The first principal component is the combination of variables that explains the greatest amount of variation. Each subsequent principal component defines less of the variation than it's preceding components. The maximum number of components that can exist is the maximum number of variables [TF01].

To demonstrate the process of PCA assume that a dataset $\mathbf{X} = \{\mathbf{x^1}, \mathbf{x^2}, ..., \mathbf{x^N}\}$

exists, where x^i is composed of 2 dimensions, a_1^i, a_2^i, with mean values $\bar{a_1}, \bar{a_2}$. For illustration purposes this example only considers 2-dimensional data sets but a similar process exists for n-dimensional sets. First, the input vectors are normalised so that they have zero mean and unity variance. Then the covariance matrix, a measure of the strength of the linear relationships between the 2 dimensions, is determined using $cov(a_1, a_2) = E[(\mathbf{a_1} - \bar{\mathbf{a_1}})(\mathbf{a_2} - \bar{\mathbf{a_2}})]$. Next, the eigenvectors, $\mathbf{e_1}, ..., \mathbf{e_k}$, and eigenvalues, $\lambda_1, ..., \lambda_k$, for the covariance matrix are calculated. The eigenvector with the highest eigenvalue is the principal component of the data set. In general, once the eigenvectors are found from the covariance matrix, they are ordered in decreasing eigenvalue. The resulting eigenvectors and eigenvalues represent degrees of variability where the first eigenvalue is the most significant mode of variation. Finally, an $n \times k$ matrix $\mathbf{A}$ is constructed, where n is the original number of dimensions and k is the number of eigenvectors to be kept. The final dataset is calculated by multiplying the transpose of $\mathbf{A}$ by the normalised original data set $\mathbf{a} - \bar{\mathbf{a}}$. This gives the original data set in terms of the principal components [Smi02]. An example of PCA is illustrated in Figure 4.6.

Several algorithms exist for choosing the number of eigenvectors (principal components) to keep. Probably the best known is the Kaiser criterion, which specifies that all components with eigenvalues greater than 1.0 should be kept and this criterion is used in this thesis.

4.8 Summary

In this chapter, six machine learning algorithms that can be used to classify students as 'strong' or 'weak' programmers were described. The algorithms were k-nearest neighbour, C4.5, naïve Bayes, logistic regression, support vector machines and backpropagation. Each of the algorithms were presented and evaluated. The

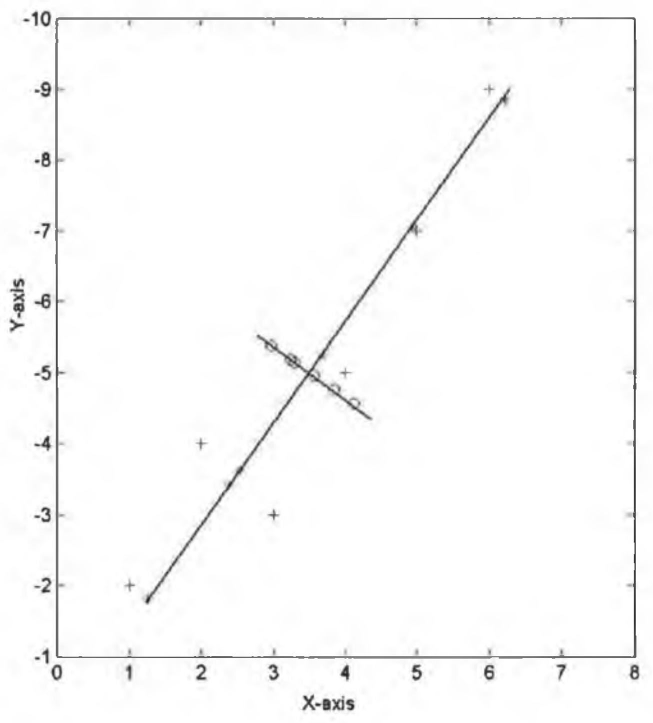


Figure 4.6: *PCA example: this figure shows six points where the coordinates have a high covariance. PCA calculates the principal axes using the input data. The first principal axis can be used to represent the data and can explain the majority of the variance found in the input data. In this example the first principal component describes 97.45% of the entire variance in the training set. By projecting the input data onto the first principal axis it is possible to represent the data in one-dimension. The +'s represent the input data, the *'s represent the input data projected onto the first principal component and the o's represent the input data projected onto the second principal component.*

chapter concluded with an overview of Principal Component Analysis, a technique to reduce the dimensionality of a dataset. In Chapter five, details on the use of PCA to reduce the number of predictors required are provided and the results of implementing each of the six machine learning algorithms to predict programming performance are described.

Chapter 5

Results: Effectiveness of the Predictors

This chapter details the results of implementing the six machine learning algorithms described in Chapter 4 using the data gathered from the empirical study as documented in Chapter 3. First, the procedure implemented to reduce the dimensionality of the data set is described. Then, the process for selecting the most important factors is outlined. Subsequently, the findings from implementing six machine learning algorithms using the most significant factors are presented. Finally, various approaches for optimising the results are described and implemented. The outcome of each optimisation is described.

5.1 Procedure

In this section, details of dimensionality reduction along with measurement techniques and the factor selection process are briefly described.

5.1.1 Dimensionality Reduction

Principal components analysis (PCA), as described in Section 4.7, is a dimension reduction technique that can be used to replace a large set of observed variables with a smaller set of new variables. PCA was implemented using two distinct approaches. Using the first approach every factor considered in the study was treated as input to PCA (irrespective of the underlying relationships between the factors). Components that satisfied the Kaiser criterion (all component with eigenvalues greater than 1.0) were retained for predicting programming performance. In the second approach PCA was applied to individual instruments where multiple items (questions) were used to measure the same underlying phenomena (where high correlations between the items are typical). PCA was applied to three separate instruments, specifically, the comfort-level scale (9 questions), the programming self-esteem scale (10 questions) and the self-efficacy scale (7 questions). Again, components that satisfied the Kaiser criterion were retained: one component for programming self-esteem, one component for self-efficacy and three components for the comfort-level questions.

The first approach was not found to be satisfactory for two reasons. Due to the effects of missing data, incorporating all factors resulted in a considerably reduced sample size. Maintaining a high sample size is very important in this study to facilitate good generalisation. Also, subsequent predictive modelling resulted in poorer performance than models built using PCA components derived from the second approach. Performing PCA on instruments with high multi-collinearity is more successful in this instance than on input that has a large amount of diversity.

As outlined in Section 3.2.2 data was collected in two study administrations. The first administration included all surveys, except the MSLQ, and was carried out early in the programming module (when the students had completed preliminary programming concepts) while the second administration was completed when they were on average one third of the way through the material. In order to determine the most important factors in predicting programming performance, two investigations were carried out. In the first investigation, all the factors from the first administration of the study were included, while in the second investigation all of the factors from both administrations were considered. Within each investigation, models were developed using both PCA approaches. Logistic regression was used to develop models to identify the most significant factors. Although any of the machine learning techniques could have been used, logistic regression was selected for three main reasons: (1) to maintain consistency with previous 'regression' based studies and with the pilot study, (2) it has been shown to perform well and is relatively straight-forward to implement, and (3) the models derived are easy to interpret and the beta-weights associated with each variable provide valuable insight for a model-builder on modifications that could improve performance.

In total over 40 models were developed with various degrees of freedom. All models were generated using 10-times 10-fold stratified cross validation. In this procedure, data is randomly split 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 nine 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 a further nine times and the results are averaged over all of the testing datasets.

5.1.2 Measurement Techniques

Three measurement techniques are employed: classification accuracy, sensitivity (true positive rate) and specificity (true negative rate).

The simplest form of evaluation is *classification accuracy*, that is, the proportion of instances correctly predicted. Using Table 5.1 for illustration, the compu-

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

Table 5.1: Sample confusion matrix

tation of this measure is given by Formula 5.1.

$$\frac{TP+TN}{TP+TN+FP+FN} \tag{5.1}$$

Sensitivity is a measure of the proportion of actual positive instances that are correctly classified, given by Formula 5.2. Specificity is the proportion of actual negative instances correctly classified, as illustrated by Formula 5.3.

$$\frac{TP}{TP+FN} \tag{5.2}$$

$$\frac{TN}{TN+FP} \tag{5.3}$$

5.1.3 Factor Selection

Over 40 logistic regression models, using combinations of the factors, were developed using the variables from the first administration of the study. A model using three predictor variables (Predictor Set 1), specifically, LC mathematics score (LCMATHEMATICS), number of hours playing computer games while taking the module (WHILEGAMES) and factor scores from PCA on the programming self-esteem scale (PROGSELFEST) were the most significant. A subset of

102 cases from a total of 123 were included in the model. The percentage of students accurately classified was significant at 76.5% and Table 5.2 provides further performance measures. Although a considerable number of other models were developed using other attributes from the first administration, no superior predictor set was found. LCMATHEMATICS and PROGSELFEST were found to have a positive relationship with performance, although WHILEGAMES was found to have a negative effect. A second model did emerge with a marginally higher prediction accuracy of 79%, but with a considerably reduced sample size ($n = 82$). The model included four predictor variables (Predictor Set 2) including the three predictor variables from the first model and the number of hours students were likely to spend studying module material per week (LIKELYHOURS) and measures of performance are given in Table 5.2. Given the considerable reduction in sample size this model is not considered further in this study.

The second investigation considered all of the variables in the study. A significant model emerged but care must be taken interpreting the results due to the reduced sample size and the differences noted earlier between the sample and population at Institute A and Institute C. The model included four predictor variables (Predictor Set 3), specifically, LCMATHEMATICS, WHILEGAMES, LIKELYHOURS and the Self-Efficacy for learning and performance scale from the MSLQ (MSLQSELFEFF). Ninety percent of students ($n = 58$) were classified correctly and the model had high sensitivity and specificity measures, as outlined in Table 5.2. Due to the considerably reduced sample size, coupled with the problems of sample representativeness it would be inappropriate to make recommendations based on these factors and thus this model is not considered in detail in this study. Subsequently, implementations of each of the machine learning algorithms outlined in Chapter 4 were developed using the factors in Predictor Set 1 and the results are presented in the following section.

Model ID	Acc %	Sen %	Spe %
Model 1 using Predictor Set 1	76.5	84	65
Model 2 using Predictor Set 2	79	82	75
Model 3 using Predictor Set 3	90	93	86
Acc = Accuracy, Sen = Sensitivity, Spe = Specificity			

Table 5.2: Most significant models found using logistic regression

5.2 Results

Each of the six machine learning algorithms were implemented using Predictor Set 1, that is, LCMATHEMATICS, WHILEGAMES and PROGSELFEST. Java implementations of these algorithms from the Waikato Environment for Knowledge Analysis, WEKA, as outlined in Witten and Frank (2005) [WE05], were used. The accuracy, sensitivity and specificity measures for each of the algorithms are presented in Table 5.3.

Based upon the accuracy measure, the most successful algorithms in descending order are naïve Bayes, support vector machine (SVM), logistic regression, back-propagation, C4.5 and 3-NN. Although overall accuracy is important in this study the sensitivity measure is deemed more important. 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 while 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, SVM, C4.5, logistic regression,

Algorithm	**Acc %**	**Sen %**	**Spe %**
Naïve Bayes	78.28	87.00	66.00
Logistic Regression	76.47	84.00	65.00
BP	75.46	84.00	63.00
SVM	77.49	87.00	63.00
C4.5	74.49	85.00	63.00
3-NN	71.64	77.00	59.00
Acc = Accuracy, Sen = Sensitivity, Spe = Specificity			

Table 5.3: Performance of each algorithm using Predictor Set 1

backpropagation and 3-NN. Finally in terms of the specificity measure, the best algorithms are, in order, naïve Bayes, logistic regression, SVM, backpropagation, 3-NN and C4.5. Using the sensitivity measure to choose an algorithm, the naïve Bayes and SVM models achieve the best results.

5.3 Improving the Results

Although the predictive accuracies found in this study are exceptionally high given the domain, further improvements were still desired. To this end, two further investigations were carried out in an attempt to increase the performance of the machine learning algorithms using Predictor Set 1. Each of these investigations is described in the following section.

5.3.1 Ensembles of Multiple Models

An ensemble of classifiers is a set of classifiers whose individual predictions are combined to classify new examples. Numerous studies have found that an ensemble of models yields higher generalisation performance than the best base model alone ([Die97]). In general, an ensemble consists of a set of base models, $m_1, ..., m_s$ where S is the size of the ensemble formed during the learning phase. Each base model in the ensemble is trained using training instances from the training set $T_1, ...T_n$. The predictions of the base models are combined during the application phase to produce the final prediction of the ensemble. Two ensembles were implemented, namely, Voting and StackingC. With Voting, the probability estimates of each of the classifiers in the learning phase is averaged in the application phase to derive a final prediction ([WE05]). Several different models using voting were implemented each with different combinations of the top five classifiers (excluding K-NN given the statistically significant differences found between it and the other classifiers). However, no improvements beyond using a single naïve Bayes classifier were found. In Stacking, a learning algorithm or level-1 learner is used to learn how to combine the predictions of the base-level classifiers also known as level-0 learners. A meta-level classifier is then used to obtain the final prediction from the predictions of the base-level classifiers ([Wol92]). StackingC implements a more efficient variant of the basic stacking algorithm using the classification probabilities of the level-0 learners ([See02]). An implementation of StackingC using linear regression as the meta-learner and naïve Bayes, SVM, logistic regression and backpropagation as the base learners resulted in a higher accuracy (82%) and sensitivity value (90%). This is an improvement of $\sim$ 4% on overall accuracy and $\sim$ 3% on sensitivity. Subsequent analysis of the errors generated by the base learners indicated that the errors made by each of the classifiers were highly correlated ($r > 0.90$) and as an ensemble can only be more accurate that its components classifiers if the individual classifiers disagree with one another, this explains why the improvements are not higher ([Die97]).

5.3.2 Model Improvement

Naïve Bayes and support vector machines produce the best predictions with regard to both accuracy and sensitivity. As such, investigations to optimise the results of both algorithms were carried out. The optimisations and results are discussed in this section.

Improving Naïve Bayes

Bagging and Boosting are two popular approaches for improving the performance of classifiers. With Bagging, a classifier, C_i, is built using each of M bootstrap training sets, $B_1, B_2, ..., B_M$, formed by uniformly sampling k instances with replacement from the original training set. A final classifier, C^*, is built from $C_1, C_2, ..., C_M$ and the estimated probability that any sample x has label $y = 1$ is the proportion of learned classifiers that output 1 for x. Boosting uses all instances at each repetition, but maintains a weight for each instance in the training set that reflects its importance; adjusting the weights causes the learner to focus on different instances and so leads to different classifiers. With both, the multiple classifiers are then combined by Voting to form a composite classifier. In Bagging, each component classifier has the same voting strength, while Boosting assigns different voting strengths to component classifiers on the basis of their accuracy ([WE05]). Implementations of the Bagging algorithm, using a resample size of $k = 50$, 66 and 100 respectively did not improve the accuracy of the naïve Bayes algorithm. AdaBoost.M1 was used to implement boosting. This approach works by generating the classifiers sequentially and changing the weights of the input training instances based on classifiers that were previously built. However, like the Bagging technique, no improvements were found in performance.

Improving Support Vector Machines (SVM) Performance

As with naïve Bayes further improvements to the SVM results were sought. The implementation previously discussed was linear, however, as outlined in Section 4.5, support vector machines can be implemented using a variety of kernels including Polynomial and Radial Basis Function (RBF) kernels. Implementations of each of these kernels were employed using LibSVM [CL01]. Polynomial SVMs with exponent values of 2, 3 and 4 resulted in poorer performance than linear SVMs. A grid search using 10 times 10 fold cross validation was performed to determine the best parameters for C (soft margin parameter) and γ (kernel parameter) for an RBF SVM. Using the best parameters ($C = 512$, $\gamma = 0.0078$) resulted in an overall accuracy of 78.95%, sensitivity of 88% and specificity of 66%. Given the only marginal improvement in results produced by this technique compared to the increased computational requirements, the selection of naïve Bayes to predict future student performance is currently recommended.

5.4 Summary

Principal Component Analysis (PCA) was implemented to reduce the dimensionality of the dataset. Two approaches were taken. Using the first approach PCA was performed on the complete dataset while the second approach applied PCA to reduce the dimensionality of individual instruments used in the study. Data in this study was collected in two separate administrations and thus two distinct investigations were carried out based on the amount of data available after each administration. Logistic regression was used to determine the factors that could account for the most amount of variance in student performance. Three factors were identified in the first study: LC mathematics result, Programming self-esteem and the number of hours spent playing computer games at the start of the module. Four factors emerged as the most important in the second investigation: LC mathematics result, the number of hours students were likely to spend studying module material, self-efficacy for learning and performance and the number of hours spent playing computer games at the start of the module. However, due to the lack of representativeness of the sample in the second investigation, subsequent analysis focused only on the factors identified in the first investigation.

Six machine learning algorithms were implemented to classify students as 'weak' or 'strong'. The most successful algorithms in descending order of prediction accuracy were naïve Bayes, SVM, logistic regression, backpropagation, C4.5 and 3-NN. Although overall accuracy is important in this study the sensitivity measure (students correctly predicted as 'weak') is more important. In order of importance based on the sensitivity measure, the algorithms of choice are naïve Bayes, SVM, C4.5, logistic regression, backpropagation and 3-NN. Using the sensitivity measure to choose an algorithm, it would appear that any algorithm except for 3-NN will result in a similar sensitivity measure with the naïve Bayes and SVM models achieving the best results. However when the overall accuracy is also considered naïve Bayes achieves the highest results.

Although the predictive accuracies found in this study are exceptionally high given the domain, further improvements in the results were sought. Two techniques, Voting and StackingC, for developing ensembles of multiple models were implemented. Several different models using voting were implemented each with different combinations of the top five classifiers but no improvements above using a single naïve Bayes classifier were found. An implementation of StackingC resulted in higher accuracy (82%) and sensitivity value (90%). This is an improvement of $\sim$ 4% on overall accuracy and $\sim$ 3% on sensitivity. Subsequent analysis of the errors generated by the base learners indicated that the errors made by each of the classifiers were highly correlated ($r > .9$) and this helps to explain why the improvements are not greater.

Attempts were also made to improve the results of the two top performing algorithms: naïve Bayes and SVM. Neither Bagging nor Boosting improved the previous naïve Bayes results. Polynomial and Radial Basis Function SVMs were implemented using LibSVM. Polynomial SVMs with exponent values of 2, 3 and 4 resulted in inferior performance than linear SVMs. An implementation of an RBF SVM resulted in an overall accuracy of 78.95%, sensitivity of 88% and specificity of 66%. Given the only marginal improvement in results produced by this technique compared to the increased computational requirements of this procedure, the selection of naïve Bayes to predict performance is currently recommended and is discussed further in Chapter 6.

Chapter 6

Discussion

This chapter provides a discussion on the programming predictors investigated in this study and the subsequent machine learning models developed. First, the significant predictors used to develop the machine learning models are presented. Next, an evaluation of the instruments and factors that were not used in the final models is described. Then, a discussion on the results achieved by the machine learning algorithms is provided and a comparison of the results found in this study with the most closely related study is presented. A technique that can be used to interpret misclassified cases is described and the chapter concludes with an epilogue study that further validates the effectiveness of the naïve Bayes algorithm for predicting programming performance.

6.1 Predictor Sets

In this section a review of the three significant predictor sets found in this thesis is presented. Predictor Set 1 included three factors: LC mathematics score, the number of hours students spent playing computer games and a student's perception of their programming self-esteem. The combination of these three factors led to a highly significant model and to the largest sample size ($n = 102$). The second set of predictors, (Predictor Set 2) incorporated the same three factors as Predictor Set 1 but also included the number of hours a student felt they were likely to spend studying for the module. The inclusion of this factor led to a marginally more accurate prediction model than Predictor Set 1 but with a reduced (possibly non-representative) sample size of $n = 82$. Discussion of this model in this thesis is merely to encourage future researchers to explore the model further but no conclusions can be drawn at this point. The final set (Predictor Set 3) was composed of four predictors: LC mathematics, the number of hours spent playing computer games, the likely number of hours spent studying for the module and the score on the MSLQ self-efficacy scale. The combination of predictors in Predictor Set 3 were found to account for the highest prediction accuracy but extreme caution must be taken due to the considerably reduced sample size ($n = 58$) and the differences noted earlier between the sample and population at Institute A and Institute C. Again, no definitive conclusions can be made using this predictor set however, future work would be justified to investigate the predictor set further.

6.1.1 Predictor Set 1

Predictor Set 1 included three factors: LCMATHEMATICS, WHILEGAMES and PROGSELFEST. The fact that LC mathematics is a useful predictor is of no great surprise given our literature review in Chapter 2. The Programming Self-Esteem Instrument was specifically designed for this study. Its success as a predictor of programming performance can be attributed to the fact that it is based on a well-established, well-respected measure of self-esteem [Ros65] and also because it can be thought of as another measure of comfort-level, which has previously been found to be a predictor of programming performance. The importance of computer game-playing as a negative predictor of performance is in line with the findings of

Cantwell-Wilson and Shrock [CWS01] and Evans and Simkin [ES89]. This study did not attempt to uncover why game-playing negatively relates to programming performance but a study to do this would be useful. It would also be useful to determine if game-playing negatively influences general academic performance, performance on computer science courses or just programming performance. If game-playing does not effect performance on non-computer science courses, it may be that non-game-playing-students view time in front of a computer as predominantly work (study) time while for game-playing students it is also seen as hobby time which may interrupt study-time. Additional studies in the area should be carried out to explore this phenomena further.

6.1.2 Predictor Set 2

Predictor Set 2 was composed of LCMATHEMATICS, WHILEGAMES, PROGSELFEST and LIKELYHOURS. The inclusion of the likely number of hours a student spends studying for the module results in an improvement in the number of students classified as strong (from 65% in Predictor Set 1 to 75% in Predictor Set 2), but with a slight decrease in the prediction performance of weak students (from 84% in Predictor Set 1 to 82% Predictor Set 2). An ANOVA test failed to reveal any significant differences between the mean likely hours that strong and weak students would study for. Thus knowing the likely number of hours a student will study improves the classification of stronger students but not weaker ones. Further investigation is needed to explore these findings.

6.1.3 Predictor Set 3

The third predictor set included LCMATHEMATICS, WHILEGAMES, LIKELYHOURS and MSLQSELFEFF. While the measures of self-regulated learning (SRL) were generally disappointing, the self-efficacy scale on the Motivated Strategies for

Learning Questionnaire (MSLQ) was found to have predictive value. This is not surprising given the findings on the importance of self-efficacy in other studies. Again, this can be thought of as a comfort-level type measure and when it is included in the model, the programming self-esteem measure is no longer needed. In fact, running the model, with the same 58 students as in Predictor Set 3 but using the programming self-esteem measure instead of the MSLQ self-efficacy measure results in poorer prediction accuracy, with 88% of the students classified correctly. This may indicate that Predictor Set 1 could be further improved if this measure was available for all students, assuming that the findings are not skewed by the sample representativeness.

While Predictor Set 2 and Predictor Set 3 result in higher classification accuracies there are considerable problems with missing data, sample size and sample representativeness. As such, the only predictor set that is recommended currently for use is Predictor Set 1. However, this thesis provides sufficient evidence to justify further research on Predictor Set 2 and Predictor Set 3.

6.2 Factors Not Used In Predictor Sets

In this section a discussion of the instruments and factors that were not incorporated into the final models is presented and suggestions on why the factors were not found to be significantly predictive are provided.

6.2.1 Background Questionnaire

Gender as a factor

Recently there has been concern about the lack of women studying computer science [CW00]. Typically, enrolment of female students is much lower than male enrollment and this study reflects this trend with 22.5% female participants and

77.5% male participants. This study found that inclusion of gender as a factor in the prediction models does not lead to higher classification accuracy. However, the models more accurately classify male students than female students, with 84% of male students and 70% of female students correctly classified. Subsequent t-tests of independent samples revealed that statistically significant differences existed between the LC mathematics score ($t(103) = 3.765$, $p = 0.001$), number of hours playing games ($t(115) = 3.634$, $p = 0.001$) and overall module performance ($t(118) = 2.210$, $p = 0.029$) of male and female students. The removal of the number of hours students spend playing computer games improves the classification of female students to 79% indicating that its inclusion as a factor is not constructive in predicting female student performance. Furthermore, a prediction model that considers LC mathematics, programming self-esteem and meta-cognitive strategy use accurately classifies 92% of female students, however the sample size is very low ($n = 14$). Thus building separate models for male and female students could be a useful future direction.

Place of study (institution) as a factor

Developing a separate prediction model using Predictor Set 1 for each institution results in a prediction accuracy of 85% at Institute A, 96% at Institute B, 92% at Institute C and 71% at Institute D. The lower result at Institute D is not surprising as only 7 students were included in the classification due to a large number of missing data. In most cases, LC mathematics score was missing. A large proportion of the students at this college were educated outside the state and did not sit the LC mathematics examination. This is a problem for the current model. Although several substitution schemes were examined, none were found suitable. Future research on other substitution schemes to alleviate this problem would be useful. An alternative solution would be to devise a mathematics examination, somewhat similar to the LC mathematics examination, that students could take at the start of the course. Aside from this issue the model does appear to fit each institution well.

The order of importance of each of the three factors is different among the institutions. At institutes B, C and D the most important predictor is playing computer games, followed by LC mathematics at Institute B and Institute C and programming self-esteem at Institute D. At Institute A the most important factor is programming self-esteem, followed closely by LC mathematics score. It is interesting that the exact same ordering of predictors is found at Institute B and Institute C. Both of these institutes have similar admission requirements and are the same type of Institute (institutes of technology). As such, students with similar academic backgrounds would attend each of these college and the model appears to adjust accordingly for this. Institute A is a university and in general would have a higher admissions requirement. Computer game-playing at this institution is the least important predictor as opposed to the most important at the others. It could be interpreted that students at Institute A who in general would have higher entry results, are more academically oriented and are less likely to play games. However, a one-way ANOVA test failed to reveal any significant differences between computer game-playing at each of the institutions. An ANOVA test revealed a statistically significant difference between the mean LC mathematics score at Institute A and each of the other institutes ($F(3, 98) = 24.985, p < 0.001$). There was no difference in the LC mathematics mean score at Institute B, Institute C and Institute D. This could partially explain why LC mathematics is a more important factor at Institute A. No statistical differences were found on programming self-esteem score at the different institutions.

Science as a factor

Results from the pilot study suggested that performance in LC science subjects was related to programming performance. However, not all students in Ireland study a science subject for LC examination as is the case in this study with only 67% of students taking a science subject for this examination. However, inclusion of this factor results in a significantly reduced (possibly unrepresentative) sample size or requires investigation of an appropriate substitution scheme to determine a suitable score (value) for students who did not study a science subject. To examine the effect of science in this study, the inclusion of science as a factor (with a reduced sample size) was first investigated. If this led to a superior predictive model it was the authors intention to investigate suitable substitution schemes. Various models including science as a factor were developed, however none were more highly predictive and thus further investigation was not warranted.

Prior Programming and Computer Experience

Prior programming experience was not found to be a predictor of performance on the module. Only 37% of students indicated that they had some form of previous programming experience. This experience was either from taking a programming course (object-oriented or procedural), a web-design course or self-taught. A t-test of independent samples revealed that there was no statistically significant difference between the mean performance of students with and without previous experience. Although previous studies have identified a relationship between prior programming experience and performance on an introductory module, as previously mentioned, these studies have typically taken place in countries where students can study programming at national examination level. This is not the case in Ireland and could explain our findings. A more detailed study of the type of prior experience students have commencing an introductory programming module, with a considerably larger sample, could help to explain this finding further. Only 12.5% of students had no previous experience of internet-surfing and emailing, and 30% of students had no previous experience using application software. No statistically significant differences were found between the performance of students with and without experience in these two areas. This result was anticipated and confirms earlier research that previous computer experience (non-programming computer experience) is not a useful predictor of programming performance. Programming is a skill and knowing how to email, surf the web or use an application such as MS Word are not advantageous in learning how to program.

Miscellaneous factors

In a previous study by Cantwell-Wilson and Shrock [CWS01] the encouragement students receive from others (parents, teachers, family members etc.) to study CS\IT and workstyle preference (preference to work alone or in a group when solving programming problems) was examined. Neither factor was found to be predictive of programming performance. To validate these findings both factors were re-examined here. In this study, there was an even distribution of students who received and who did not receive encouragement to study CS\IT. No statistically significant differences were found between students with or without encouragement. Forty-one percent of students indicated that they preferred to work alone when solving programming problems with the remaining 59% indicating a preference to work in groups. Again, no statistically significant differences could be found between the mean performance of students who preferred to work alone and those who preferred to work in groups. However, it is interesting to note that a t-test did tend towards significance ($t(117) = 1.995$, $p = 0.053$) with students who preferred to work alone having a higher performance. A possible explanation for this marginal effect could be that weaker students prefer the support of working with other (possibly stronger) students than working alone. Further research on the types of students weaker students prefer to work with (for example students perceived to be worse than, same as or better than themselves) could be useful in assigning students to group activities such as peer-programming and PBL. Finally, as outlined in Chapter 2 some studies have found the number of hours a student spends working at a part-time job relates to programming performance. No such relationship was identified in this study. In fact, students who indicated that they worked between 6 to 10 hours per week had the highest average programming performance. Students who did not work or worked over 16 hours per week had similar mean performance. A possible explanation for this could be that some underlying variable, perhaps motivation, is more important than time spent working at a part-time job.

6.2.2 Cantwell-Wilson and Shrock Comfort-Level measure

While our programming self-esteem measure proved useful in our classification model, the Cantwell-Wilson and Shrock [CWS01] comfort-level measure did not add any further value to the models. The instrument resulted in only a slightly poorer classification model when used with LC mathematics and game playing, (omitting PROGSELFEST). This suggests that it is measuring the same phenomena as programming-self-esteem, however, programming self-esteem is a superior measure.

6.2.3 Computer Programming Self-Efficacy Scale

The 'Computer Programming Self-Efficacy' scale did not add any further value to the models. Given that we are trying to capture attributes at a very early stage in the programming course, only seven questions asking students to judge their ability at specific programming tasks, from the scale could be administered. Clearly, this shortened version is not sufficient to capture programming self-efficacy.

6.2.4 Motivated Strategies for Learning Questionnaire

Analysis of the SRL measures using independent t-tests revealed that weaker students had lower intrinsic motivation than stronger students ($t(77) = -3.298$, $p = 0.001$). In addition, weaker students used less meta-cognitive strategies (specifically, planning, monitoring and regulating) than stronger students ($t(79) = -4.566, p = 0.001$). While, use of meta-cognitive strategies and intrinsic motivation level do not increase the accuracy of the models (perhaps because the information they provide is already captured in the model), this information is useful to educators who are seeking to help students learn programming. Given the evidence provided in this thesis on the role of SRL in learning to program future studies would be well justified in considering this area further.

6.2.5 Summary of Factors Not In Predictor Sets

In summary, Predictor Set 1 was found to predict better the performance of male students than female students. Removing the number of hours students spend playing computer games results in a higher percentage of female students being correctly classified. The inclusion of the meta-cognitive strategy scale from the MSLQ appears to be an important predictor of female student performance. Building separate models for male and female students could be a useful future direction. Developing a separate model using Predictor Set 1 for each institution, results in a high classification accuracy at each institution. Missing LC mathematics scores at Institute D is problematic because a large proportion of students were educated outside the state and consequently did not sit the LC mathematics examination. Future research on devising a mathematics examination that students could take at the start of the course would be useful.

Performance in LC science was not found to be a significant predictor of programming performance in this study. Although previous studies have identified a relationship between prior programming experience and performance on an introductory module, these studies have typically taken place in countries where students can study programming at final second-level examination level. As this is not the case in Ireland this could explain why prior programming experience was not found to be a predictor of programming performance in this study. The Cantwell-Wilson and Shrock comfort-level measure [CWS01], resulted in a slightly poorer classification model when used with LC mathematics and game playing, (omitting PROGSELFEST) indicating that it is measuring the same phenomena as programming-self-esteem, however, programming self-esteem is a superior measure. The shortened version of the computer programming self-efficacy scale was not sufficient to capture self-efficacy. Analysis of the SRL measures revealed that weaker students had lower intrinsic motivation than stronger students and that weaker students used less meta-cognitive strategies than stronger students. This is in line with previous findings on SRL and programming performance.

6.3 Performance of the Machine Learning Algorithms

A review of the accuracy, sensitivity and specificity measures of Predictor Set 1 in Chapter 5 indicated that many of the algorithms had highly comparable results. Given such similar results selection of the most suitable 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 significant 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 [TF01].

Based upon the accuracy measure, the most successful algorithms in descending order are naïve Bayes, SVM, logistic regression, backpropagation, C4.5 and 3-NN. An ANOVA test revealed that there were statistically significant differences on the overall accuracy of 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, SVM, backpropagation and C4.5. However, 3-NN was found to have statistically significant lower accuracy than naïve Bayes, logistic regression and SVM but no statistically significant differences were found between it and C4.5 or backpropagation.

In order of importance based on the sensitivity measure, the algorithms of choice are naïve Bayes, SVM, C4.5, logistic regression, backpropagation and 3-NN. With regard to the sensitivity measure, an ANOVA test revealed that there were significant statistical differences between the algorithms, $F(5, 594) = 6.496$, $p < 0.001$. 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.

Specificity, although important is not as critical a measure in this study. The best algorithms, in order, on this measure were naïve Bayes, logistic regression, SVM, backpropagation, 3-NN and C4.5. However, no statistically significant differences were found between the algorithms based on specificity.

Using the sensitivity measure to choose an algorithm, it would appear that any algorithm except for 3-NN will result in a sensitivity measure that does not have a statistically significant difference with any of the other algorithms. This is important as it means that if an interested party has a preference for or expertise with a particular algorithm they can be confident that it achieves statistically comparable results to all of the other algorithms (except 3-NN). However, naïve Bayes achieves the highest 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, SVM and backpropagation have statistically significant higher training times than naïve Bayes and C4.5. This provides further evidence to support the use of naïve Bayes to solve this classification problem.

Although this is the first study to use a variety of machine learning algorithms to predict programming performance, a study by Kotsiantis *et al.* investigated the effectiveness of the same machine learning algorithms to predict performance on a distance learning course [KPP04]. In this study, naïve Bayes, SVMs, backpropagation, k-nearest neighbour, logistic regression and a decision tree (C4.5) were implemented and therefore it is useful to compare our findings to theirs to determine (1) if similar algorithms are useful at predicting performance in other academic domains, and (2) to check if any considerable differences exist between the findings of the two studies that could suggest implementation problems. The results of the study are illustrated in Table 6.1.

Kotsiantis *et al* argue for the use of naïve Bayes as the best overall algorithm. In their study naïve Bayes had a higher statistically significant sensitivity measure than all of the other algorithms. Backpropagation, logistic regression and SVM had the next highest sensitivity measure, with no statistical differences between them. With regard to overall accuracy, naïve Bayes, logistic regression, backpropagation and SVM were the top performers with no statistically significant differences between them. Finally, although less important, no significant statistical differences were found between the SVM, logistic regression and backpropagation algorithms on the specificity measure, while naïve Bayes was found to have statistically sig-

Table 6.1: Performance of each algorithm in the study by Kotsiantis *et al.* [KPP04]

Algorithm	**Acc(%)**	**Sen (%)**	**Spe(%)**
Naïve Bayes	72.48%	78.00%	67.37%
Logistic regression	72.32%	76.06%	68.52%
Backpropagation	72.26%	76.32%	68.31%
SVM	72.17%	76.05%	69.06%
C4.5	69.99%	73.89%	66.44%
3NN	66.93%	71.49%	62.00%
Acc = Accuracy, Sen = Sensitivity, Spe = Specificity			

nificant lower specificity than the SVM and logistic regression algorithms.

Table 6.2 provides a comparison of the results between the [KPP04] study and the current one. As can be seen, in both studies naïve Bayes is the top performer with SVM, backpropagation and logistic regression following closely behind on all measures. As naïve Bayes is relatively straight-forward to implement and understand and achieves the highest overall performance, the author recommends its use for predicting incoming student performance. However, this thesis has also shown that each of the other algorithms have statistically comparable performance (except for 3-NN) and thus can be used to predict programming performance either.

6.4 Using Classification Probabilities

Each of the algorithms in this study can estimate the probability of belonging to a particular class in addition to predicting a class. However, previous empirical

Table 6.2: Comparison of results

Kotsiantis *et al.*	**Bergin**
Ordered by Accuracy	
Naïve Bayes	Naïve Bayes
Backpropagation	SVM
Logistic regression	Logistic regression
SVM	Backpropagation
C4.5	C4.5
3-NN	3-NN
Ordered by Sensitivity	
Naïve Bayes	Naïve Bayes
Backpropagation	SVM
Logistic regression	C4.5
SVM	Logistic regression
C4.5	Backpropagation
3-NN	3-NN
Ordered by Specificity	
SVM	Naïve Bayes
Logistic regression	Logistic regression
Backpropagation	SVM
NB	Backpropagation
C4.5	3-NN
3-NN	C4.5

studies indicate that while naïve Bayes is a very accurate classifier care must be taken when interpreting the class probabilities due to the assumption of conditional independence. The technique works well at predicting the correct class but the strengths of the predictions tend to be inaccurate [Mit97]. For illustration purposes logistic regression is used here as the class probabilities are very easily computed. In logistic regression classification probabilities are used to determine which class a student belongs. In general, the cut-off value is 0.5. Thus, we are not restricted to treating our outcome as dichotomous (weak or strong) but can further classify performance using the classification probabilities. This is an important benefit as it allows borderline students to be identified, that is students who are clearly not very strong or very weak, for example, students with a classification probability between 0.35 and say 0.65. For example, using the attributes described in Predictor Set 1 a classification model can be derived using all 102 students as training data (for illustration purposes it is simpler to consider a single training set than 10-fold cross validation). Twenty students are misclassified. However, analysis of the classification probabilities indicates that 10 of the misclassified students have a classification probability between 0.35 and 0.65 and thus form a borderline group of students. Assuming the objective is to assist weaker students, students in this borderline group should also be monitored. Of the remaining 10 students, 3 are classified as strong but are actually weak and 7 are classified as weak who are actually strong. Given the above objective it could be argued that the only significant error is the 3 students classified as strong who are weak. Furthermore, the classification probabilities can be used as a confidence measure of how well a student belongs in a particular group. For example, one would be much more confident that a student with a classification probability of 0.9 belongs in the stronger class than a student with a classification probability of 0.6.

6.5 Epilogue

In the academic year 2005-2006 students enrolled on CS101, an introductory programming module, at the Department of Computer Science were asked to participate in an additional study to verify the effectiveness of naïve Bayes at predicting programming performance. Students were asked to answer questions based on the three factors identified in Predictor Set 1, that is their LC mathematics result, the number of hours spent playing computer games and questions on the programming self-esteem scale. Twenty-one of the 22 students (95%) who completed the module participated in the study. The study was carried out when the students had completed three weeks of Java programming (variable types, selection statements and iteration).

The full set of students who participated in the main study and had no missing data, (n = 102), were used as training instances to develop a final naïve Bayes model. The model achieved an overall prediction accuracy of 81% (4 students were misclassified). The sensitivity measure was 80% (2 students misclassified) and the specificity measure achieved was 82% (2 students misclassified). With regards to the two students who were predicted to be 'strong' programmers but were actually 'weak', the first student achieved an overall result of 54.97% and the cut-off value for weak was 55.5%. That is, had the student achieved 0.6% more they would have been correctly classified, increasing the overall accuracy measure to 86% and the sensitivity measure to 90%. The second student who was misclassified as 'strong' did not attend any lab or PBL workshop sessions and attended less than 5% of the lectures in the second semester. Prior to their non-attendance the student had performed well in their class and lab exams.

This study further confirms the effectiveness of the naïve Bayes model at predicting programming performance.

6.6 Summary

In this chapter the significant predictor sets (Predictor Set 1, Predictor Set 2, Predictor Set 3) identified in this thesis were discussed. With regard to Predictor Set 1, the findings on the importance of mathematics and computer game-playing is in line with other studies as outlined in the literature review in Chapter 2. Although the Programming Self-Esteem instrument was developed for this study it can be thought of as another measure of comfort-level, which has previously been found to be a predictor of programming performance. Analysis of how each of these factors influence weak and strong students, implies that strong programming students, on average, tend to achieve a D grade (pass) in higher-level LC mathematics or an A grade in ordinary-level LC mathematics. They tend to have high levels of programming self-esteem and on average spend three hours playing computer games per week. Weak programming students achieve on average a B grade in ordinary-level LC mathematics, have lower levels of programming self-esteem and spend, on average five hours playing computer games.

Predictor Set 2, included the likely number of hours a student would spend studying for the module and resulted in a slightly more predictive model but with a considerably reduced sample size. Further research is required to validate this model. With regard to Predictor Set 3, the inclusion of the MSLQ self-efficacy scale in place of the programming self-esteem scale resulted in a considerable improvement in classification accuracy (90%). Again, this can be thought of as a comfort-level type measure and suggests that Predictor Set 1 could be further improved if this measure was available for all students, assuming that the findings are not affected by lack of sample representativeness. Based on the problems of missing data and sample representativeness associated with Predictor Set 2 and Predictor Set 3, this study recommends the use of Predictor Set 1 for classifying student performance. However, sufficient evidence has been provided to justify further research studies on the predictor sets.

Subsequent analysis based on gender indicated that the factors in Predictor Set 1 are better at predicting the performance of male than female students. Removing the number of hours students spend playing computer games improves the classification of female students suggesting that its inclusion is not constructive in predicting female student performance while the use of the meta-cognitive strategy scale from the MSLQ appears to be an important predictor of female student performance. Building separate models for male and female students could be a useful future direction.

Developing a separate prediction model for each institution, results in a high classification accuracy for each. The poorest prediction at Institute D (71%) is not surprising as only seven students were included in the classification due to a large number of missing data. These missing data were predominantly LC mathematics score as a large proportion of students at this college were educated outside the state and did not sit the LC mathematics examination. This is a problem for the current model. Future research on devising a mathematics examination, somewhat similar to the LC mathematics examination, that students could take at the start of the course would be useful.

This chapter also provided a discussion on the factors that were not incorporated into the final models. Of note, science was not found to be a significant predictor of programming performance and inclusion of this factor leads to problems of missing data substitution. Although previous studies have identified a relationship between prior programming experience and performance on an introductory module, these studies have typically taken place in countries where students can study programming at final second-level examination level. This is not the case in Ireland and could explain why prior programming experience was not found to be a predictor of programming performance in this study. The Cantwell-Wilson and

Shrock comfort-level measure [CWS01], resulted in slightly poorer classification when used with LC mathematics and game playing, (omitting PROGSELFEST) indicating that it is measuring the same phenomena as programming self-esteem, however, programming self-esteem is a superior measure. The shortened version of the computer programming self-efficacy scale was not sufficient to capture self-efficacy. Analysis on the SRL measures revealed that weaker students had lower intrinsic motivation than stronger students and that weaker students used less meta-cognitive strategies than stronger students. This is in line with previous findings on SRL and programming performance.

With regard to the choice of machine learning algorithm based on the sensitivity measure, it would appear that any algorithm except for 3-NN will result in a sensitivity measure that does not have a statistically significant difference with any of the other algorithms. However, naïve Bayes achieves the best overall results. Further, analysis based on training times further confirms the choice of naïve Bayes to classify student performance. Although this is the first study to use machine learning algorithms to predict programming performance, a similar study by Kotsiantis *et al.* to predict performance on a distance learning course [KPP04]. A comparison of the results found in each study indicated a similar ranking of classifiers. In particular, the Kotsiantis *et al.* study also found naïve Bayes to be the top performer with SVM, backpropagation and logistic regression following closely behind on all measures.

The chapter concluded with the recommendation that classification probabilities could be used to determine the students who were neither clearly weak or clearly strong. As such it was recommended that any student with a probability distribution between 0.35 and 0.65 should be treated as a borderline student and handled accordingly. An illustration was provided using logistic regression.

Chapter 7

Conclusion and Future Directions

This thesis detailed longitudinal research on factors that influence programming success and described a number of techniques for predicting programming performance. In addition, recommendations on the most suitable techniques for predicting future performance were provided. This concluding chapter summarises the contributions made and provides suggestions for possible future directions for this work.

7.1 Contributions

The work outlined in this thesis makes three fundamental contributions to the field. This section summarises and comments on each of these contributions

7.1.1 Longitudinal Research on Programming Predictors

This thesis documents a three year multi-institutional, multivariate study to determine the factors that influence programming success. The research involved four institutions, investigating 25 factors. The vast majority of previous studies on this research problem are not replicated, have small sample sizes, and involve a single class. Given this, one is reluctant to fully trust the generality of the findings. The institutes that participated in this research were diverse in that one was a university, two were institutes of technology and one was a college of further education and as such, they attract students with varying previous academic experience and achievement. In addition, the specifics of the programming modules (including language taught) was not the same nor were the modules taught by the same teacher. However, the prediction models developed achieve high prediction accuracy at each of the institutions. This increases confidence in the likelihood that the models will generalise well elsewhere.

The studies documented in this thesis provided evidence on the importance of performance in LC mathematics, perceived level of programming self-esteem and the effects of playing computer games on programming success. These three factors were found to significantly predict introductory performance. The fact that the factors were found to be so predictive and were measured at the start of the year when students had only minimal experience of programming concepts is especially important. Early interventions can now be developed and tailored to assist struggling students. It is important to note that the factors are also predictive of strong students and thus additional resources or alternative streams could be provided to further develop and foster their skills from a very early stage.

The study also examined numerous other factors and found that they failed to contribute further to the prediction model, for example, prior programming experience, number of hours a student spends working at a part time job, encouragement from others to study programming, preference to work alone or in a group when solving problems and number of hours using application software, emailing or surfing the web before and during the early stages of the course. It is disappointing that these factors were not found to be useful. However, given that this research was carried out over three years and involved multiple institutions, it may be more beneficial to direct future research on identifying new factors as it seems unlikely that these factors will prove predictive in other studies.

7.1.2 Instrument Development

During the course of this research a number of questionnaire items and instruments were developed. Of particular importance was the development of a new instrument, the programming self-esteem scale, based on Rosenberg's self-esteem measure [Ros65]. The scale proved to be a very important comfort-level type measure and was found to out-perform the Cantwell-Wilson and Shrock comfort-level measure and the shortened computer-programming self-efficacy scale in predicting programming performance.

7.1.3 Initial Investigation of the role of Self-Regulated Learning

The work carried out on self-regulated learning in this thesis was the first detailed investigation on using SRL to predict programming performance. While the problems of missing data and sample representativeness meant that the findings on SRL had to be interpreted with caution the results did suggest that aspects of SRL, in particular the self-efficacy scale on the MSLQ, were useful in predicting programming performance. Furthermore, the findings that weaker students are less intrinsically motivated than stronger students and use fewer meta-cognitive strategies justifies further investigation in this area.

7.1.4 Model Development

Typically, the models built to predict programming performance are statistical, with multiple linear regression the most common technique used. The work outlined in this thesis is a first attempt to utilise a variety of machine learning (ML) algorithms to predict introductory programming performance. While similar techniques have been investigated in other academic domains, a detailed review of the literature suggests that no comprehensive study on the effectiveness of a variety of ML techniques to predict performance has been carried out within this domain. This work is important as it bridges the gap between predicting programming performance and the application of machine learning techniques, and provides the foundations for future work on the use of artificial intelligence techniques for analysing this problem. It also encourages computer science educators to consider a broader suite of techniques.

The models developed as part of this thesis are the most effective models ever developed to predict programming performance. Similar results have never been achieved before. Further optimisations were also carried out to investigate if the results could be further improved. Several techniques were implemented with varying performance. StackingC, an ensemble algorithm, was shown to result in a higher prediction accuracy than that achieved by a single classifier. This finding not only gives interested parties an additional means of improving prediction accuracy but also justifies and warrants further research and development on optimisation techniques and, in particular, ensemble methods.

7.1.5 Recommended Algorithm

Although visual inspection indicated that naïve Bayes was the most effective algorithm for predicting programming performance, detailed statistical analysis was carried out to determine if there were any statistically significant differences between the prediction accuracy of each of the algorithms. This was important as interested parties may have a preference for the choice of algorithm they would like to implement and as such need to know if the use of a particular algorithm(s) would result in a statistically significant lower prediction performance. The analysis determined that all of the algorithms, except 3-NN, had statistically comparable prediction performance and thus could be used to predict incoming student performance. However when consideration is also given to training times along with ease of implementation and interpretation, naïve Bayes is recommended as the best choice for predicting future student performance.

7.2 Future Directions

There are numerous possible directions for future work in this area. In this section we provide suggestions for future work within the area.

7.2.1 Further Research on Programming Predictors

It is important that work continues on determining the attributes to predict programming performance. Approximately 20% of students in our studies are still misclassified and sizeable improvements to this figure may not take place until further significant factors are identified.

As such, future work should seek to validate the effectiveness of SRL in predicting programming performance. The MSLQ should be re-administered to determine if it is a useful indicator when administered in the very early stages of the module. If it is, then incorporating the measure with the number of hours a student is likely to study on the module, LC mathematics score and the number of hours playing games could result in an improved model. In addition, future studies should further examine SRL with a view to developing interventions to assist students. In this study weaker students were found to have lower intrinsic motivation levels and to use less meta-cognitive strategies than stronger students in this study. If future studies can determine that these factors have a causal relationship on programming performance, that is, that level of intrinsic motivation and meta-cognitive strategy use effect (cause) programming performance, then interventions that improve intrinsic motivation levels and meta-cognitive strategy use could improve programming performance.

7.2.2 Assessing Mathematical Ability

With regards to LC mathematics score, studies need to be carried out in other countries to see if performance on alternative mathematics tests can be used in the model instead. Moreover, work to develop a test that could be given to introductory programming students would remove the necessity of relying on results on a test that some students may not have taken, for example foreign students.

7.2.3 Developing suitable interventions

The prediction model proposed in this thesis accurately classifies $\sim 80\%$ of students as 'weak' or 'strong'. Suitable interventions can now be developed and used to assist such students. Learning technologies are receiving considerable research attention of late and possible future work could look at the development of an online learning system, for example, a tutoring system tailored to provide remedial support for weaker students.

7.2.4 Non-Completing Students

A considerable amount of data was gathered from students who subsequently dropped-out of the programming module at each of the participating institutions. A study analysing this data could lead to insight as to why students do not continue with programming and could be useful in developing interventions to encourage and support students to persevere with the module.

7.2.5 Targeting Secondary School Students

The naïve Bayes prediction model developed in this thesis can be used to predict the introductory programming performance of students when they have experienced only preliminary programming concepts. As such, this tool with minor adjustment, could be employed by guidance councillors in secondary schools to assess a students likely success in programming after similar concept exposure. Research on an alterative mathematics measure would be required.

7.2.6 Application of the Approach to Other Computer Science Topics

It would be useful to determine how effective the naïve Bayes model is at predicting performance in other computer science topics, for example, discrete mathematics. The only modification required to the instruments would be to re-word the programming self-esteem measure to represent the new topic. Where students study computer science as part of a science or arts degree, it would also be useful to determine how well the model predicts performance on the associated science or arts modules. Such a predictive model would be a highly significant contribution to the education community and again future research could investigate its effectiveness when measured in secondary schools.

7.2.7 Optimising Results of Machine Learning Algorithms

This thesis provides a baseline for further studies on the application of machine learning techniques to predict programming performance. The use of an ensemble of mixed models appears fruitful and justifies the examination of further techniques other than Stacking and Voting. Although Bagging and Boosting did not result in an improved performance of the naïve Bayes algorithm, other alternative optimisations could be examined, for example, tree augmented naïve Bayes. The results of the RBF SVM implementation were encouraging and subsequent studies could be carried out to gather more data that can be used to investigate this approach further.

Appendix A

Background Survey

1. Please circle the level of mathematics studied for the Irish Leaving Certificate and the grade achieved. If this is not applicable *circle* $N\backslash A$

Higher Lower Foundation $N\backslash A$

A1 A2 B1 B2 B3 C1 C2 C3 D1 D2 D3

If you selected $N\backslash A$ please explain why:

____

2. Did you take a science subject in the Leaving Certificate? If yes, please enter the subject name, the level studied and the grade achieved.

Subject:	Level	Grade:
______	______	______
______	______	______
______	______	______

3. Tick each statement that applies to you:

__ I took an object-oriented programming course e.g. Java, C++, prior to this module.

__I took a procedural programming course e.g. PASCAL, BASIC, COBOL prior to this module.

__I took a programming course on another type of language prior to this module. Please specify language: _______

__I took a web design course prior to this module.

__I learned to program on my own by reading books, talking with others, etc. but did not take a class.

4. How many hours per week on average did you spent on the following items **prior** to taking this class?

__Number of hours/week surfing the web and using e-mail.

__Number of hours/week playing computer games.

__Number of hours/week using application software such as word-processing, spreadsheets etc.

5. How many hours per week on average did you spent on the following items **while** taking this class?

__Number of hours/week surfing the web and using e-mail.

__Number of hours/week playing computer games.

__Number of hours/week using application software such as word-processing, spreadsheets etc.

6. Why did you take this course:

__There is good money in computers

__I like computers /programming

__It was something to do

__I thought I might be good at it

__I wanted to prove I could do it

__None of the above

If you selected 'None of the above' please explain why:

____

7. Did someone give you encouragement and/or the desire to study computer science either by words or role modelling? Yes   No

If yes, put a tick by the most important person. (Choose only one)

__Parent / Guardian

__Other family member

__Teacher or guidance counsellor or other secondary school staff

__Friend / peer

__Professional in the computer field (does not have to be an acquaintance)

__College recruitment

__Other

If you selected 'Other' please give details:

____

8. Indicate the number of hours on average per week you have worked at a part-time job while taking this course:

__None

__1 - 5

__6 -10

__11-15

__16+

9. Indicate the number of hours per week available to you out of class for studying

the material on this module:

__None

__1 - 5

__6 -10

__11-15

__16+

10. Being as honest as you can indicate the likely number of hours per week that you believe you will actually spend studying the material on this module:

__None

__1 - 5

__6 -10

__11-15

__16+

11. When given a programming assignment or when a test is coming up which method would you prefer? (check one)

__Individual / competitive work or study

__Co-operative / group work or study

Appendix B

Cantwell-Wilson and Shrock Comfort-Level measure

The following questions are based on a larger set of questions used in a study by Cantwell-Wilson and Shrock [CW00]. The questions focus on the same issues as the original questionnaire but are re-structured to reduce the length of completion time.

Please read each of the following items completely and circle the number that most accurately reflects how you feel about each item.

1. Always uncomfortable	2. Sometimes uncomfortable	3. Neutral	4. Sometimes comfortable	5. Always comfortable

Asking questions in lectures	1 2 3 4 5
Asking questions in lab	1 2 3 4 5
Answering questions in lectures	1 2 3 4 5
Going to the lecturer after a lecture to ask a question about the lecture or an assignment	1 2 3 4 5
Going to the lecturer's office to ask a question about the lecture or an assignment	1 2 3 4 5

Please read each of the following items completely and circle the number that most accurately reflects how you feel about each item.

1. Very difficult	2. Mostly difficult	3. Neutral	4. Mostly easy	5. Very easy

Understanding Java programming concepts	1 2 3 4 5
Designing the logic of a program without help	1 2 3 4 5
Completing lab assignments	1 2 3 4 5

How do you rate your level of understanding of the programming module:

__ Higher than others in the class

__ Higher than most of the class

__ Average

__ Lower than most of the class

__ Lower than any of the others in the class.

Appendix C

Self-efficacy for Computer Programming

The Computer Programming Self-Efficacy Scale [RW98] consists of 33 items that ask students to judge their capabilities in a wide range of programming tasks and situations. As this instrument was administered when students had very limited experience of the programming module, a shortened version of this scale using only seven questions about simple programming tasks was used.

Below is a list of statements dealing with your general feelings about your Java* programming ability. If you strongly agree, circle 1. If you agree with the statement, circle 2. If you disagree, circle 3. If you strongly disagree, circle 4.

1. Not at all confident	2. Mostly not confident	3. Slightly confident	4. 50/50	5. Fairly confident	6. Mostly confident	7. Absolutely confident

1.	I can write syntactically correct Java statements.	1 2 3 4 5 6 7
2.	I understand the language structure of Java and the usage of the reserved words.	1 2 3 4 5 6 7
3.	I can write logically correct blocks of code using Java.	1 2 3 4 5 6 7
4.	I can write a Java program that displays a greeting message.	1 2 3 4 5 6 7
5.	I can write a Java program that computes the average of three values.	1 2 3 4 5 6 7
6.	I can write a Java program that computes the average of any given number of values.	1 2 3 4 5 6 7
7.	I can write a small Java program given a small problem that is familiar to me.	1 2 3 4 5 6 7

* Where students were studying Pascal or Visual Basic, an identical questionnaire with the appropriate language was administered.

Appendix D

Programming Self-Esteem Scale

The Rosenberg Self-Esteem (RSE) questionnaire ([Ros65]) was adapted to apply to programming self-esteem. The scale consists of 10 questions and has been shown to have generally inter-item and test-retest reliability. Each of the questions were re-worded to relate to programming self-esteem and not to self-esteem directly, for example the first question was changed from 'On the whole, I am satisfied with myself' to 'On the whole, I am satisfied with my Java programming progress'.

Below is a list of statements dealing with your general feelings about your Java programming ability. If you strongly agree, circle 1. If you agree with the statement, circle 2. If you disagree, circle 3. If you strongly disagree, circle 4.

1. Strongly agree	2. Agree	3. Disagree	4. Strongly disagree

1.	On the whole, I am satisfied with my Java programming progress.	1 2 3 4
2.	At times I think I am no good at all at Java programming.	1 2 3 4
3.	I feel that I have a number of good Java programming qualities.	1 2 3 4
4.	I am able to do Java programming as well as most other students in my class.	1 2 3 4
5.	I feel I do not have much Java programming ability to be proud of.	1 2 3 4
6.	I certainly feel useless at Java programming at times.	1 2 3 4
7.	I feel that I'm a person of worth, at least on an equal plane with the other Java programmers in my class.	1 2 3 4
8.	I wish I could have more respect for my Java programming ability.	1 2 3 4
9.	All in all, I am inclined to feel I am a failure at Java programming.	1 2 3 4
10.	I take a positive attitude towards my Java programming ability.	1 2 3 4

Appendix E

Motivated Strategies for Learning Questionnaire

The Motivated Strategies for Learning Questionnaire (MSLQ) is a self-report instrument designed by Pintrich *et al.* to measure students' motivation and self-regulated learning in classroom contexts [PSGM91].

The following questions ask about your motivation for and attitudes about this class. Remember there are no right or wrong answers, just answer as accurately as possible. Use the scale below to answer the questions. If you think the statement is very true of you, circle the 7; if a statement is not at all true of you, circle the 1. If the statement is more or less true of you, find the number between 1 and 7 that best describes you.

1. Not at all true of me	2.	3.	4.	5.	6.	7.Very true of me

1	In a class like this, I prefer course material that really challenges me so I can learn new things	1 2 3 4 5 6 7
2	If I study in appropriate ways, than I will be able to learn the material in this course	1 2 3 4 5 6 7
3	When I take a test I think about how poorly I am doing compared with other students	1 2 3 4 5 6 7
4	I think I will be able to use what I learn in this course in other courses	1 2 3 4 5 6 7
5	I believe I will receive an excellent grade in this class	1 2 3 4 5 6 7
6	I'm certain I can understand the most difficult material presented in the readings for this course	1 2 3 4 5 6 7
7	Getting a good grade in this class in the most satisfying thing for me right now	1 2 3 4 5 6 7
8	When I take a test I think about items on other parts of the test I can't answer	1 2 3 4 5 6 7
9	It is my own fault if I don't learn the material in this course	1 2 3 4 5 6 7
10	It is important for me to learn the course material in this class	1 2 3 4 5 6 7
11	The most important thing for me right now is improving my overall grade point average, so my main concern in this class is getting a good grade	1 2 3 4 5 6 7
12	I'm confident I can learn the basic concepts taught in this course	1 2 3 4 5 6 7
13	If I can, I want to get better grades in this class than most of the other students	1 2 3 4 5 6 7

continued on next page

continued from previous page

14	When I take tests I think of the consequences of failing	1 2 3 4 5 6 7
15	I'm confident I can understand the most complex material presented by the instructor in this course	1 2 3 4 5 6 7
16	In a class like this, I prefer course material that arouses my curiosity, even if it difficult to learn	1 2 3 4 5 6 7
17	I am very interested in the content area of this course	1 2 3 4 5 6 7
18	If I try hard enough, then I will understand the course material	1 2 3 4 5 6 7
19	I have an uneasy, upset feeling when I take an exam	1 2 3 4 5 6 7
20	I'm confident I can do an excellent job on the assignments and tests in this course	1 2 3 4 5 6 7
21	I expect to do well in this class	1 2 3 4 5 6 7
22	The most satisfying thing for me in this course is trying to understand the content as thoroughly as possible	1 2 3 4 5 6 7
23	I think the course material in this class is useful for me to learn	1 2 3 4 5 6 7
24	When I have the opportunity in this class, I choose course assignments that I can learn from, even if they don't guarantee a good grade	1 2 3 4 5 6 7
25	If I don't understand the course material, it is because I didn't try hard enough	1 2 3 4 5 6 7
26	I like the subject matter of this course	1 2 3 4 5 6 7
27	Understanding the subject matter of this course is very important to me	1 2 3 4 5 6 7
28	I feel my heart beating fast when I take an exam	1 2 3 4 5 6 7

continued on next page

continued from previous page

29	I'm certain I can master the skills being taught in this class	1 2 3 4 5 6 7
30	I want to do well in this class because it is important to show my ability to my family, friends employer of others	1 2 3 4 5 6 7
31	Considering the difficulty of this course, the teacher, and my skills, I think I will do well in this class	1 2 3 4 5 6 7
32.	When I study the readings for this course, I outline the material to help me organise my thoughts	1 2 3 4 5 6 7
33.	During class time, I often miss important points because I'm thinking of other things	1 2 3 4 5 6 7
34.	When studying for this course, I often try to explain the material to a classmate or friend	1 2 3 4 5 6 7
35.	I usually study in a place where I can concentrate on my coursework	1 2 3 4 5 6 7
36.	When reading for this course, I make up question to help focus my reading	1 2 3 4 5 6 7
37.	I often feel so lazy or bored when I study for this class that I quit before I finish what I planned to do	1 2 3 4 5 6 7
38	Often find myself questioning things I hear or read in this course to decide if I find them convincing	1 2 3 4 5 6 7
39.	When I study for this class, I practice saying the material to myself over and over	1 2 3 4 5 6 7
40.	Even if I have trouble learning the material in this class, I try to do the work on my own, without help from anyone	1 2 3 4 5 6 7

continued on next page

continued from previous page

41.	When I become confused about something I'm reading for this class, I go back and try to figure it out	1 2 3 4 5 6 7
42.	When I study for this course, I go through the readings and my class notes and try to find the most important ideas	1 2 3 4 5 6 7
43.	I make good use of my study time for this course	1 2 3 4 5 6 7
44.	If course readings are difficult to understand, I change the way I read the material	1 2 3 4 5 6 7
45.	I try to work with other students from this class to complete the course assignments	1 2 3 4 5 6 7
46.	When studying for this course, I read my class notes and course reading over and over again	1 2 3 4 5 6 7
47.	When a theory, interpretation or conclusion is presented in class or in the readings, I try to decide if there is good supporting evidence	1 2 3 4 5 6 7
48.	I work hard to do well in this class even if I don't like what we are doing	1 2 3 4 5 6 7
49.	I make simple charts, diagrams, or tables to help me organise course material	1 2 3 4 5 6 7
50.	When studying for this course, I often set aside time to discuss course material with a group of students from the class	1 2 3 4 5 6 7
51.	I treat the course material as a starting point and try to develop my own ideas about it	1 2 3 4 5 6 7
52.	I find it hard to stick to a study schedule	1 2 3 4 5 6 7

continued on next page

continued from previous page

53.	When I study for this class, I pull together information from different sources, such as lectures, readings and discussions	1 2 3 4 5 6 7
54.	Before I study new course material thoroughly, I often skim it to see how it is organised	1 2 3 4 5 6 7
55.	I ask myself questions to make sure I understand the materials I have been studying in this class	1 2 3 4 5 6 7
56.	I try to change the way I study in order to fit the course requirements and the instructor's teaching style	1 2 3 4 5 6 7
57.	I often find that I have been reading for this class but don't know what it was all about	1 2 3 4 5 6 7
58.	I ask the instructor to clarify concepts I don't understand well	1 2 3 4 5 6 7
59.	I memorise key words to remind me of important concepts in this class	1 2 3 4 5 6 7
60.	When course work is difficult, I either give up or only study the easy parts	1 2 3 4 5 6 7
61.	I try to think through a topic to decide what I am supposed to learn from it rather than just reading it over when studying for this course	1 2 3 4 5 6 7
62.	I try to relate ideas in this subject to those in other courses whenever possible	1 2 3 4 5 6 7
63.	When I study for this course, I go over my class notes and make an outline of important concepts	1 2 3 4 5 6 7
64.	When reading for this class, I try to relate the material to what I already know	1 2 3 4 5 6 7

continued on next page

continued from previous page

65.	I have a regular place set aside for studying	1 2 3 4 5 6 7
66.	I try to play around with ideas of my own related to what I am learning in this course	1 2 3 4 5 6 7
67.	When I study for this course, I write brief summaries of the main ideas from the readings and my class notes	1 2 3 4 5 6 7
68.	When I can't understand the material in this course, I ask another student in this class for help	1 2 3 4 5 6 7
69.	I try to understand the material in this class by making connections between the readings and the concepts from the lectures	1 2 3 4 5 6 7
70.	I make sure that I keep up with the weekly readings and assignments for this course	1 2 3 4 5 6 7
71.	Whenever I read or hear an assertion or conclusion in this class, I think about possible alternatives	1 2 3 4 5 6 7
72.	I make lists of important items for this course and memorise the lists	1 2 3 4 5 6 7
73.	I attend this class regularly	1 2 3 4 5 6 7
74.	Even when course materials are dull and uninteresting, I manage to keep working until I finish	1 2 3 4 5 6 7
75.	I try to identify students in this class whom I can ask for help if necessary	1 2 3 4 5 6 7
76.	When studying for this course I try to determine which concepts I don't understand well	1 2 3 4 5 6 7
77.	I often find that I don't spend very much time on this course because of other activities	1 2 3 4 5 6 7

continued on next page

continued from previous page

78.	When I study for this class, I set goals for myself in order to direct my activities in each study period	1 2 3 4 5 6 7
79.	If I get confused taking notes in class, I make sure I sort it out afterwards	1 2 3 4 5 6 7
80.	I rarely find time to review my notes or readings before an exam	1 2 3 4 5 6 7
81.	I try to apply ideas from course readings in other class activities such as lecture and discussion	1 2 3 4 5 6 7

Appendix F

Sample Cognitive Test Questions

The following questions have been kindly provided by Ms. Jacqueline McQuillan, Department of Computer Science, NUI Maynooth (jmcq@cs.nuim.ie). The cognitive test measured a number of cognitive abilities including number sequencing ability, letter sequencing ability, arithmetic reasoning ability, procedural ability and ability to follow simple syntax. A sample question and answer within each of these categories is provided.

F.1 Number Sequencing Ability

What is the next number in this sequence?
2, 4, 16,
Answer: 256

F.2 Letter Sequencing Ability

Which letter should come next in this sequence?
Z, X, U, Q,

Answer: L

F.3 Arithmetic Reasoning Ability

The area of a rectangular field is 24 m2 and its perimeter is 20 m. Given that its length is bigger that its width, what is the width of the field?

Answer: 4 m

F.4 Procedural Ability

Follow the steps below.

Box Number	1	2	3	4	5	6
Contents	3	6	-4	1	-2	3

Step 1. Double the contents of box number 5

Step 2. Add the result from step 1 to the contents of box number 4 and put the result in box number 6

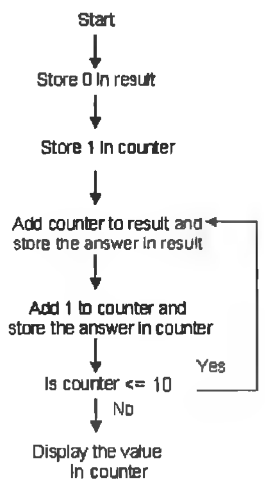
Step 3. Add the square of the contents of box number 6 to box number 1 and put the result in box number 2

What is the value in box number 2?

Answer:12

F.5 Problem Translation Skills

What value will be printed?


Start
Store 0 in result
Store 1 in counter
Add counter to result and store the answer in result
Add 1 to counter and store the answer in counter
Is counter <= 10
Yes
No
Display the value in counter

Answer: 11

Appendix G

Consent Form

Research project:	Factors that influence performance on an introductory programming module.
Researchers:	Susan Bergin and Professor Ronan Reilly (Supervisor), Department of Computer Science, NUI Maynooth, Maynooth, Co. Kildare
Contact details:	sbergin@cs.nuim.ie, ronan@cs.nuim.ie

I, (Name of research subject), agree to participate in the research conducted by Susan Bergin and Ronan Reilly on factors that influence novice programming performance at the Department of Computer Science, National University of Ireland Maynooth. The project is under the supervision of Ronan Reilly. The purpose of the research is to determine factors that may predict a student's overall score on an introductory programming module. My participation will consist essentially of participating in no more than eight fifteen-minute sessions (2 hours in total) during the academic year X (either 2003-2004 / 2004-2005 / 2005-2006). During these sessions I may be asked to answer questions on my academic background, my

previous computer experience and on my experience participating in an introductory programming module. I may also be asked to solve puzzles and aptitude test items using paper and pencil or on a computer. The data gathered will only be used by the researchers and the findings may be published in suitable conferences, journals and in the PhD thesis of Susan Bergin. I understand my confidentiality will be respected. I have received assurance from the researchers that the information I will share will remain strictly confidential and that no information that discloses my identity will be released or published without my specific consent to the disclosure. I am free to withdraw from the study at any time and can refuse to answer any of the questions asked or to participate in any of the tasks. All data gathered will be stored in a secure manner and only the above named researchers will have access to it. There are two copies of the consent form, one of which I may keep. If I have any questions about the research project, I may contact Susan Bergin or her supervisor Professor Ronan Reilly at the email addresses provided above.
Principal Researcher's signature: (Signature) Date: (Date)
Research Subject's signature: (Signature) Date: (Date)

Appendix H

Student Information Sheet

Research project:	Factors that influence performance on an introductory programming module.
Researchers:	Susan Bergin and Professor Ronan Reilly (Supervisor), Department of Computer Science, NUI Maynooth, Maynooth, Co. Kildare
Contact details:	sbergin@cs.nuim.ie, ronan@cs.nuim.ie

I would like to invite you to participate in this study, which is concerned with identifying factors that influence performance on an introductory programming module. The study is part of the PhD research being carried out by Susan Bergin at the Department of Computer Science, NUI Maynooth. It is hoped that the findings of the project would help lecturers and course coordinators to identify students that may have difficulty with the module and to put in place appropriate facilities to assist them.

Participation will consist of no more than eight fifteen-minute sessions (two hours in total). During these sessions you may be asked to answer questions on your academic background, your previous computer experience and on your experience participating in the introductory programming module (CS100/SE101) at the Department of Computer Science, NUI Maynooth. You may also be asked to solve puzzles and aptitude test items using paper and pencil or on a computer. When the PhD thesis has been completed a summary of the findings will be produced which can be sent to you if you are interested. The data gathered will only be used by the the above mentioned researchers and the findings may be published in suitable conferences, journals and in the PHD thesis of Susan Bergin. Your participation would be on a confidential basis and no information that discloses your identity will be released or published. Your participation in this project is entirely voluntary. You are not obliged to take part, you have been asked to participate in this study because you are participating in an introductory programming module. This does not mean you have to. If you do not wish to take part you do not have to give a reason. Similarly, if you do agree to participate you are free to withdraw at any time during the study if you change our mind. Additionally, you can agree to participate but refuse to answer a question asked of you or to participate in a task. If you have any questions about this study, you may contact Susan Bergin at sbergin@cs.may.ie

Cited References

[AR94] H. Anton and A. Rorres. *Elementary linear algebra.* Wiley, 7 edition, 1994.

[Aus87] H.S. Austin. Predictors of pascal programming achievement for community college students. *SIGCSE '87: Proceedings of the eighteenth SIGCSE technical symposium on Computer science education*, pages 161–164, 1987.

[BL01] P. Byrne and G. Lyons. The effect of student attributes on success in programming. *ACM SIGCSE Bulletin*, 33(3):49–52, 2001.

[BR05a] S. Bergin and R. Reilly. Examining the role of self-regulated learning on introductory programming performance. *Proceedings of the 2005 international Workshop on Computing Education Research*, pages 81–86, 2005.

[BR05b] S. Bergin and R. Reilly. The influence of motivation and comfort-level on learning to program. *Proceedings of the 17th Workshop on Psychology of Programming*, pages 293–304, 2005.

[BR05c] S. Bergin and R. Reilly. Programming: Factors that influence success. *ACM SIGCSE Bulletin*, 37(1):411–415, 2005.

[BR06a] S. Bergin and R. Reilly. A computational model to predict introductory programming performance. Technical report, NUIM Technical Report Series NUIM-CS-TR-2006-01, January 2006.

[BR06b] S. Bergin and R. Reilly. Predicting programming performance using machine learning techniques. *NUIM Postgraduate Symposium*, March 2006.

[BR06c] S. Bergin and R. Reilly. Predicting introductory programming performance: a multi-institutional multivariate study. *Computer Science Education*, Submitted January 2006.

[BR06d] S. Bergin and R. Reilly. Using machine learning techniques to predict introductory programming performance. *Applied Artificial Intelligence*, Submitted May 2006.

[BU83] R.J. Barker and E. A. Unger. A predictor for success in an introductory programming class based upon abstract reasoning development. *ACM SIGCSE Bulletin*, 15(1):154–158, 1983.

[Bur98] C.J. Burges. A tutorial on support vector machines for pattern recognition. *Knowledge Discovery and Data Mining*, 2(2):121–167, 1998.

[Cam02] C. Campbell. Kernel methods: a survey of current techniques. *Neurocomputing*, 48:63–84, 2002.

[CL01] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`, 2001.

[CW00] B. Cantwell-Wilson. *Contributing factors to success in computer science: a study of gender differences.* PhD thesis, Southern Illinois University at Carbondale, 2000.

[CWS01] B. Cantwell-Wilson and S. Shrock. Contributing to success in an introductory computer science course: A study of twelve factors. *ACM SIGCSE Bulletin*, 33(1):184–188, 2001.

[DHS01] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification.* John Wiley and Sons, Inc, second edition, 2001.

[Die97] T.G. Dietterich. Machine-learning research: four current directions. *AI Magazine*, 18(4):97–136, 1997.

[ES89] G.E. Evans and M.G. Simkin. What best predicts computer proficiency? *Communications of the ACM*, 32(11):1322–1327, 1989.

[FP04] S. Fincher and M. Petre. *Computer Science Education Research.* Taylor and Francis, 2004.

[Gib00] D.C. Gibbs. The effect of a constructivist learning environment for field-dependent/independent students on achievement in introductory computer programming. *ACM SIGCSE Bulletin*, 32(1):207–211, 2000.

[GR00] A. Goold and R. Rimmer. Factors affecting performance in first-year computing. *ACM SIGCSE Bulletin*, 32(2):39–43, 2000.

[Haw99] S. Hawkin. *Neural Networks: A comprehesive foundation.* Prentice Hall, Inc, second edition, 1999.

[Hin95] P.R. Hinton. *Statistics Explained: A guide for social science students.* Routledge, 1995.

[HM00] D. Hagan and S. Markham. Does it help to have some programming experience before beginning a computing degree program? *ACM SIGCSE Bulletin*, 32(3):25–28, 2000.

[Hos83] T.R. Hostetler. Predicting student success in an introductory programming course. *ACM SIGCSE Bulletin*, 15(3):40–43, 1983.

[HW86] L. Honour-Werth. Predicting student performance in a beginning computer science class. *ACM SIGCSE Bulletin*, 18(1):138–143, 1986.

[HW04] E. Holden and E. Weeden. The impact of prior experience in an information technology programming course sequence. *Proceedings of the 4th conference on information Technology Curriculum*, pages 41–46, 2004.

[KPP04] S. Kotsiantis, C. Pierrakeas, and P. Pintelas. Predicting students' performance in distance learning using machine learning techiques. *Applied Artificial Intelligence*, 18:411–426, 2004.

[Kur80] B.L. Kurtz. Investigating the relationship between the development of abstract reasoning and performance in an introductory programming class. *ACM SIGCSE Bulletin*, 12(1):110–117, 1980.

[KWS83] J. Konvalina, S.A. Wileman, and L.G. Stephens. Math proficiency: a key to success for computer science students. *Communications of the ACM*, 26(5):377–382, 1983.

[Lar05] D.T. Larose. *Discovering Knowledge in Data.* John Wiley and Sons Inc., 2005.

[Lev60] H. Levene. *In Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling.* Stanford University Press, 1960.

[LS82] R.R. Leeper and J.L. Silver. Predicting success in a first programming course. *ACM SIGCSE Bulletin*, 14(1):147–150, 1982.

[MAD$^+$01] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. *ITiCSE-WGR '01: Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 125–180, 2001.

[MDV86] R.E. Mayer, J.L. Dyck, and W. Vilberg. Learning to program and learning to think: what's the connection? *Communications of the ACM*, 29(7):605–610, 1986.

[Men01] S Menard. *Applied Logistic Regression Analysis.* Sage Publications, Inc, second edition, 2001.

[MFK01] M. Morgan, R. Flanagan, and T. Kellaghan. A study of non-completion in undergraduate university courses. *The Higher Education Authority*, 2001.

[Mit97] T. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[MST94] D. Michie, D.J. Spiegelhalter, and C.C. Taylor. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, 1994.

[New75] Peter R. Newsted. Grade and ability predictions in an introductory programming course. *ACM SIGCSE Bulletin*, 7(2):87–91, 1975.

[OBG$^+$04] J. O'Kelly, S. Bergin, J. Ghent, P. Gaughran, S. Dunne, and A.Mooney. An overview of the integration of problem based learning into an existing computer science programming module. *PBL International Conference*, 2004.

[Pal05] J. Pallant. *SPSS Survival Manual.* Open University Press, second edition, 2005.

[Pam00] F.C. Pampel. *Logistic Regression: A Primer.* Sage Publications, Inc, 2000.

[Pat96] D.W. Patterson. *Artificial Neural Networks: Theory and applications.* Prenicd Hall, 1996.

[PB90] P. Pokay and P. Blumenfeld. Predicting achievement early and late in the semester: the role of motivation and learning strategies. *Journal of Educational Psychology*, 82(1):41–50, 1990.

[PBV00] F. Pajares, S. Brinter, and G. Valiante. Relation between achievement goals and self-beliefs of middle school students in writing and science. *Contemporary Educational Psychology*, 25(4):406–422, 2000.

[PD90] P.R. Pintrich and E. DeGroot. Motivational and self-regulated learning components of classroom academic performance. *Journal of Educational Psychology*, 82(1):33–40, 1990.

[PG91] P.R. Pintrich and T. Garcia. Student goal orientation and self-regulation in the college classroom. *Advances in Motivation and Achievement*, 7:371–403, 1991.

[Pin89] P.R. Pintrich. The dymanic interplay of student motivation and cognition in the college classroom. *Advances in motivation and achievement: motivation enhancing environments*, 6:117–160, 1989.

[Pin99] P.R. Pintrich. The role of motivation in promoting and sustaining self -regulated learning. *International Journal of Educational Research*, 31:459–470, 1999.

[PRP99] H. Patrick, A.M. Ryan, and P.R. Pintrich. The differential impact of extrinsic and mastery goal orientations on males' and females' self-regulating learning. *Learning and Individual differences*, 11(2):153–171, 1999.

[PSGM91] P.R. Pintrich, D. Smith, T. Garcia, and W. McKeachie. A manual for the use of the motivated strategies for learning questionnaire. technical report 91-b-004. Technical report, The Regents of The University of Michigan, 1991.

[RHW86] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by back-propagating errors. *Nature*, 323(9):533–536, 1986.

[RLW04] V. Ramalingam, D. LaBelle, and S. Wiedenbeck. Self-efficacy and mental models in learning to program. *ACM SIGCSE Bulletin*, 36(3):171–175, 2004.

[Ros65] M. Rosenberg. *Society and the adolescent self image.* Princeton University Press, Princeton, NJ, 1965.

[RP03] S. Russell and P.Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, Inc, second edition, 2003.

[RRR02] N. Rountree, J. Rountree, and A. Robbins. Predictors of success and failure in a cs1 course. *ACM SIGCSE Bulletin*, 34(4):121–124, 2002.

[RW98] V. Ramalingham and S. Wiedenbeck. 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, 1998.

[See02] A.K. Seewald. How to make stacking better and faster while also taking care of an unknown weakness. *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 554–561, 2002.

[Smi02] L.I. Smith. A tutorial on principal components analysis. http://kybele.psych.cornell.edu/l.pdf, 2002.

[SS02] B. ScholKopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, 2002.

[Ste02] M.V. Stein. Mathematical preparation as a basis for success in cs-iie. *Journal of Computing in Small Colleges*, 17(4):28–38, 2002.

[TF01] B.G. Tabachnick and L.S. Fidell. *Using Multivariate Statistic.* Allyn and Bacon, fourth edition, 2001.

[TK99] S. Theodoridis and K. Koutroumbas. *Pattern Recongition.* Academic press, first edition, 1999.

[Ven02] P.R. Ventura. Identifying predictors of success for an objects-first cs1. *Computer Science Education*, 15(3):223–243, 2002.

[Ven03] P. Ventura. *On the Origins of Programmers: Identifying Predictors of Success for an Objects-First CS1.* Department of computer science and engineering, The State University of New York at Buffalo, 2003.

[WE05] I. Witten and E.Frank. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann Publishers, second edition, 2005.

[Wie05] S. Wiedenbeck. Factors affecting the success of non-majors in learning to program. *Proceedings of the 2005 international Workshop on Computing Education Research*, pages 13–24, 2005.

[Wol92] D.H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[Zim86] B. Zimmerman. Becoming a self-regulated learner: which are the key sub-processes? *Contemporary Educational Psychology*, 11(4):307–313, 1986.

[ZMP90] B. Zimmerman and M. Martinez-Pons. Student differences in self-regulated learning: relating grade, sex and giftedness to self-efficacy and strategy use. *Journal of Educational Psychology*, 82(1):51–59, 1990.