2007

Neural Network Prediction of Math and Reading Proficiency as Reported in the Educational Longitudinal Study 2002 Based on Non-Curricular Variables

Jason Brown

NEURAL NETWORK PREDICTION OF MATH AND READING PROFICIENCY AS REPORTED IN THE EDUCATIONAL LONGITUDINAL STUDY 2002 BASED ON NON-CURRICULAR VARIABLES

by

Jason D. Brown

Submitted in partial fulfillment of

the requirements for the degree

Doctor of Education

Instructional Technology Doctoral program

School of Education

Duquesne University

December 2007

DUQUESNE UNIVERSITY
SCHOOL OF EDUCATION

Dissertation

Submitted in Partial Fulfillment of the Requirements
For the Degree of Doctor of Education (Ed.D.)

EdDIT Doctoral Program

Presented by:

Jason D. Brown
M.S., Environmental Science and Management, Duquesne University, 2003
B.S., Computer Science/Math, Duquesne University, 1989

September 21, 2007

NEUTRAL NETWORK PREDICTION OF MATH AND READING PROFICIENCY AS REPORTED IN THE EDUCATIONAL LONGITUDINAL STUDY 2002 BASED ON NON-CURRICULAR VARIABLES

Approved by:

_____, Chair
Connie Moss, Ed.D.
Associate Professor, Department of Foundations and Leadership

_____, Member
Misook Heo, Ph.D.
Assistant Professor, Department of Instruction and Leadership in Education

_____, Member
Susan M. Brookhart, Ph.D.
Senior Research Associate, CASTL

Abstract
NEURAL NETWORK PREDICTION OF MATH AND READING PROFICIENCY AS REPORTED IN THE EDUCATIONAL LONGITUDINAL STUDY 2002 BASED ON NON-CURRICULAR VARIABLES
Jason Brown
Doctor of Education, December 2007
Duquesne University
Chair: Connie Moss, Ed.D.

Predicting student achievement is often the goal of many studies, and a frequently employed tool for constructing predictive models is multiple linear regression. This research sought to compare the performance of a three-layer back propagation neural network to that of traditional multiple linear regression in predicting math and reading proficiency from 103 non-curricular variables collected in the National Center for Educational Statistics' 2002 Educational Longitudinal Study. The neural network model was implemented using the Java programming language and the coefficients for the regression equations were produced by SPSS. The results showed that, for this data set, neither model provided an advantage over the other in terms prediction accuracy when presented with error-free cases. When synthetic noise was introduced into the data, however, the neural network model showed a greater resistance to degradation. The fact that the neural network model performed as well as, and in some cases better than, regression suggests that further study of neural network modeling is warranted to better understand the most effective ways to harness this flexible modeling technology.

ACKNOWLEDGEMENTS

Conventional wisdom suggests that doctoral programs are not to be undertaken while one is in the midst of moving, changing jobs, or having children. The successful end to this journey, however, is testament to the fact that one can succeed under these circumstances for the simple fact that while a dissertation is the product of one person, it is the result of a much larger network of individuals that support and guide the one person to their goal.

First and foremost I would like to thank my committee members Dr. Sue Brookhart and Dr. Misook Heo as well as my committee chair Dr. Connie Moss. When corporate mergers or the arrival of my third child drew my attention away from my writing you waited patiently for me to resume my task, never losing faith, never tiring of reviewing my drafts, and consistently providing just the right feedback to help develop the manuscript into the polished work it is today.

I would also like to thank my fellow cohort members from Cohort I of the EdDIT program at Duquesne University. We embarked together on a journey of discovery and along the way formed new friendships, sharing in each other's lives. Many of you provided feedback on my early research ideas and all of you contributed to my appreciation of the program.

Finally, it was the never-ending encouragement and love of my wife Leslie, my two daughters, Natalie and Meredith, and my son, Andrew, that carried me to my goal. Thanks for waiting. The weekends are ours once again.

TABLE OF CONTENTS

Table of Contents (cont.)

Table of Contents (cont.)

LIST OF TABLES

Page

List of Tables (cont.)

LIST OF FIGURES

Page

List of Figures (cont.)

CHAPTER 1

INTRODUCTION

Overview

Assessing student performance has long been a goal of educators, administrators, and academicians. By accurately predicting which students are at risk, interventions can happen earlier in an effort to increase the overall success rate of a given student population.

Traditionally, though, predicting student outcomes has been accomplished through the application of traditional statistical tests such as correlation, ANOVA, MANOVA, and regression. A limitation of these analyses, though, is the need for a hypothesis that describes the interrelationships between the variables being studied. Consequently, statistical studies often focus on small sets of variables in order to make the model understandable and the mathematics manageable.

Streifer and Schumann, in their 2005 work involving data mining techniques, made this same observation when they wrote "Traditional analytics can neither easily nor systematically handle the complexities of school data to address the queries school leaders have about achievement" (Streifer & Schumann, 2005). Neural networks, however, may provide an alternative analysis that can be employed with complex data sets having no suitable hypothesis.

In a direct comparison of regression and neural network approaches, Snyder observes that regression models require a specific hypothesis, which leads to a specific model that introduces difficulties in handling imperfect data. He then contrasts this with

neural networks in which the only decisions that need to be made are how many hidden layers to use, the number of neurons per layer, the neuron transfer function, learning rate, and training algorithm (Snyder, 1996).

Neural networks take a black box approach to model simulation in that few, or no, assumptions are made about the interactions of the variables. In neural network modeling identifying as many potentially involved variables as needed, along with a suitable set of sample data for training purposes, is possible. The neural network can also be trained to mimic the performance of the known system. Once trained, the neural network can produce accurate outcomes based on new inputs representing the same system.

Neural network models have been used in the past to predict synthetic or man-made systems, such as performance prediction systems in the stock market or the success rates of MBA candidates. Saad (1998) developed a back-propagation neural network model that effectively predicted profit opportunities in the stock market (Saad, 1998). In this study, the future price performance was modeled on past price performance. Additional examples of neural networks in use for prediction can be found in the section Neural Networks in Use in chapter 2.

Neural networks attempt to simulate the biological computation of the brain by constructing layers of simple processing units (neurons) connected through (synaptic) weights. In practice, feed-forward neural networks are presented with a set of input values that are passed to a hidden layer where each neuron computes its own activation potential and produces an output. These outputs feed the inputs to subsequent layers and are either strengthened or diminished by the inter-layer synaptic weights. It is the

synaptic weights that form the "memory" or pattern-recognition abilities of the neural network.

Feed-forward neural networks that employ error back-propagation have shown to be able to accurately model complex systems of many potentially interrelated variables by training with a known data set (Mandic & Chambers, 2001). Back-propagation training of a feed-forward neural network involves presenting a set of input values and allowing the network to compute an output response. The output is then compared with the expected value, an error measurement is made, and the error is propagated back through the network to adjust the synaptic weights. The fact that the network will converge to an optimal set of weights has been shown by Werbos' work on the back-propagation algorithm cited in Mandic & Chambers 2001 text on neural networks for prediction.

Modeling Student Achievement

The following citations, from a brief sampling of the literature, are demonstrative of typical studies, which focus on modeling various factors that may impact achievement. In each of these studies, assumptions were tested about the impact certain variables had on the sample populations.

In 2003, Bridglall and Gordon studied the factors leading to high performance in African American and Latino students in Department of Defense schools (Bridglall & Gordon, 2003). Isaacs studied the impact of counselor interventions (Isaacs, 2003). Lashway produced a work in 2002 that serves as a guidebook for school administrators on mining performance data and reporting (Lashway, 2002). Russell and Zhang

attempted to determine if gender, poverty, or ethnicity impacted performance on the Hawaii State Assessment reading test (Russell & Zhang, 2006). Skidmore modeled gender, motivation, learning strategies, and other affective factors in attempting to predict final exam scores (Skidmore, 2003). Tell and McDonald investigated whether a student's performance in 10th grade could serve as a predictor for their performance in the first year of college (Tell & McDonald, 2003). Crawford, Tindal, and Stieber used students' performance in oral reading exercises to predict performance on statewide achievement tests (Crawford, Tindal, & Stieber, 2001). House and Keeley focused on graduate students in researching the correlation between performance on the Miller Analogies Test (MAT) and the ability to predict subsequent graduate student achievement (House & Keeley, 1993). And finally, Chen, Campbell, and Suleiman attempted to build a prediction model for student performance at a minority professional school using the United States Medical Licensure Examination, Medical College Admission Test score, medical school freshman grade point average, sophomore course performance, and financial aid work-study dollars (Chen, Campbell, & Suleiman, 2001).

Problem Statement

Much of the research on academic achievement has been focused on evaluating student performance after-the-fact by generalizing study populations to other similar populations. These traditional studies, however, typically require a sense of the underlying model and the relationships of the variables in that model. In a sense, one must completely, or nearly completely, understand the inner workings of the system in question.

Developing predictive models of student achievement, as evidenced by the literature, is the focus of real interest within the education community. However, traditional studies continue to focus on model development first, followed by the analysis of a small number of independent variables. It is expected that the pattern recognition capabilities of neural networks will overcome the inaccuracy, lack of generalizability, and limitations on independent variables present in more traditional modeling techniques.

Goal Statement

This research seeks to develop a neural network model that can accurately predict student achievement on standardized math and reading tests and thus provide a mechanism for identifying students who are at risk of under-performing.

To build a neural network model capable of predicting math and reading proficiencies as reported in the Educational Longitudinal Study: 2002 and to test the prediction accuracy of feed-forward, back-propagation neural networks, this study will focus on five groups of non-curricular variables as defined by the Educational Longitudinal Study: 2002 (ELS: 2002). The ELS: 2002 collected data on 15,362 high school sophomores from 752 public, Catholic, and private schools to produce a general-purpose dataset for the study of various educational policy issues. (Ingels et al, 2005).

Significance of the Study

The value in this research comes from its attempt to generalize a predictive framework for student achievement based on non-curricular variables present across all institutions, rather than curricular attributes which are often specific to a given state, school district, or school. Further, the ability to predict student achievement serves

educational administrators, teachers, and parents by identifying those students at risk of underperforming and providing early opportunities for intervention. In the shadow of national assessments such as NCLB, Goals 2000, America 2000, and others (Holbrook, 2003) as well as similar state and local initiatives, the sooner at-risk students or settings can be identified, the sooner interventions can occur.

Limitations of the Models

As with any modeling exercise, certain limitations arise through the data, the mathematical tools, or both. From a data perspective, this study will focus on a known population of high school sophomores. Therefore, the resulting model will only be applicable to other sophomores. Further, the study examines students in Public, Private, and Catholic schools. Data from Charter school students was not available in the ELS: 2002 study. As for the mathematical tools, due to the black box nature of neural network modeling, while the model may prove to be more accurate than traditional regression it will not provide an explanation of what parameters should be changed for an at-risk student in order to increase their chances of success.

Definition of Terms

Accountability. In the context of this study, accountability refers to the responsibility held by schools, school administrators, and teachers to ensure that students are making adequate academic progress.

Axon. An outgoing, branched fiber from a biological neuron, which carries a neuron's signal to the input of other neurons.

Data Driven Decision Making. The act of basing policymaking and other decision processes on quantifiable measures.

Data Mining. The act of using various pattern recognition techniques to search for patterns in a large pool of data.

Dendrite. An incoming, branched fiber of a biological neuron that receives input signals from other neurons.

Dichotomous Output. An output with only two possible states.

Hyperplane. An object in n-dimensional space having n-1 dimensions and dividing the n-dimensional space into two parts. For example, a point is hyperplane that divides a line into two rays; a line is hyperplane that divides a plane into two planes; and a plane is a hyperplane that divides 3-dimensional space into two spaces. The concept can be carried into as many dimensions as required.

Linear Function. A function resulting in a straight line, generally of the form $f(x)=mx+b$.

Logit. The logarithm of the odds of probability p where the odds are expressed as $p/1\text{-}p$.

Neural Network. In the context of this work, a neural network is a collection of artificial neurons arranged in layers with every neuron in a given layer fully connected via synaptic weights to the neurons in the following layer.

Neuron. In the context of a neural network a neuron is the fundamental processing unit that takes a weighted sum of its inputs and passes that value through a transfer function to product an output.

Non-linear Function. A function whose graph does not result in a straight line and for which each point may have a different slope.

Probit. A "probability unit" defined by Charles Bliss used to generate a more or less straight-line plot of probability of the normal distribution. Also, a technique used in regression with a dummy coded (dichotomous) dependent variable.

Synaptic Weight. In the context of this work a synaptic weight connects a neuron in one layer to a neuron in the following layer and is used to strengthen or diminish the output of the first neuron as it passes to the input of neuron in the next layer.

Topology. In the context of this work topology refers to an arrangement of neurons and layers in a neural network.

CHAPTER 2

REVIEW OF THE LITERATURE

Neural Network Overview

Neural Networks have been an on-again, off-again research area in both computer science and cognitive psychology. Neural networks, in a nutshell, are an attempt to model the biological processes that occur in the human brain that allow it to learn, remember, and predict. Fundamentally, the human brain stores and processes information via neurons and the connections formed between neurons. Biologically speaking, a neuron consists of a dendritic tree that collects input signals from other neurons, a cell body, which integrates the inputs and generates a response, and a branching axon that distributes the response to other neurons (Reed & Marks, 1999).

The first neural network model, created by Warren McCulloch and Walter Pitts in the early 1940's featured digital neurons with no learning capability (Blum, 1992). Shortly thereafter, Donald Hebb proposed the idea of Hebbian learning which detailed a method of altering the synaptic weights between neurons that enabled networks to learn. Frank Rosenblatt furthered this idea with his work on what would come to be known as perceptrons, when he published the perceptron convergence theorem, which provided a methodology for updating synaptic weights in a way that would guarantee convergence on an optimal set of weights (Blum, 1992).

Not everyone was convinced of the utility of machine learning based on biological systems, and in the late 60's Marvin Minsky and Seymour Papert worked together to disprove Rosenblatt's claims regarding the usefulness of perceptrons. Minsky

and Papert ultimately discovered that a single layer of perceptrons, no matter how large, could not do something as simple as representing the various states of the exclusive-or function (XOR). Given two inputs, A and B, the XOR function returns true if A = true or B = true and returns false if both A and B = true or both A and B = false. In other words, one or the other must be true, but not both. Seemingly discredited, neural networks faded from the limelight until 1974 when the XOR problem was solved and multi-layer neural networks utilizing a new error propagation algorithm were born (Satinover, 2001).

Werbos developed the back propagation algorithm, which supported learning in multi-layer neural networks and, along with that, the ability to approximate any non-linear function with a sufficiently large network (Blum, 1992). This is not to say that neural networks are without limits. The previous statement "sufficiently large neural network" implies that a mathematical proof may exist for a network topology to be able to approximate any function, however, in practical terms, the network may need to be so large that it is infeasible to implement in practice.

This begs the question, "how do neural networks approximate non-linear functions?" In general terms, a function is something that maps a set of input values to a set of output values. This becomes interesting when we begin to group certain output conditions into recognizable patterns so that certain groups of inputs produce an output that falls in the same region as the other outputs from the group of inputs. In this way, groups of inputs (conditions) become recognizable as belonging to a certain class (output region). A non-linearly separable function produces many such output regions, depending on the function, and requires many hyperplanes to define the regions. Multi-layer neural networks permit the approximation of non-linear functions by being able to produce as

many hyperplanes as needed depending on the size of the neural network (Reed & Marks, 1999).

The premise then is that neural networks can learn from examples, which define the regions (classifications), and then, when presented with new inputs (conditions) correctly classify the new output as belonging to the correct group. It is this sort of supervised learning neural network that is of interest for this study on predicting academic performance.

Neural Network Implementation

Neural networks provide a unique mechanism for the study of student achievement due to the fact that they are often applied in complex pattern recognition problems and also with time-varying sets of data (Mandic & Chambers, 2001). In their simplest form, neural networks are composed of arrays of computational elements each taking one or more input values, performing a computation on the inputs and producing an output which is distributed to one or more additional elements or considered the output of the network as shown in Figure 2-1.

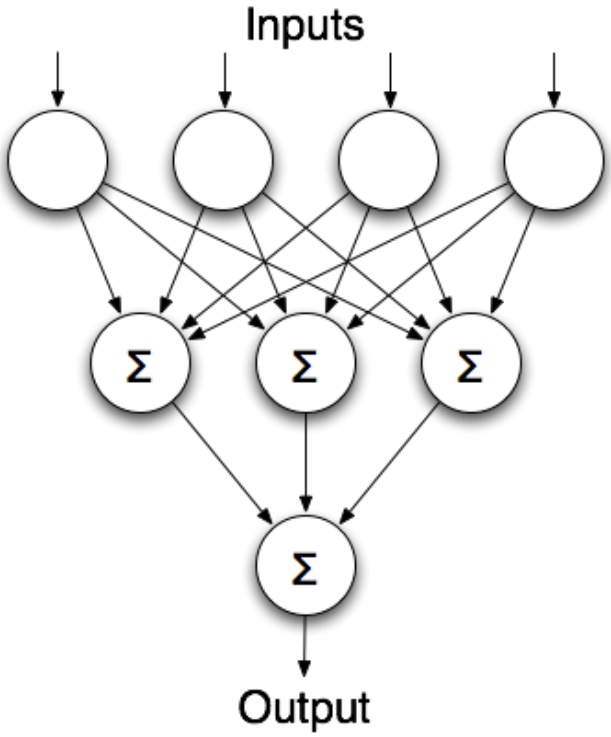


Figure 2-1. Three layer neural network model

Neural networks learn, as it were, by loosely simulating the behavior of biological neurons in the brain in which connections between neurons either favor or inhibit the transmission of signals from one neuron to the next. Whether or not the synaptic connection between two neurons is excitory or inhibitory is the result of learning that has occurred. Excitory connections are typically those with strong synaptic connections that enhance the outputs of neurons in the previous layer and inhibitory connections are those with weak synaptic connections.

In a neural network, the behavior of biological synapses is captured through synaptic weights. Synaptic weights are applied to the inputs, usually by multiplying the

input times its weight, to either strengthen or diminish the signal. Figure 2-2 shows a subset of the connections in the example model with their weights.

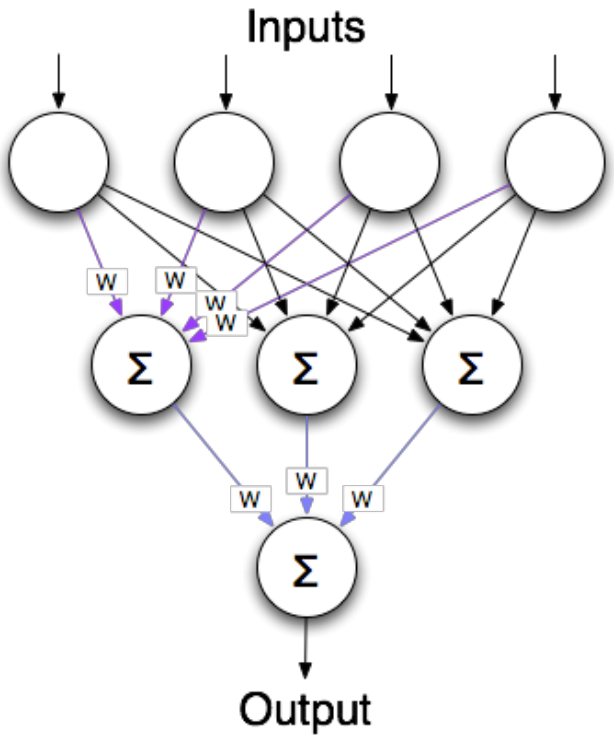


Figure 2-2. Synaptic weighting of connections

The final process in computing the output of a neural network model involves each neuron summing its weighted inputs and then providing that result as input to a non-linear "squashing" function, f(x), in order to constrain the output of the neuron to some known range of values such as 0,1 or -1,1. Common functions include inverse tangent (tan^{-1}) and sigmoid ($1/1+e^{-1}$). These continuous, non-linear functions ensure that neuron outputs fall within a known range as shown in Figure 2-3.

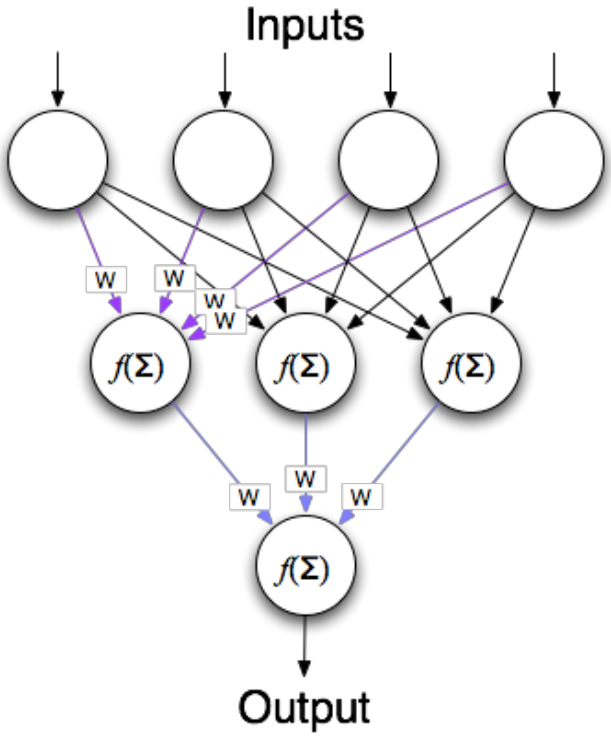


Figure 2-3. Processing neuron outputs through a non-linear function

Neural Networks in Use

One of the attractive features of the neural network approach is that there is no real penalty, save a computational one, for modeling as many input variables as possible. One of the behaviors of neural networks is that they are good at determining which input factors are significant and strengthening those synaptic connections, and which inputs represent noise, or are of little value and diminishing their effect on the network's output. A review of the Institute of Electrical Engineers and the Institute of Electrical and Electronics Engineers (IEE/IEEE) publications demonstrates a growing interest in neural

networks for prediction. In 1990 approximately 10 articles were published. In 1999 that number rose to nearly 140 (Mandic & Chambers, 2001).

In reviewing the literature on the application of neural networks, two facts became evident 1) There has been very little written on the use of neural networks in predicting academic performance; and 2) There has been much more written on the applications of neural networks in other fields. There do, however, seem to be two broad classifications of systems in which neural networks are applied, natural and synthetic.

Naturally occurring systems include topics such as weather, geology, and the environment while synthetic systems focus on man-made topics such as the stock market, finance, information retrieval and academics. The field of medicine was slightly different in that it seems to straddle the line between synthetic and natural systems because the application of neural networks in that field often involves a natural system, such as cancer or heart disease detection, in conjunction with man-made treatments and medical procedures.

In this review of neural networks in use, examples are provided of applications in both natural and synthetic systems. Special emphasis, however, will be placed on applications in education, since that is the central topic of this research.

Natural Systems

The literature reveals that there are many examples of neural network models utilized for predictive purposes. For instance, a neural network was employed to forecast the runoff due to daily precipitation, temperature, and snowmelt for a watershed in Maryland (Tokar & Johnson, 1999). The neural network model in this study compared

favorably to more common statistical regressions or simple conceptual models ordinarily used. In addition, the authors found that the model reduced the necessary size of calibration data and reduced the time necessary to calibrate the model while at the same time producing more accurate predictions in a more flexible manner.

In another work, a neural network was used to aid in predicting tornado formation as a result of updrafts during severe thunderstorms (Marzban & Stumpf, 1996). Data extracted from Doppler radar feeds comprising 23 separate variables were provided as input to a feed-forward neural network, the output of which was the predicted existence of a tornado. Compared to existing rule-based algorithms and discriminant analysis, the neural network model proved to be more accurate.

The prediction of stream flow (flash floods) and water quality was the focus of the 2006 work of Sahoo, Ray, and De Carlo. The researchers in this study employed a back-propagation neural network in the assessment of the quantity and quality of running water in Hawaii. Their model, which used rainfall, stream flow, stream stage and water quality values as inputs accurately predicted the resultant stream flow with an R value of 0.99 (Sahoo, Ray, & De Carlo, 2006). R in the case of this study represented the line of exact fit with 1.0 being a perfect fit between the model and the measured values.

Martynenko and Yang employed a neural network to model the drying characteristics of ginseng. This problem is particularly challenging because ginseng root is still biologically active and counteracts the drying action through internal physiological processes. In particular the drying rate is known to be non-linear at the beginning and end of the drying process (Martynenko & Yang, 2006). By way of validation they compared their neural network model to the performance of previously established models such as

the exponential model and Page's model and found that the neural network model gave the best fit for their experimental data.

In modeling another natural phenomenon, a researcher built a neural network for predicting the ozone forecast in an industrialized urban area (Yi, 1996). This article points out that while more conventional models exist, they need to be more accurate and that the interplay of the meteorological variables and photochemical reactions is complex. As previously discussed in this paper, neural networks are efficient at pattern recognition, and the author of this work used that to his advantage and found that a neural network approach was superior in estimating ozone concentrations over more traditional statistical approaches.

Synthetic Systems

On to more synthetic activities, neural networks are also at use in the area of finance. Researchers at the University of Oklahoma built a neural network for bankruptcy prediction (Odom & Sharda, 1990) and found that it compared favorably to the more traditional method of multivariate discriminant analysis.

Stock prediction is also an area of focus for neural networks and three researchers studied the accuracy of three separate neural network topologies for market forecasting (Saad, Prokhorov & Wunsch, 1998). Their findings were that each of the network layouts was feasible and performed better than conventional stock analysis tools. In a separate text by Mandelbrot and Hudson, The Misbehavior of Markets, they state "Wall Street has long been the computer industry's biggest customer, unleashing 'genetic algorithms,'

'neural networks,' and other computational techniques on the market…" (Mandelbrot & Hudson, 2004).

Exchange rate prediction was the subject of the work of Ince and Trafalis. The challenge presented by this problem is found in the inherent efficiency of financial markets. That is, any information that would make one direction favorable over another is quickly known by all, and the market again equalizes. Nevertheless, the authors compared the non-parametric approaches of support vector regression and artificial neural networks and found that both performed well when coupled with the appropriate input selection (Ince & Trafalis, 2005).

Murat and Ceylan studied the ability to forecast energy transport demand in Turkey in their 2006 work with neural networks. The authors constructed a feed-forward neural network and compared its modeling ability to that of the model developed by the Ministry of Energy and Natural Resources (MENR). Using as input the annual gross national product (GNP), population, and vehicle density, they found the neural network resulted in a lower total minimum average error when compared to MENR predictions (Murat & Ceylan, 2006).

Predicting the final prices of online auction items was the focus of the 2006 work of Xuefeng, Lu, Lihua, and Zhao. This study collected auction data from a single auction site for all auctions of an identical item and based the predictive model on 11 seller attributes such as auction start and stop time, seller credibility (as reported by the auction web site), shipping and payments methods, and the final price. Five bidder attributes were also included in the model including bidder credibility, bid amount, item demand, and bid time. A three layer feed forward neural network was then constructed and trained using

the familiar back-propagation algorithm. In summary, the neural network demonstrated an average accuracy of 91.29% while their comparison method of logistic regression recorded an average accuracy of just 76.46% (Xuefeng, Lu, Lihua, & Zhao, 2006).

Information filtering and retrieval represents another area of applied neural networks. With the explosion of information on the Internet and in globally networked databases, as well as in the files on our personal computers and in the contents of our email messages, efficient means of search and retrieval have become paramount. Boger, Kuflik, Shoval, and Shapira applied a neural network approach to information filtering and retrieval in their 2000 work and found that their neural network approach outperformed traditional keyword filtering systems (Boger, Kuflik, Shoval, & Shapira, 2000).

Neural Networks in Education

In a 2004 study, Naik and Ragothaman explored a neural network's ability to predict the success of MBA students as part of the admissions process at a private midwestern university. In this work, variables typically used in the college admission screening process were identified such as overall undergraduate GPA, junior/senior GPA, undergraduate major and institution, and GMAT score.

These variables are typically evaluated with various statistical models such as discriminant analysis, multiple regression and stepwise regression to predict an applicant's success in an MBA program. Naik goes on to point out some of the shortcomings of these typical approaches such as the assumption that there is multivariate normality, the rather skewed distribution of graduate GPAs found in other research such

as Abedi in 1991, and that statistical models only use objective data disregarding potentially relevant subjective data.

The neural network topology for the Naik and Ragothaman study was a three-layer back-propagation network. Ten variables comprised the input layer with 1 neuron in the output layer. The size of the hidden layer was unspecified. The result (output) of the network was either a 1 (successful) or 0 (marginal). A successful MBA student was deemed to be one who achieved an overall GPA of 3.3. A marginal student would have achieved a GPA less then 3.3.

After training, which consisted of using historical data from admitted MBA students and their final GPA, the neural network was used to analyze 184 MBA applicants. The results of the neural network were then compared to the results of two common linear models, Logit and Probit.

The results showed that the neural network model correctly predicted 93.38% of the successful students and 80.90% of the marginal students for an overall accuracy of 89.13%. Comparatively speaking, Logit correctly predicted 86.78% of the successful students and 46.03% of the marginal students for an overall accuracy of 72.83% and the Probit model correctly classified 87.60% of the successful students and 46.03% of the marginal students for an overall accuracy of 73.37%.

These results would seem to indicate that the neural network model was able to make use of some subtleties in the relationships of the variables that defy more traditional linear regression. In either case, as is intended for this research, the authors suggest that their predictive tool be used as an additional factor in aiding decision making, not as the only factor.

Hoefer and Gould also compared neural networks to linear and non-linear regression in an attempt to forecast the success of MBA students. In their 2000 study they found the neural network to only be marginally better than the traditional methods. They did note, however, that the neural network model allowed them to include qualitative variables in the model such as gender, birth date, and students graduating from tier 1 schools (Hoefer & Gould, 2000).

In Jing Luan's 2002 paper presented at the Annual Forum for the Association of Institutional Research (Luan, 2002) he proposed the use of neural networks to predict the likelihood of student dropouts in higher education. Using the predictive capacity of neural networks allows the college to intervene prior to a dropout in an effort to enhance retention.

Gonzalez and DesJardins also apply the predictive capabilities of neural networks in their 2001 and 2002 papers, which studied the ability to predict what engineering school students would apply to. They then compared this to the traditional logistic regression modeling and discovered that neural networks proved an enhancement in making this sort of prediction (Gonzalez & DesJardins, 2002, 2001). The authors point out, along with a neural network's predictive ability, the additional benefit of not having to first culled the relationships between variables, as required in more traditional statistical analyses.

Finally, neural networks have also been applied in the field of education to forecast educational spending. In the 1999 work of Baker and Richards, three neural network architectures were used to predict the 1991-1995 per-pupil spending in U.S. public elementary and secondary schools. Their results were compared to the National

Center for Educational Statistics' multivariate regression model and found to range from comparable to superior (Baker & Richards, 1999).

Summary

This review of neural network applications reveals a common thread – in all cases, neural networks seem to be applied when the interrelationships among variables are either too numerous to account for or too complex to model well using traditional mathematical constructs. In many cases, where existing models are already in place, they seem to make assumptions that put boundaries around the solution space in order to permit the construction of a finite model. It is here that neural networks excel in allowing the model to be what it is and discerning the patterns and relationships; learning as it were, from pre-existing data.

CHAPTER 3

METHODOLOGY

Research Question

This study sought the answer to the question, "Can a neural network model of non-curricular variables provide greater accuracy in predicting student performance on standardized math and reading tests for high school sophomores when compared to standard multiple regression?" The non-curricular variables that were used came from the National Center for Educational Statistics Educational Longitudinal Study: 2002 encompassing 15,362 high school sophomores from 752 public, Catholic, and private schools.

Expected Result

Given a Neural Network's proven ability to accurately model both linear and non-linear systems, it was expected that the Neural Network would outperform the standard regression predictors in the prediction of both math and reading scores. This would seem plausible given that that multi-layer Neural Networks are able to model non-linear functions to an arbitrary degree of precision (Satinover, 2001). Further, Neural Networks have been shown to outperform linear predictors in a variety of applications (Mandic, 2001).

Model performance was compared via standard error of estimate (RMSE) and Wilmott's indices of agreement (d_1 and d_2). A second comparison was also performed with varying levels of noise introduced into the data. It was expected that the accuracy of

both models would degrade in this second case but that the neural network would degrade more gradually.

Classification of Independent Variables

The non-curricular data collected by the ELS: 2002 were organized into five categories. The five categories include 1) socio-demographics, 2) students' perceptions of school, 3) extracurricular and sports activities, 4) students' use of time outside of school, and 5) students' values, expectations and future plans.

Prior studies such as the National Longitudinal Study of the High School Class of 1972, the High School and Beyond Longitudinal Study, and the National Educational Longitudinal Study of 1988 have all pointed to a relationship between socio-demographics and student achievement (Ingels et al, 2005). Studies by Green et al (1995), Ladd and Birch (1997), and Osterman (2000) connected students' perceptions of their school and teachers to educational expectations and achievement test scores (Green, et all 1995; Ladd and Birch, 1997; Osterman, 2000).

The relationship between extracurricular activities and achievement, however, is less clear. As reported by Ingels et al, it is tempting to associate high achievement with participation in extracurricular activities, but this cannot be confirmed. This is partly because it is difficult to determine if participants in extracurricular activities perform better due to those activities or because they tend to be from higher socio-demographic status (Ingels et al, 2005). In the realm of how students spend their time outside of the classroom, Ingels et al report that findings vary with respect to affect on achievement.

Participation in extracurricular activities, homework outside of school, and reading for pleasure tend to be positively associated with achievement though (Ingels et al, 2005).

Ingels et al (2005) did not relate the final category, life values and student expectations, to achievement in anyway, but this category captures student perceptions of themselves and what they believe parents, teachers, and counselors expect of them. It is conceivable then, that factors in this category may impact student achievement as the students' perceptions in this area likely have direct influence on student motivation (e.g., Pintrich & De Groot, 1990, Bandura 1986, Bandura 1997, Hammouri 2004, Pajares & Graham 1999).

Tables 3-1 through 3-5 outline the variables to be studied in each of the five categories. The name(s) in parentheses is the variable name from the ELS: 2002 data set. These variables represent the independent variables in the model. For a full inventory of independent variables, including survey response options, please refer to Appendix A.

Table 3-1

Socio-Demographic Variables

Variable	Description
Family composition (BYFCOMP)	A nominal measure of the family configuration such as (1) mother and father and (2) mother and male guardian.
Father's education (FATHED)	A nominal measure of the father's highest level of education such as (1) Did not finish high school and (2) Graduated from high school.
Mother's education (MOTHED)	A nominal measure of the mother's highest level of education similar to FATHED.
Parent's education (PARED)	A nominal measure similar to FATHED and MOTHED reporting the highest level of education attained by either parent.
Socioeconomic status (SES1QU)	An ordinal measure of the student's socioeconomic status as classified in one of four quartiles.

Table 3-2

Student Perceptions of School

Variable	Description
Region (BYREGION)	A nominal measure of the geographic region in which the school is located such as Northeast or South.
Type (BYSCTRL)	A nominal measure of the type of school such as public, Catholic, or Private.
Location (BYURBAN)	A nominal measure of the metropolitan status of the school such as Urban, Suburban, or Rural.
Cutting class (BYS24B)	An ordinal measure in five ranks of how many times the student skipped class.
No books/homework (BYS38B)	An ordinal measure in four ranks of how often the student came to class without texts or completed homework.
High school program (BYS26)	A nominal measure of the student's self-reported high school program such as General, College Prep (Academic), or Vocational (including technical or business).
Crime and bullying (BYS22A, BYS22B, BYS22C, BYS22D, BYS22E, BYS22F, BYS22G, BYS22H)	These ordinal measures indicate how often a student experienced various types of school crime or bullying. The ranks include never, once or twice, and more than twice.
Importance of good grades (BYS37)	An ordinal measure in four ranks of the importance of grades to the student.
Likes school (BYS28)	An ordinal measure in three ranks of how much the student likes school.
Reasons for going to school (BYS27A, BYS27B, BYS27C, BYS27D, BYS27E, BYS27F, BYS27G, BYS27H, BYS27I)	Ordinal measures in four ranks of various reasons the student attends school.
School Rules (BYS21A, BYS21B, BYS21C, BYS21D, BYS21E)	Ordinal measures in four ranks of how much students agreed or disagreed with various school rules.
School safety (BYS20J, BYS20M, BYS20N)	Ordinal measures in four ranks of how much students agree or disagree with various statements about school safety.
School and teachers (BYS20A, BYS20B, BYS20C, BYS20E, BYS20F, BYS20G)	Ordinal measures in four ranks of how much students agreed or disagreed with various statements about their teachers and their school.

Table 3-3

Extracurricular and Sports Activities

Variable	Description
School activities (BYS41A, BYS41B, BYS41C, BYS41D, BYS41E, BYS41F, BYS41G, BYS41H, BYS41I)	Indicates which school sponsored activities, if any, in which the student participated.
Intramural (BYS39A-BYS39H)	Indicates which intramural sports, if any, in which the student participated.
Interscholastic sports (BYBASEBL, BYSOFTBL, BYFOOTBL, BYSOCCER, BYTEAMSP, BYSOLOSP, BYBSKTBL)	Indicates which interscholastic sports, if any, in which the student participated.
Work (BYS72)	A nominal measure of the students work history answering the question "have you ever worked for pay?" with the responses No, Yes, and I am currently employed, and Yes, but I am currently not employed.

Table 3-4

Student Use of Time

Variable	Description
Computer use (for schoolwork BYS46A/other than schoolwork BYS46B)	A measure of the number of hours spent by the student using computers for schoolwork and purposes besides schoolwork.
Computer use for various purposes (BYS45A, BYS45B, BYS45C)	Ordinal measures of how much the student used computers, in any location, for various purposes. The choices are Never, Rarely, Less than once a week, Once or twice a week, Almost Every Day, or Every Day.
Extracurricular activities (BYS42)	A measure of the number of hours the student spent on school-sponsored extracurricular activities.
Math homework (in school BYS35A/out of school BYS35B)	Measures of the number of hours the student spent on math homework.
English homework (in school BYS36A/out of school BYS36B)	Measures of the number of hours the student spent on English homework.
Total homework (in school BYS34A/out of school BYS34B)	Measures of the number of hours the student spent on homework per week in all subjects.
Outside reading (BYS43)	A measure of the number of hours the student spent reading material not assigned by school.
Working for pay (BYS75)	A measure of the number of hours per week a student currently works or has worked in the past if they are currently unemployed.

Table 3-5

Student Values Expectations and Future Plans

Variable	Description
Educational expectations (STEXPECT)	A nominal measure of the student's expected highest level of academic achievement such as Less than high school, high school, 2 year community college or vocational school, and so on.
Education past high school (BYS57)	A nominal measure of the student's expectation of continuing their education past high school, if they reported they thought they would complete high school. Values are Yes, right after high school, Yes, after staying out of school for 1 year, Yes, but I don't know when, No, I don't plan to continue my education after high school and so on.
Participate in college sports (BYS60)	An indicator of whether or not students who indicated they planned to continue their education planned to participate in college sports (not intramural).
Athletic scholarship (BYS61)	For students planning to continue to their education and planning to participate in college sports, they were asked to indicate if they hoped to receive an athletic scholarship.
Life values (BYS54A-L, BYS54N, BYS54O)	Nominal measures from Not Important to Very Important, of the student's perception of the importance of a series of life values related to work and education, family and friends, and the community.
Right after high school (BYS66A, BYS66B, BYS66F)	Student's perceptions of what they think is the most important thing to do right after high school from the point of view of their parents, school counselor, and favorite teacher. The possible choices were Get a full-time job, Enter a trade school, Enter the military, Get married, Whatever the student wants to do, or Don't know.

Dependent Variables

The dependent variables for the model were the standardized math and reading scores. For the purposes of the ELS: 2002 study, the math and reading achievement scores were standardized to a mean of 50 and a standard deviation of 10.

Table 3-6

Dependent Variables

Variable	Description
Standardized math (BYTXMSTD)	Standardized math achievement score.
Standardized reading (BYTXRSTD)	Standardized reading achievement score.

Procedures

Sample Selection

The population being studied consisted of 15,362 high school sophomores from 752 public, Catholic, and private schools. Since all the data had been collected and is in electronic form, the entire population was theoretically available for study. For the purposes of this research, however, a representative sample of 10% of the population was selected and used to A) train the neural network model and B) develop the regression prediction equations. In determining the sample size a power table was consulted to ensure that comparisons would be statistically meaningful. In order to detect small to medium effect sizes at power = .80 and alpha = .05, this study requires 800 (small effect) and 85 (medium effect) samples respectively (Cohen, 1977). Since this study used

approximately 10% of the total population, or 1,534 samples, there was more than enough data to provide a statistically meaningful analysis. Another 10% of the population (distinct from the first 10%) was also selected to A) test the accuracy of the neural network model and B) test the accuracy of the regression prediction equations.

Prior to case selection, the data were cleaned so that any cases containing missing or invalid responses were removed. Following the cleaning process 3,068 cases remained. Since the original intent was to use a sample size of 10%, or 1,536 cases, for training and test purposes, the remaining 3,068 cases were split into two groups of 1,534 cases each.

Case selection was performed with an algorithm developed specifically for this study that would ensure an even distribution of survey responses in each of the two 10% samples. Initially, the data were dummy coded so that they would be in a format suitable for the regression analysis and the scores were scaled to the range [0.0, 1.0] by simply dividing by 100. Scaling the scores into this range facilitated training the neural network since the outputs of the neural network are limited to the range [0.0, 1.0] as a consequence of using a sigmoid transfer function in the output layer.

The case selection process then built a map for each dummy variable of those cases for which each variable had a value of 1. Case selection then proceeded by randomly selecting a dummy variable, without replacement, and then randomly selecting one of the cases for which this variable had the value 1. Once this process had cycled through all of the dummy variables, the pool of dummy variables was recycled and the process repeated until 1,534 cases had been selected. The overall algorithm is shown in Figure 3-1.

The selected 1,534 cases were then written to a file to be used for training the neural network and developing the regression equation. The remaining 1,534 cases were written to another file to be used to test the performance of the neural network and regression equations. Four additional files of test data were also produced at this time, one each with 10, 15, 20, and 25 noisy variables introduced. The purpose of these additional test files was to measure how each model degraded as the quality of the data degraded. To introduce noise into the data, 10 variables were selected at random and marked as missing. For each subsequent file, another 5 variables were selected at random and also marked as missing thus allowing each noisy test file to carry forward the noise from the previous file. Then an additional 5 noisy (missing) variables were added.

Following the case selection process, the 10% training and test samples were compared to the total population of cases to ensure that the distribution of standardized math and reading scores was similar. The math and reading scores in the ELS: 2002 data were standardized to a mean of 50 and a standard deviation of 10. Histograms for both the math and reading scores for the total population and the 10% samples show that the case selection process had not skewed the distribution of scores. The histograms are shown in Figures 3-2 through 3-7.

A program written for this study performed the case selection process. The source code for the software is shown in appendix B.

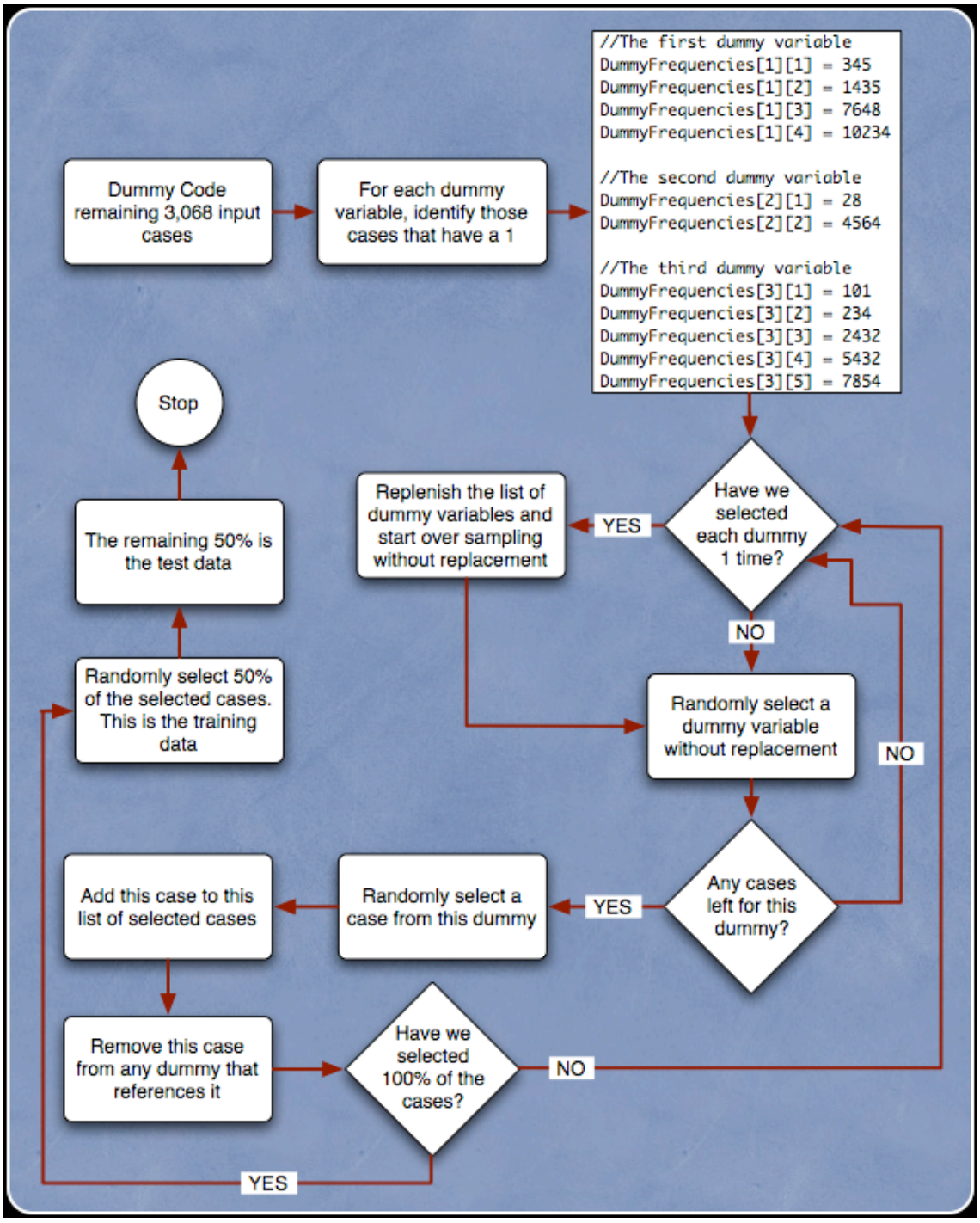


Figure 3-1. Case selection method

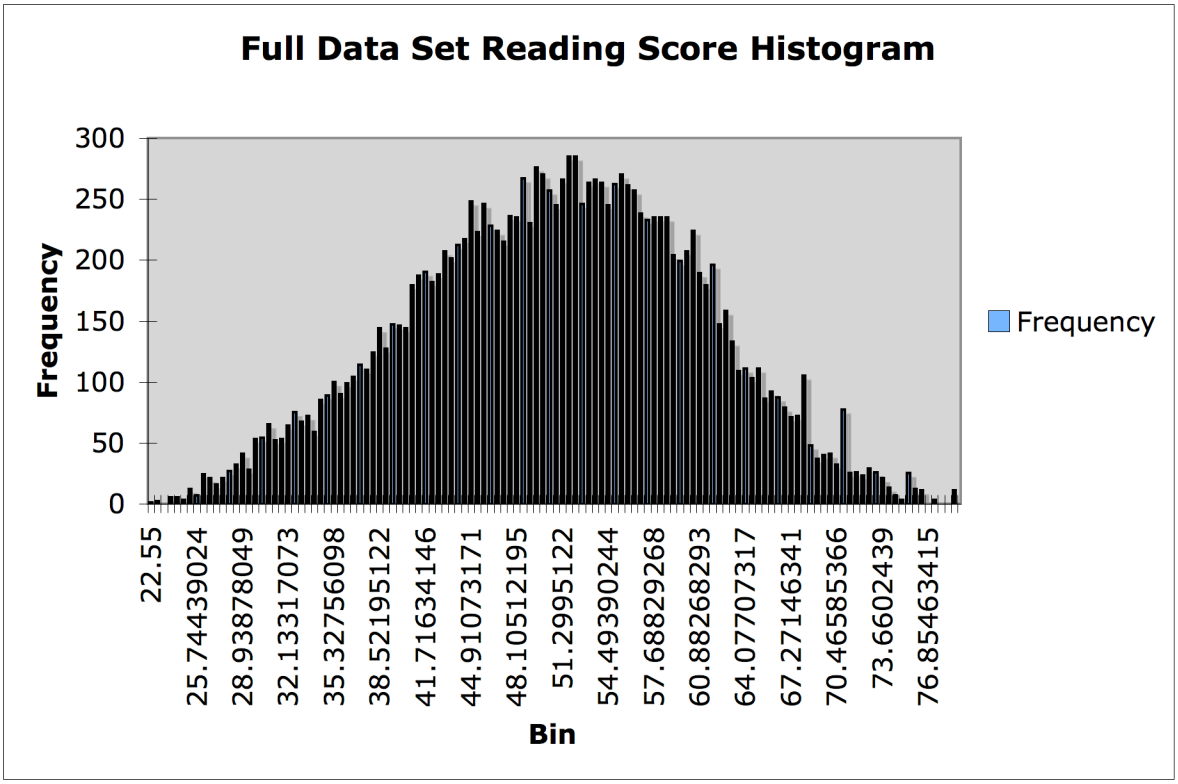


Figure 3-2. Histogram of reading scores for the total population

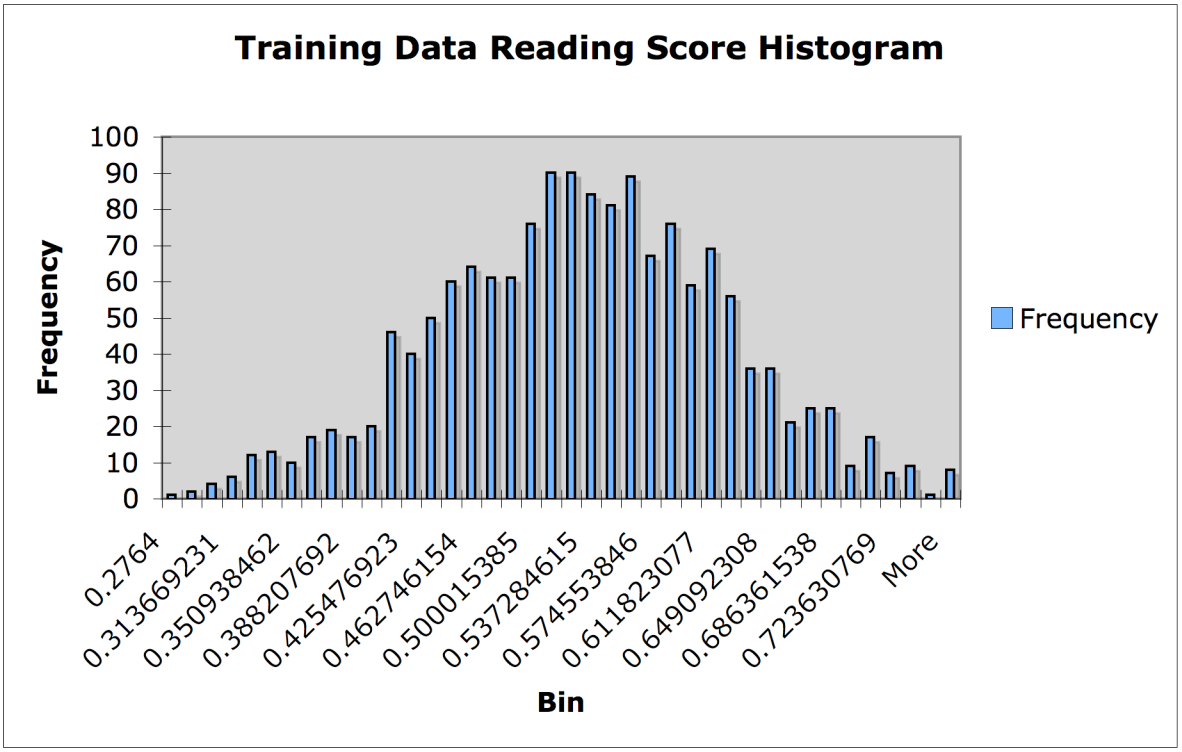


Figure 3-3. Histogram of reading scores for the training sample

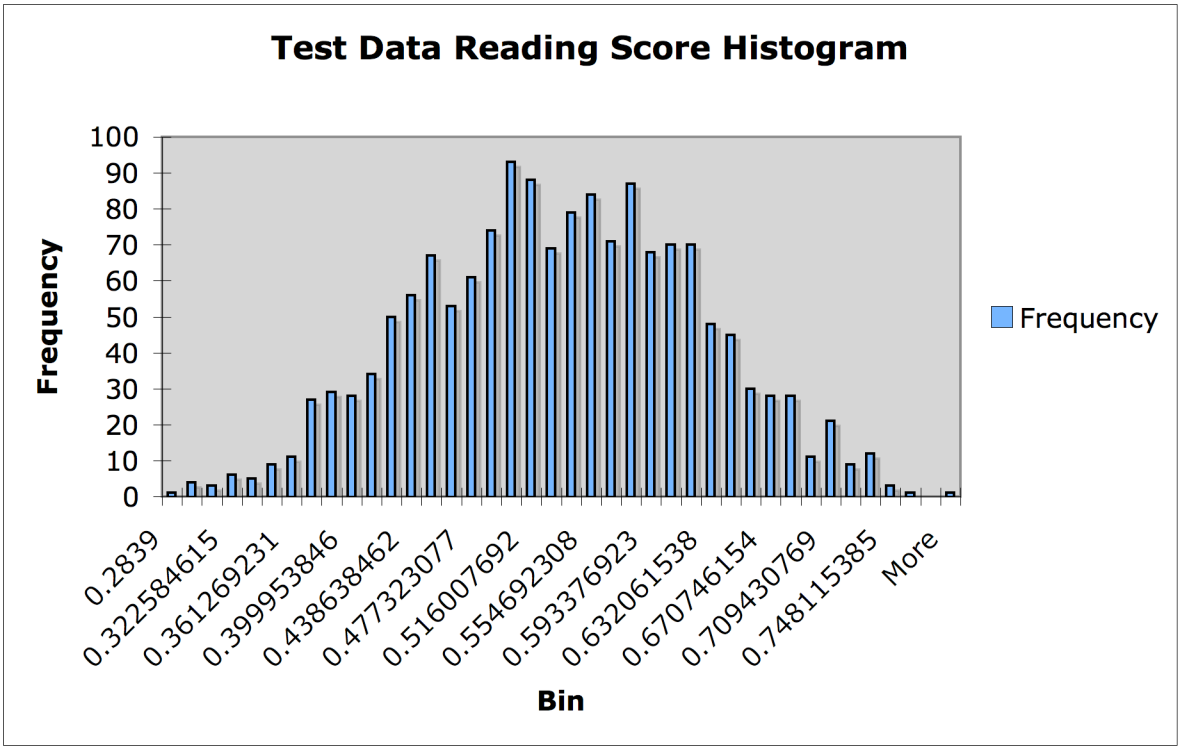


Figure 3-4. Histogram of reading scores for the test sample

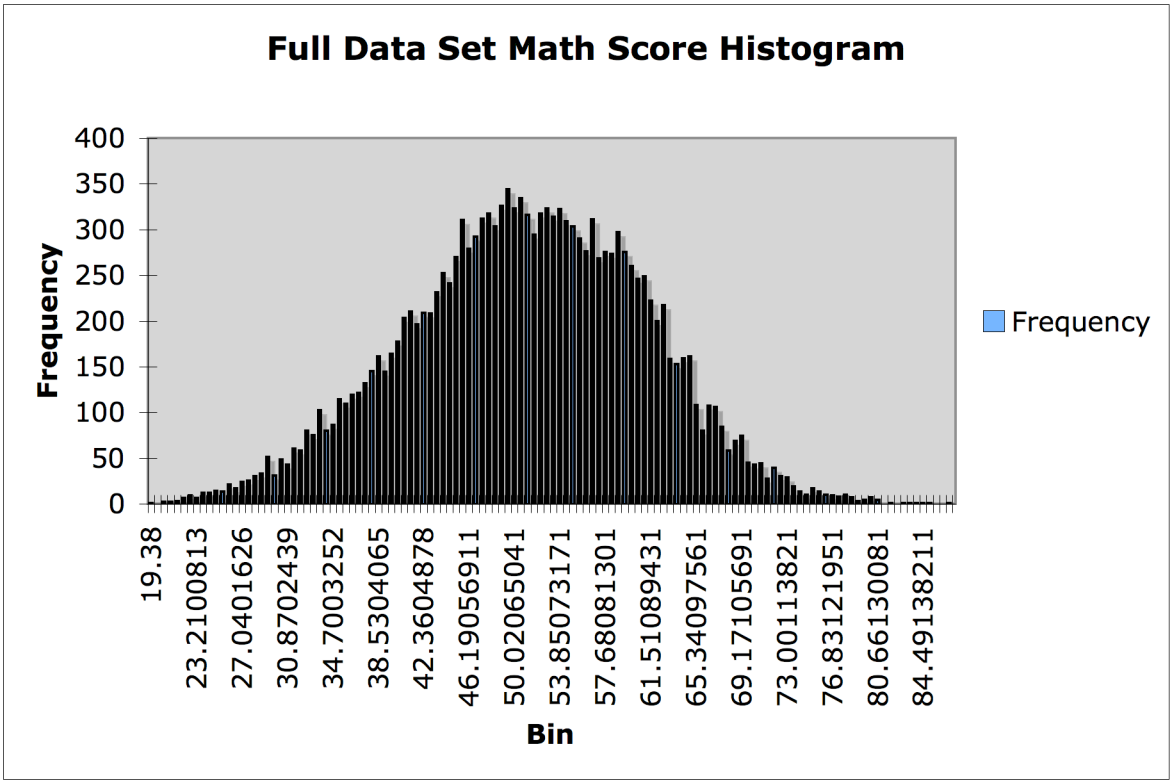


Figure 3-5. Histogram of math scores for the total population

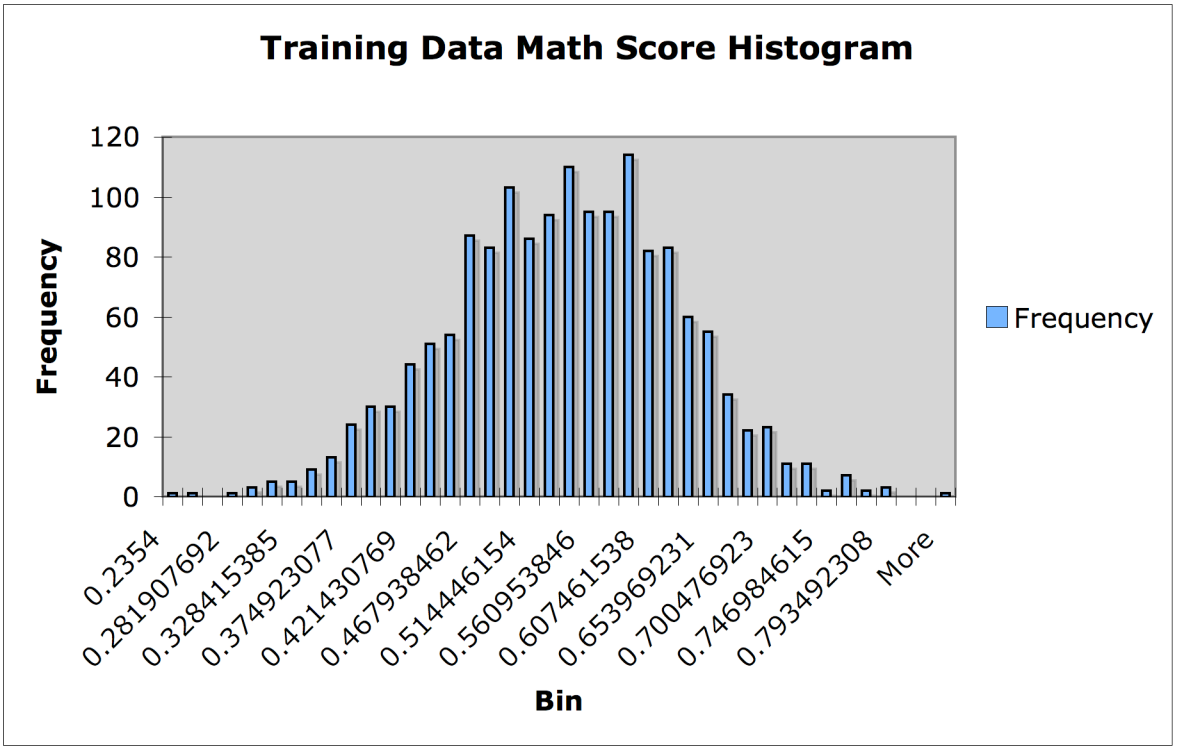


Figure 3-6. Histogram of math scores for the training sample

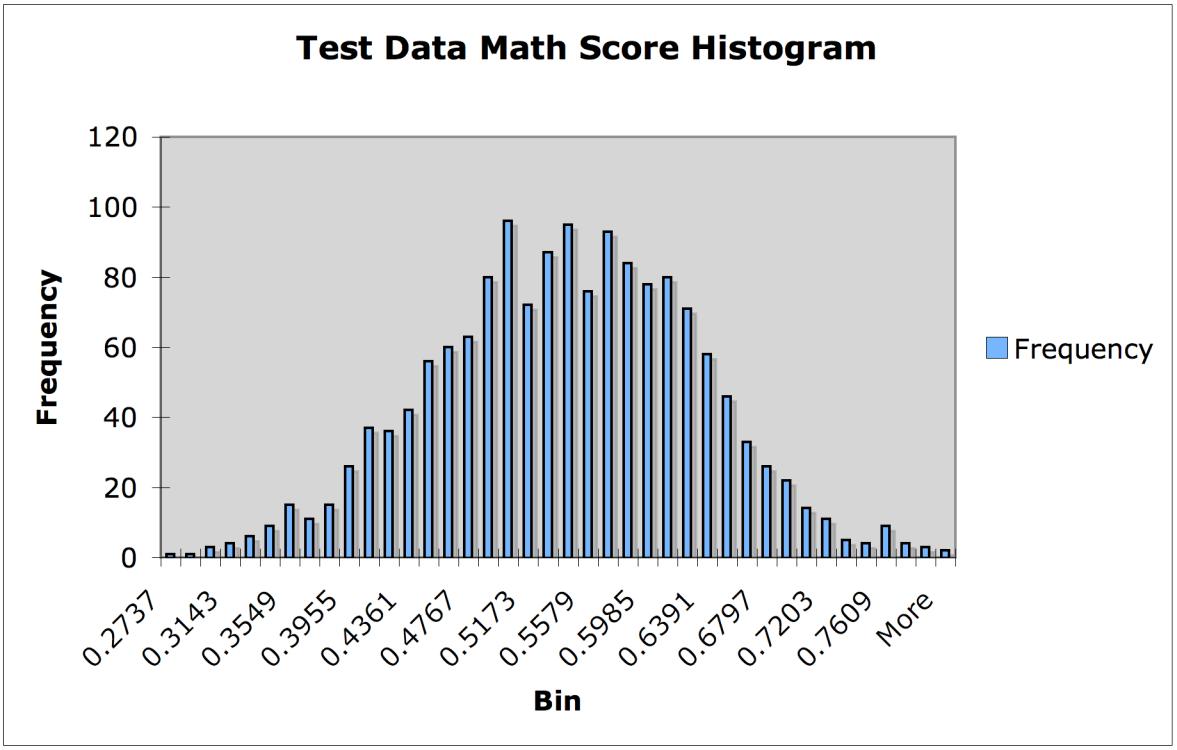


Figure 3-7. Histogram of math scores for the test sample

Instruments

A three-layer back propagation neural network and regression prediction equations were developed in an effort to forecast student math and reading scores based on the non-curricular independent variables. In the case of the regression equations, dummy variables were created for all nominal and ordinal variables, and separate regression equations were developed to predict math and reading scores independently.

The regression equation was developed by importing the dummy coded training data, which was produced by the case selection process, into SPSS and performing a linear regression with 103 independent variables. This regression was performed twice:

once for math scores and once for reading scores. The resulting regression coefficients were then exported from the SPSS output viewer to an excel spreadsheet as shown in Figure 3-8.

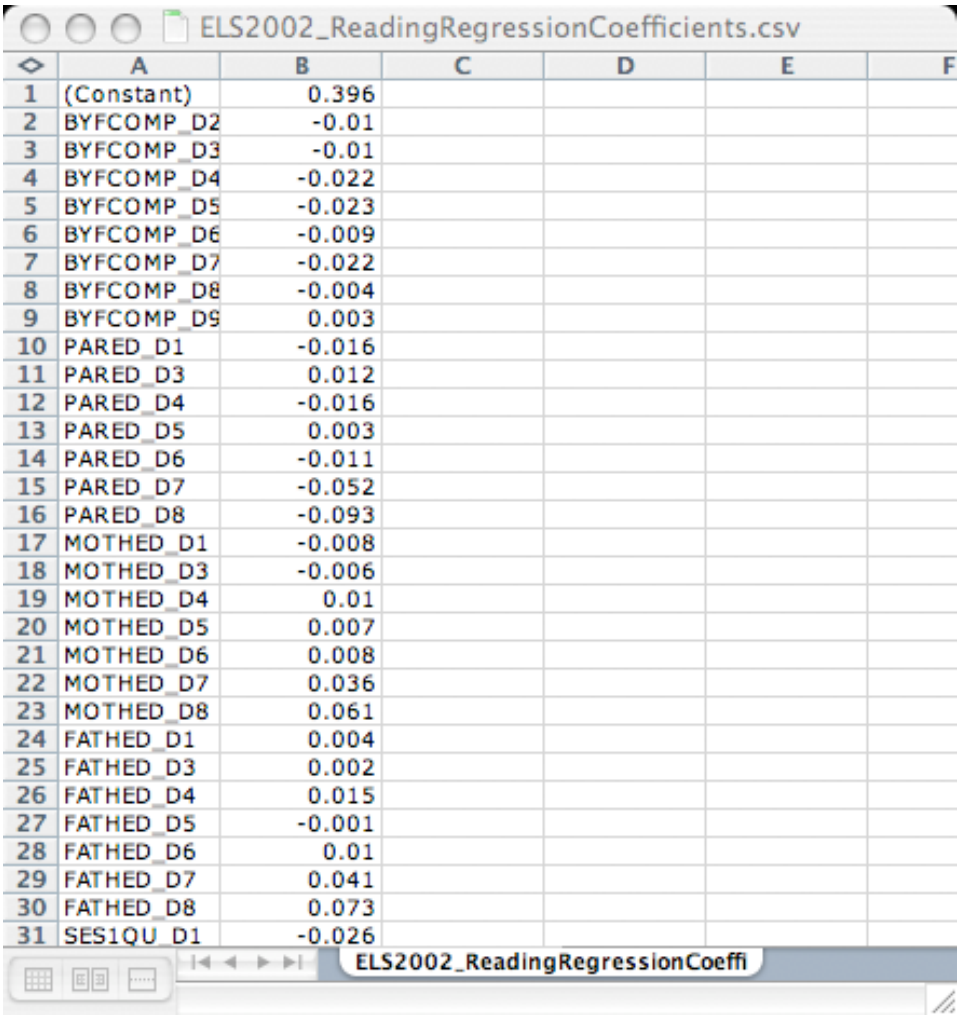
ELS2002_ReadingRegressionCoefficients.xls

	A	B	C	D	E	F	G	H
1	Coefficients(a)							
2			Unstandardized Coefficients		Standardized Coefficients			
3	Model		B	Std. Error	Beta	t	Sig.	
4	1	(Constant)	0.396	0.047		8.427	0.000	
5		BYFCOMP_D2	-0.010	0.007	-0.033	-1.432	0.153	
6		BYFCOMP_D3	-0.010	0.012	-0.018	-0.800	0.424	
7		BYFCOMP_D4	-0.022	0.016	-0.031	-1.370	0.171	
8		BYFCOMP_D5	-0.023	0.006	-0.088	-3.592	0.000	
9		BYFCOMP_D6	-0.009	0.013	-0.016	-0.714	0.475	
10		BYFCOMP_D7	-0.022	0.022	-0.022	-1.022	0.307	
11		BYFCOMP_D8	-0.004	0.034	-0.002	-0.108	0.914	
12		BYFCOMP_D9	0.003	0.019	0.003	0.154	0.878	
13		PARED_D1	-0.016	0.016	-0.039	-1.004	0.315	
14		PARED_D3	0.012	0.013	0.039	0.955	0.340	
15		PARED_D4	-0.016	0.014	-0.050	-1.145	0.253	
16		PARED_D5	0.003	0.015	0.010	0.196	0.845	
17		PARED_D6	-0.011	0.015	-0.045	-0.720	0.472	
18		PARED_D7	-0.052	0.022	-0.164	-2.368	0.018	
19		PARED_D8	-0.093	0.036	-0.241	-2.618	0.009	
20		MOTHED_D1	-0.008	0.010	-0.026	-0.775	0.439	
21		MOTHED_D3	-0.006	0.010	-0.019	-0.571	0.568	
22		MOTHED_D4	0.010	0.011	0.032	0.910	0.363	
23		MOTHED_D5	0.007	0.011	0.022	0.633	0.527	
24		MOTHED_D6	0.008	0.011	0.029	0.673	0.501	
25		MOTHED_D7	0.036	0.017	0.088	2.186	0.029	
26		MOTHED_D8	0.061	0.028	0.093	2.154	0.031	
27		FATHED_D1	0.004	0.009	0.013	0.391	0.696	
28		FATHED_D3	0.002	0.010	0.007	0.226	0.821	
29		FATHED_D4	0.015	0.012	0.039	1.232	0.218	
30		FATHED_D5	-0.001	0.012	-0.003	-0.081	0.935	
31		FATHED_D6	0.010	0.013	0.038	0.817	0.414	
32		FATHED_D7	0.041	0.018	0.112	2.196	0.028	
33		FATHED_D8	0.073	0.033	0.168	2.237	0.025	
34		SES1QU_D1	-0.026	0.012	-0.111	-2.186	0.029	

Sheet

Figure 3-8. Regression coefficients exported as an Excel spreadsheet

The spreadsheet was then reduced to two columns of data representing the variable names and the regression coefficients as shown in Figure 3-9.



ELS2002_ReadingRegressionCoefficients.csv

	A	B	C	D	E	F
1	(Constant)	0.396				
2	BYFCOMP_D2	-0.01				
3	BYFCOMP_D3	-0.01				
4	BYFCOMP_D4	-0.022				
5	BYFCOMP_D5	-0.023				
6	BYFCOMP_D6	-0.009				
7	BYFCOMP_D7	-0.022				
8	BYFCOMP_D8	-0.004				
9	BYFCOMP_D9	0.003				
10	PARED_D1	-0.016				
11	PARED_D3	0.012				
12	PARED_D4	-0.016				
13	PARED_D5	0.003				
14	PARED_D6	-0.011				
15	PARED_D7	-0.052				
16	PARED_D8	-0.093				
17	MOTHED_D1	-0.008				
18	MOTHED_D3	-0.006				
19	MOTHED_D4	0.01				
20	MOTHED_D5	0.007				
21	MOTHED_D6	0.008				
22	MOTHED_D7	0.036				
23	MOTHED_D8	0.061				
24	FATHED_D1	0.004				
25	FATHED_D3	0.002				
26	FATHED_D4	0.015				
27	FATHED_D5	-0.001				
28	FATHED_D6	0.01				
29	FATHED_D7	0.041				
30	FATHED_D8	0.073				
31	SES1QU_D1	-0.026				

ELS2002_ReadingRegressionCoeffi

Figure 3-9. Variable names and regression coefficients

This file was then saved as a comma-separated-value file, which was read by the cross-validation software written for this study. The cross-validation program first read the variable names and coefficients and then a file of test data. Next, it executed the regression equation for each case in the test file and, when complete, computed the RMSE, d_1, and d_2 statistics. See the analysis section in this chapter for further discussion

of the comparison statistics. The source code the cross-validation program is shown in appendix C.

The neural network model employed by this study was a three-layer, feed-forward neural network trained via resilient back-propagation. The input layer consisted of 103 neurons, one for each feature (independent variable) identified in the ELS: 2002 data set.

The input neurons were fully connected to the hidden layer. This means that each neuron in the input layer was connected to every neuron in the hidden layer. The hidden layer consisted of 30 neurons. The sizing of the hidden layer has no hard and fast rules and, generally speaking, is usually smaller than the input layer but larger than the output layer. Often, for a given application, several different hidden layer sizes are tried in an attempt to balance network accuracy and training time since the more neurons there are in the hidden layer, the more computationally expensive the network will be to train.

The output layer consisted of two neurons, one each to provide the network's prediction of the student's math and reading scores. Each neuron in the hidden layer was fully connected to the neurons in the output layer.

Figure 3-10 displays the neural network model. Note that connections are only shown from the first neuron in the input layer to the hidden layer and from the neurons in the hidden layer to the first output layer neuron. The input layer neurons have been grouped in the figure by the five classifications of data present in the ELS: 2002 for illustrative purposes but otherwise have no impact on the topology or functioning of the network.

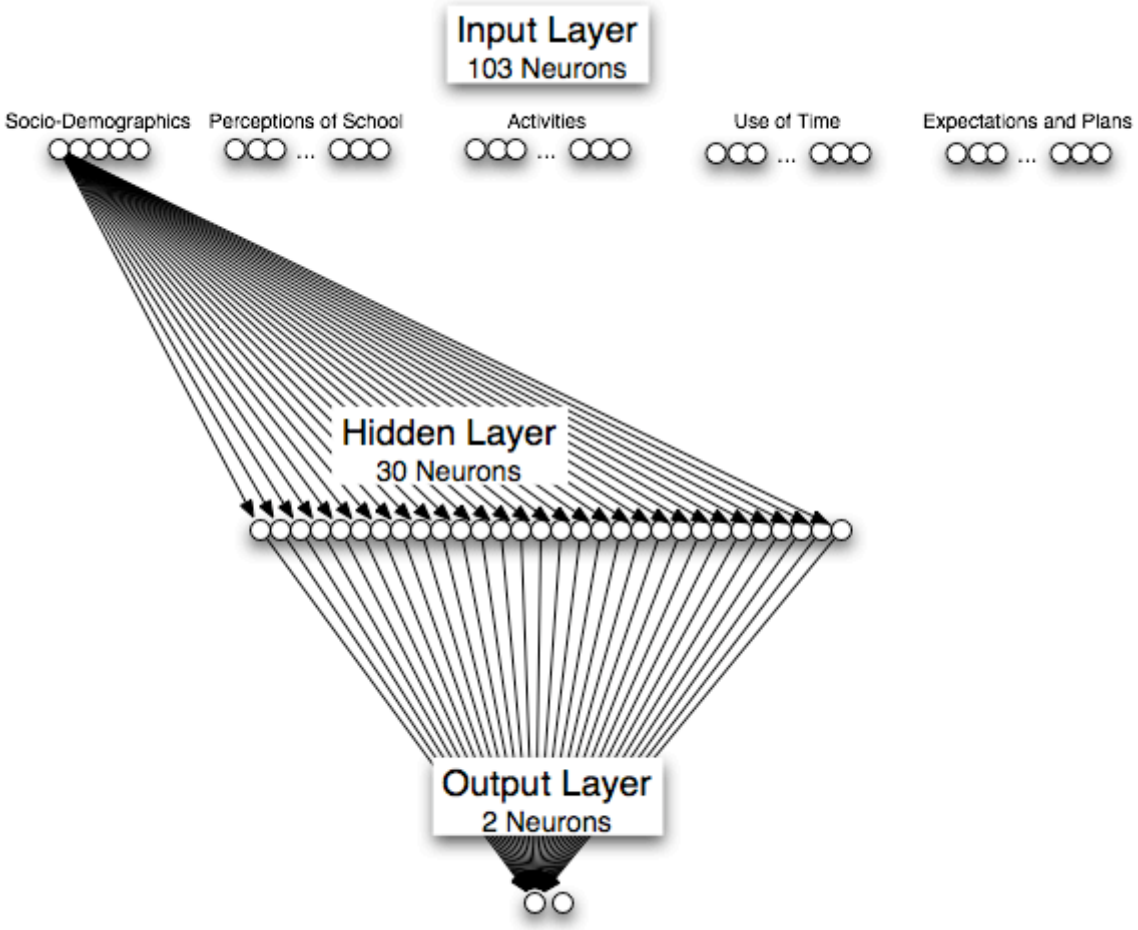


Figure 3-10. Neural network model

Training the neural network consisted of presenting the file of training cases to the neural network, computing the RMSE, and propagating the error back through the synaptic weights to minimize the network's error. Each pass through all of the test cases represented one training epoch, and the network was trained for a total of 500 epochs.

To present each case to the neural network, the input values from the training data were encoded for the 103 input neurons. Encoding the input variables consisted of scaling each input value to the range [0.2, 0.8] based on the position of the input value in the list

of possible input values. Missing values were encoded as 0. For example, the variable STEXPECT had the following list of possible values of -1, 1, 2, 3, 4, 5, 6, and 7. If the response for a given case was 4, then the encoded value for the neural network would be 0.2 + (4/7)*0.6 = 0.54286. Scaling the input values in this way helps to avoid swamping the hidden layer neurons, given that each hidden layer neuron has 103 incoming connections.

One risk, however, of extensive training of a neural network is over-fitting. Over-fitting occurs when the neural network so tightly fits the training data that it doesn't generalize well to the overall population. To avoid this problem, 10%, or 153 cases, of the training data were set aside. The network was then trained using the remaining 1,381 training cases. After every 10^{th} epoch, the neural network was shown the 153 reserved training cases and the RMSE computed. If this RMSE was the best one so far for the 153 training cases, the state of the neural network was set aside and the training continued. When 500 training epochs had completed the final state of the network, the network with the best RMSE for the 153 reserved training cases, was saved.

Once the network had been trained, cross-validation was performed in two steps: first by setting the network's training property to false; second by presenting the network with the test data files produced during the case selection process. In this mode the neural network would read the test cases, run each case through the network to compute the predicted math and reading scores, and then compute the RMSE, d_1, and d_2 statistics once all cases had been processed. See the analysis section for further discussion of the comparison statistics. The source code for the neural network program is shown in appendices D and E.

Data Collection

Data for this study was borrowed from the National Center for Educational Statistic from the Education Longitudinal Study: 2002. The ELS: 2002 collected data from school administrators, teachers, parents, and students. For this research, however, the primary data of interest comes from the student assessment (standardized math and reading scores) and the student survey, with just a few pieces of data from the parent survey.

The ELS: 2002 was administered in a group setting in each school. The items on the questionnaire were partly based on past performance as well as continuing relevance of items from prior longitudinal studies. The questionnaire was field tested in 2001. This field-testing investigated response rates, reliability and factor structure, differential item functioning, reliabilities of scales, and inter-item consistency (Ingels et al, 2005).

Analysis

Two measures of prediction accuracy were computed for the neural network model and the regression equations: 1) standard error or estimate, also known as root mean square error (RMSE), and 2) Wilmott's indices of agreement, d_1 and d_2. Standard error of estimate is the square root of the average of the total squared error between the predicted and actual values. This measure is often used in model comparison but suffers from sensitivity to outliers. The general form of this measure is

$$RMSE = ([\Sigma(\hat{Y}_i - Y_i)^2]/n)^{1/2}$$

where n is the total number of cases. The smaller the value of RMSE the more closely the model predicts the actual values.

Wilmott's indices of agreement, d_1 and d_2, offer another indicator of model fit that measures the degree to which a model's predictions are correct. It does this by showing the degree to which the model's predictions vary about the mean as compared to the actual observations' variance around the mean. d_1 is the more conservative measure, using simple differences, while d_2 uses square differences (Comrie, 1997). For Wilmott's indices of agreement, the closer the measure is to 1, the more accurately the model fits the data. The general forms of d_1 and d_2 are:

$$d_1 = 1 - \frac{\left[\sum_{i=1}^{n}\left(|\hat{Y}_i - Y_i|\right)\right]}{\left[\sum_{i=1}^{n}\left(|\hat{Y}_i - \overline{Y}|\,|Y_i - \overline{Y}|\right)\right]}$$

$$d_2 = 1 - \frac{\left[\sum_{i=1}^{n}\left(|\hat{Y}_i - Y_i|\right)^2\right]}{\left[\sum_{i=1}^{n}\left(|\hat{Y}_i - \overline{Y}|\,|Y_i - \overline{Y}|\right)^2\right]}$$

To compare the performance of the neural network and the regression equations, RMSE and d_1 and d_2 were calculated for the neural network's predicted math and reading scores, as well as for the regression equations' predicted math and reading scores. Once this comparison was completed, the neural network and regression equations were run with the same test data but with ever increasing levels of noise introduced into the data.

CHAPTER 4

FINDINGS AND DATA ANALYSIS

Introduction

The goal of this study was to compare the performance of a feed-forward neural network to multiple linear regression to determine if either could provide more accurate modeling of student performance on math and reading achievement tests based on non-curricular variables. Two separate samples of 1,534 cases each were pulled from the survey data to be used for training and cross-validation.

Once the regression equations had been developed and the neural network trained, performance comparisons were made, first with the training data, to see which model provided a better fit to the training population, and then with the cross-validation data to see which model provided a better fit for cases that had never been seen before. Additional comparisons were also made using noisy versions of the cross-validation data to see if either model was more resistant to imperfect data. The following tables and figures show the results of those comparisons and, in the case of differences, where those differences are statistically significant.

Model Comparison of Training Data

Following the development of the regression equations and the training of the neural network, the comparison statistics RMSE, d_1, and d_2 were computed for the training data, that is, the data used to develop the regression equation as well as to train the neural network. For RMSE measures, smaller values indicate better fit with zero

being a perfect fit. For the d_1 and d_2 measures, larger values indicate a better fit with a value of 1.0 indicating a perfect fit. For all three measures of fit, the regression equation modeled the data more accurately as shown in Table 4-1 and Figure 4-1.

Table 4-1

Comparison of Multiple Linear Regression and Neural Network Models with Training Data

Model	Math RMSE	Math d_1	Math d_2	Reading RMSE	Reading d_1	Reading d_2
Multiple Linear Regression	0.05856	0.64783	0.85112	0.05852	0.64784	0.85406
Neural Network	0.06557	0.56425	0.78083	0.06644	0.55685	0.77714

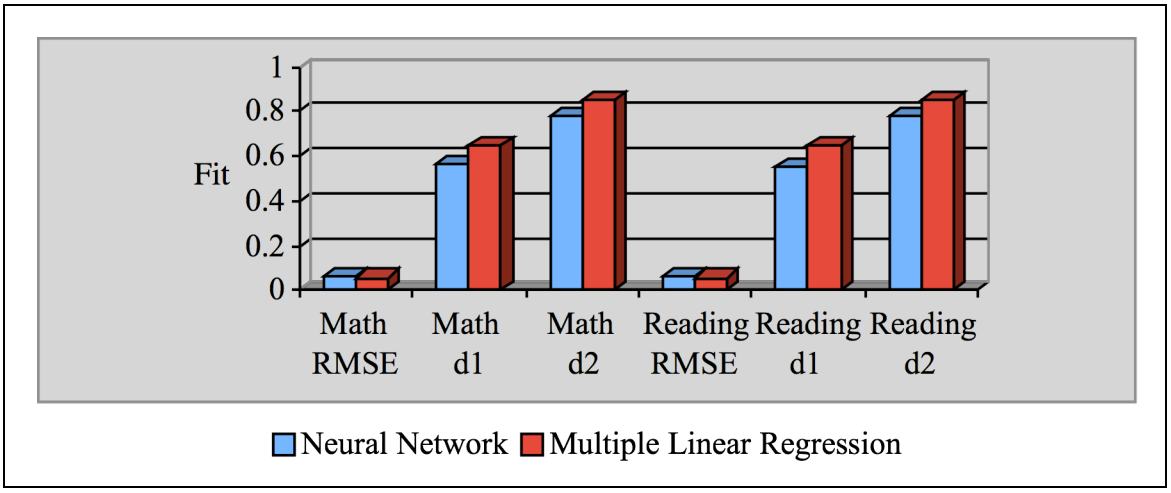


Figure 4-1. Comparison of models with training data

This is not necessarily a surprising result as the regression equation is the theoretical best possible fit through the training data while the neural network was trained to avoid over-fitting. By avoiding over-fitting, the final neural network chosen was not the one that necessarily fit the training data as tightly as possible but the one that was the best fit for a small reserve of the training data during training as described in Chapter 3.

Model Comparison of the Cross-Validation Data

When the models were executed with the cross-validation data, the second 10% sample, the results showed that multiple linear regression and the neural network were virtually indistinguishable. The neural network had slightly lower RMSE values for both math and reading score prediction while the d_1 and d_2 statistics were nearly identical as shown in Table 4-2 and Figure 4-2.

Table 4-2

Comparison of Multiple Linear Regression and Neural Network Models with Cross-Validation Data

Model	Math RMSE	Math d_1	Math d_2	Reading RMSE	Reading d_1	Reading d_2
Multiple Linear Regression	0.08129	0.51019	0.69804	0.08218	0.50927	0.70343
Neural Network	0.07351	0.50250	0.71164	0.07535	0.50066	0.70230

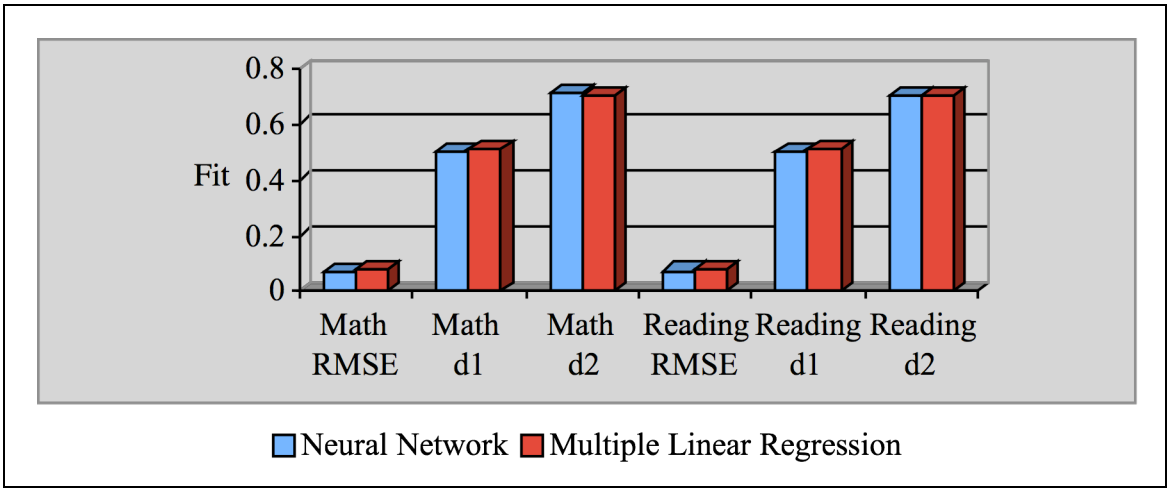


Figure 4-2. Comparison of models with cross-validation data

Model Comparison of the Cross-Validation Data with Synthetic Noise

The final comparison involved the performance of each model when faced with data involving progressively larger numbers of missing input values. During the case selection process, following the selection of the 10% training and cross-validation samples, four additional versions of the cross-validation data were produced with 10, 15, 20, and 25 missing variables respectively. Each model was then run using these additional data sets.

For reading score prediction, the neural network performed slightly better across all three measures of fitness. See Tables 4-3, 4-4, and 4-5 and Figures 4-3, 4-4, and 4-5. The differences, however, were not statistically significant at an alpha level of .05 between any of the three measures of fitness.

Table 4-3

RMSE Comparison of Multiple Linear Regression and Neural Network Models for Reading Score Prediction with Synthetic Noise

Model	0 Mising Inputs	10 Missing Inputs	15 Mising Inputs	20 Missing Inputs	25 Missing Inputs
Multiple Linear Regression RMSE	0.08128	0.08229	0.09484	0.09777	0.11939
Neural Network RMSE	0.07535	0.08497	0.08774	0.08769	0.08278

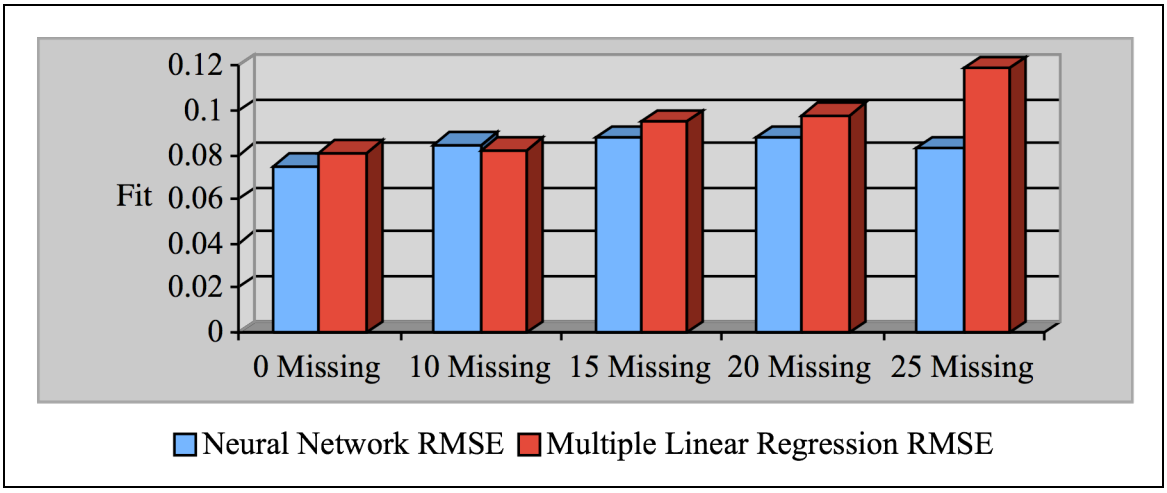


Figure 4-3. RMSE comparison of reading score prediction

Table 4-4

d_1 Comparison of Multiple Linear Regression and Neural Network Models for Reading Score Prediction with Synthetic Noise

Model	0 Mising Inputs	10 Missing Inputs	15 Mising Inputs	20 Missing Inputs	25 Missing Inputs
Multiple Linear Regression d_1	0.50927	0.47752	0.44784	0.43519	0.39055
Neural Network d_1	0.50066	0.47722	0.47633	0.47829	0.48096

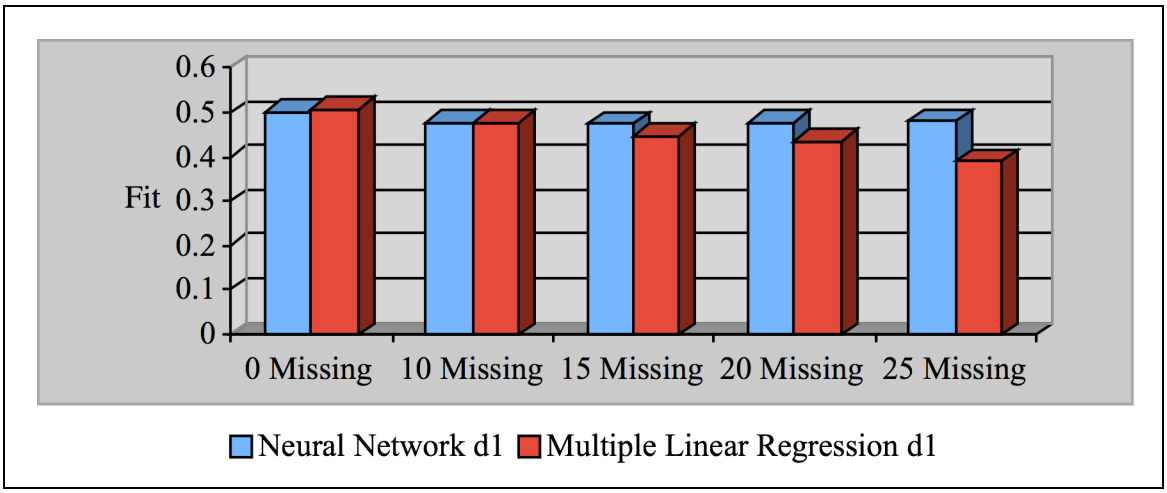


Figure 4-4. d_1 comparison of reading score prediction

Table 4-5

d_2 Comparison of Multiple Linear Regression and Neural Network Models for Reading Score Prediction with Synthetic Noise

Model	0 Mising Inputs	10 Missing Inputs	15 Mising Inputs	20 Missing Inputs	25 Missing Inputs
Multiple Linear Regression d_2	0.70343	0.6686	0.62404	0.60734	0.54806
Neural Network d_2	0.70320	0.66274	0.66079	0.66292	0.66745

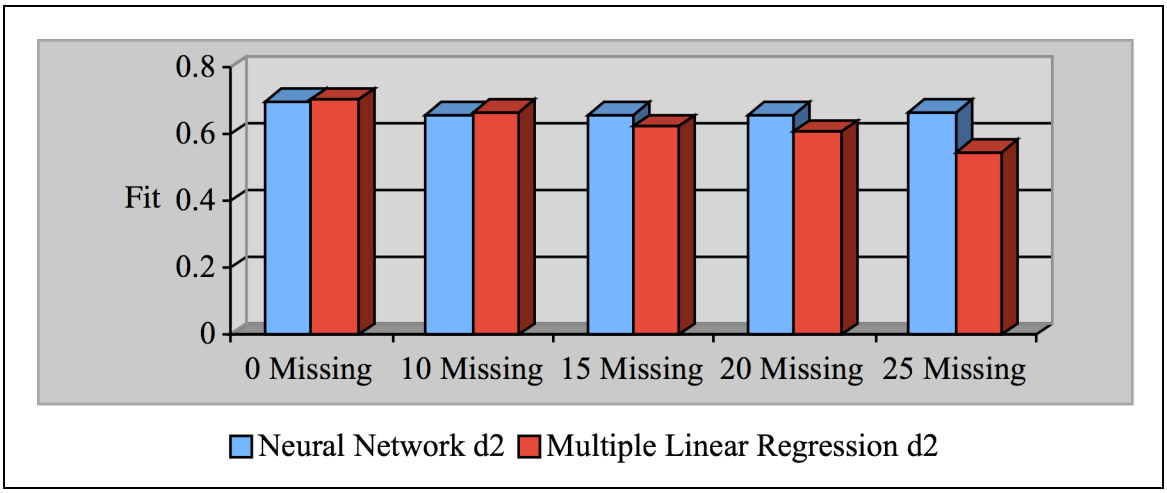


Figure 4-5. d_2 comparison of reading score prediction

For math scores prediction, again the neural network showed less degradation across all three measures of fitness and for the d_1 and d_2 statistics, the differences were statistically significant at an alpha level of .05, $t_{d1}(4)=2.43$, $p_{d1}=0.03$ (one-tailed) and $t_{d2}(4)=3.21$, $p_{d2}=0.02$ (one-tailed). See Tables 4-6, 4-7, and 4-8 and Figures 4-6, 4-7, and 4-8.

Table 4-6

RMSE Comparison of Multiple Linear Regression and Neural Network Models for Math Score Prediction with Synthetic Noise

Model	0 Mising Inputs	10 Missing Inputs	15 Mising Inputs	20 Missing Inputs	25 Missing Inputs
Multiple Linear Regression RMSE	0.08129	0.10062	0.08191	0.08137	0.08649
Neural Network RMSE	0.07351	0.07553	0.07624	0.08172	0.09606

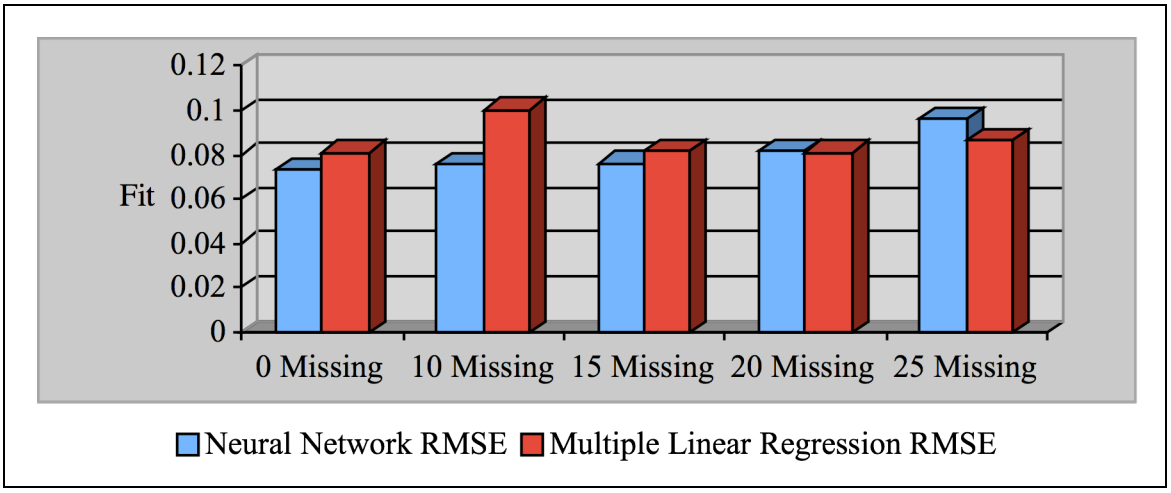


Figure 4-6. RMSE comparison of math score prediction

Table 4-7

d_1 Comparison of Multiple Linear Regression and Neural Network Models for Math Score Prediction with Synthetic Noise

Model	0 Mising Inputs	10 Missing Inputs	15 Mising Inputs	20 Missing Inputs	25 Missing Inputs
Multiple Linear Regression d_1	0.51019	0.44535	0.48406	0.46737	0.43895
Neural Network d_1	0.50250	0.50456	0.51160	0.50165	0.45765

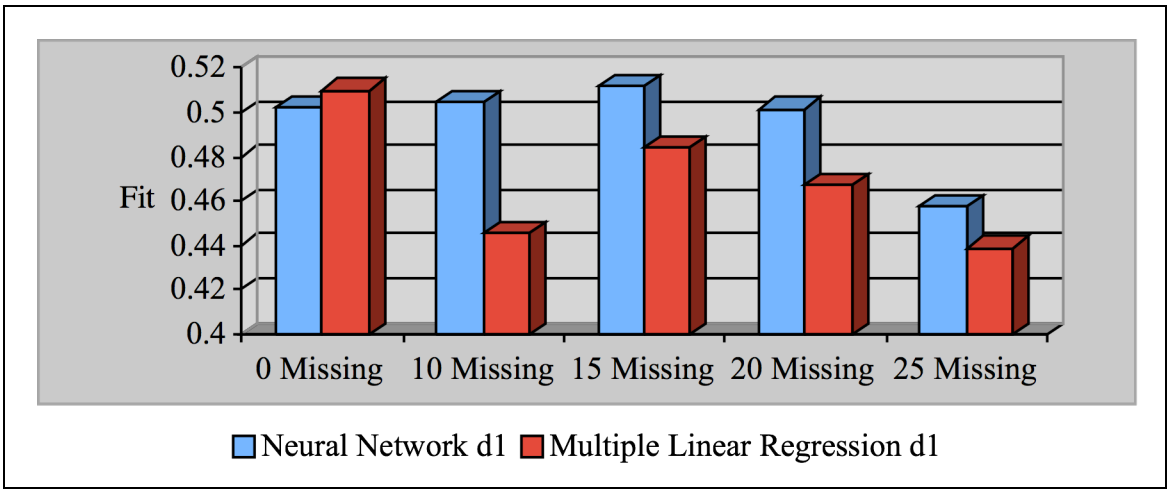


Figure 4-7. d_1 comparison of math score prediction

Table 4-8

d_2 Comparison of Multiple Linear Regression and Neural Network Models for Math Score Prediction with Synthetic Noise

Model	0 Mising Inputs	10 Missing Inputs	15 Mising Inputs	20 Missing Inputs	25 Missing Inputs
Multiple Linear Regression d_2	0.69804	0.62124	0.67066	0.65278	0.61710
Neural Network d_2	0.71164	0.70684	0.71357	0.69115	0.63791

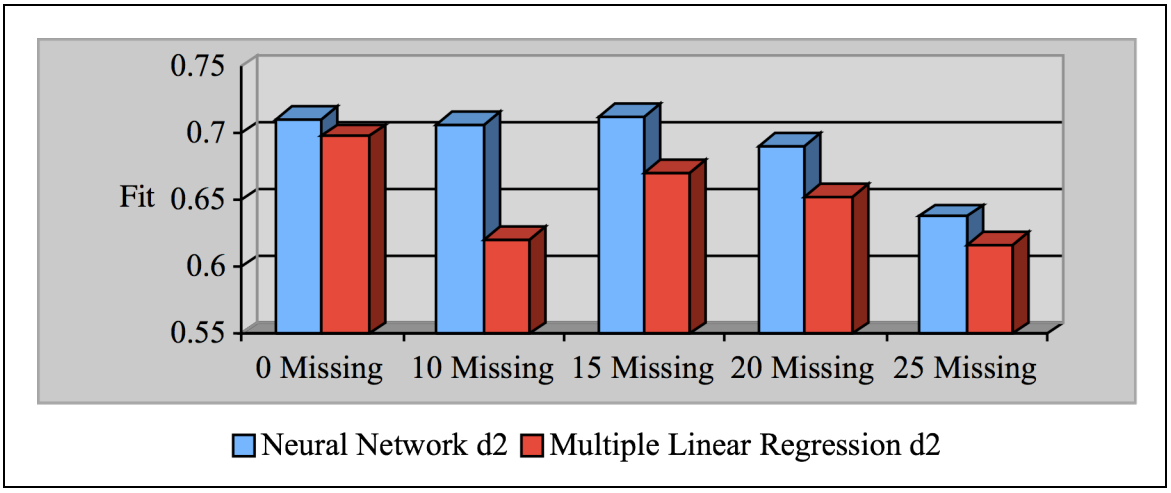


Figure 4-8. d_2 comparison of math score prediction

Summary

In terms of model performance the regression equation performed better on the training data, although this was not strictly unexpected. In the case of regression, the equation was the best possible fit for the training data. The neural network, however, avoided over fitting the training data by balancing the fit of the training data with the fit of a small set of cases reserved from the training data. Had the neural network been permitted to fit the training data as tightly as possible the total error could have been reduced to, or near, zero.

Comparison of the cross-validation data showed virtually no difference in performance between the two models. When noise was introduced to the cross-validation data, however, some differences did begin to emerge. Visually, the neural network appeared to slightly outperform the regression equations for both the math and reading scores. The differences, though, were only statistically significant for the d_1 and d_2 measures for the math scores.

CHAPTER 5

SUMMARY AND RECOMMENDATIONS

Introduction

As stated in Chapter 1, assessing student performance has been a common goal of those involved in the education enterprise. Through more accurate modeling, leading to more accurate predictions, educators and administrators may be able to glimpse a short distance into the future and identify students who are at risk, permitting earlier intervention in the hope of increasing the overall success rate of a given student population.

At present, however, modeling and prediction are commonly accomplished through a variety of tools such as analysis of variance, regression, and correlation. These methods, while very well understood, require a hypothesis that describes the interrelationships between the variables being studied. Consequently, studies often focus on small sets of variables in order to make the model understandable and the mathematics manageable.

Neural networks provide an alternate modeling tool that can be employed with complex data sets having no suitable hypothesis; or, as a means to arrive at a hypothesis by exploring a complex pool of data to learn what inferences might be drawn. Inspired by the biological processes that describe the function of real neurons and synapses, neural networks are able to learn from the data, infer the model from examples and then make predictions about never-before-seen cases.

The purpose of this study then, was to compare the performance of neural network modeling to that of multiple linear regression. These comparisons were drawn from a large pool of data collected by the National Center for Educational Statistics, using three measures of fit; root mean square error (RMSE) and Wilmot's indices of agreement, d_1 and d_2.

Summary of Procedures

For the purposes of modeling, data collected by the National Center for Educational Statistics, Educational Longitudinal Study: 2002 (ELS: 2002) was used to supply 103 independent non-curricular variables, and two dependent math and reading score variables.

Since this study focused on the performance of two modeling strategies, data that were already described in the literature as related, were chosen rather than discovering the theoretical foundation of the model. Specifically, the categories of non-curricular variables chosen from the ELS: 2002 were proved to be related to student achievement in several studies (Green, et all 1995; Ladd and Birch, 1997; Osterman, 2000; Ingels, et al., 2005).

Three pieces of software were developed to clean the data, perform the dummy coding required of the nominal and ordinal variables, perform case selection, execute the linear regression model, and train and execute the neural network model. The source code for these applications, shown in appendices B thru E. SPSS, was used to perform the linear regression and generate the regression coefficients for the math and reading score predictor equations.

The case selection process resulted in a set of cases to be used for training and another set to be used for cross-validation. The training data were used to develop regression equations for the math and reading scores as well as to train the neural network model. Once the regression coefficients had been determined and the neural network trained, the cross-validation cases were used to compare predictions made by linear regression and the neural network.

Summary of Findings

It was expected, based on other studies of neural network modeling, that the neural network would provide measurable improvements over regression analysis. This, however, was not the case. In a direct comparison of the neural network to multiple linear regression, the performance of the two models was virtually identical. The neural network had slightly lower RMSE values, where lower values demonstrate a better fit, with values of 0.07351 vs. 0.08129 for math scores and 0.07535 vs. 0.08218 for reading scores. Linear regression, however, had slightly higher d_1 and d_2 values for reading, where higher values demonstrate a better fit, with values of 0.50927 and 0.70343 vs. 0.50066 and 0.70230. For math scores, linear regression had a higher d_1 value 0.51019 vs. 0.50250 while the neural network had a higher d_2 value, 0.71164 vs. 0.69804.

A second comparison that introduced synthetic noise into the cross-validation data, however, did reveal that the neural network was somewhat more resistant to degradation when faced with noisy data. As the cross-validation was performed with progressively noisier data the neural network maintained lower overall RMSE values and

higher overall d_1 and d_2 values. This advantage, however, was only statistically significant for math score prediction and only for the d_1 and d_2 measures of fitness.

These results are somewhat surprising given the results of other neural network studies. For instance, in Xuefeng's study of auction price prediction that compared linear regression to a three-layer, feed-forward, back-propagation neural network similar in topology to the neural network employed in this study, the results indicated that the neural network achieved 91.29% accuracy compared to 76.46% accuracy for regression (Xuefeng, 2006). Sahoo's study of flash flooding and water quality that employed a similar neural network but with two hidden layers, achieved RMSE values near zero (Sahoo, 2006). And, in another study by Naik and Ragothaman of neural network prediction involving the success of MBA students, when neural network modeling was compared to the Logit and Probit models, results showed that the neural network achieved 89.13% accuracy while Logit and Probit achieved 72.83% and 73.37% respectively.

Given the results of these studies it was expected that the neural network would show more pronounced improvement over linear regression in predicting math and reading scores in this study. The difference between this study's result and that of other studies suggests the need to explore the differences between the previously mentioned models and the model in this study.

Perhaps the most significant difference involved the nature of the independent variables. In each of the previously mentioned studies, the independent variables were entirely, or mostly, scalar values. The independent variables for this study, however, were all nominal or ordinal. In order to present nominal and ordinal values to the neural

network the values were artificially scaled to the range [0.2, 0.8] as discussed in chapter 3. Alternatively, dummy coding could also have been used for the input layer of the neural network but that would have increased the number of input neurons from 103 to nearly 500 with a related increase in the size of the hidden layer. A neural network of that size would require significantly longer to train and, according to neural network rules of thumb, violate the need for 10 times the number of training cases as input neurons.

Two smaller differences were also present related to network topology and the nature of the dependent variables. The Sahoo (2006) study involved a four layer neural network having one input layer, two hidden layers, and one output layer. By contrast, the neural network model in this study was the more familiar three-layer topology with an input layer, hidden layer, and output layer. Naik's study involved a discrete, rather than continuous dependent variable. Naik's neural network classified MBA candidates as either "successful" or "marginal" while the neural network in this work predicted reading and math scores as continuous values ranging from [0.0, 1.0].

One additional benefit demonstrated by the neural network, however, was the neural network's ability to predict multiple dependent variables simultaneously. In order to predict both math and reading scores, two regression equations were developed: one for each dependent variable. In the case of the neural network, though, a single network was able to predict both scores simultaneously.

Research Implications

The results of this study suggest that neural networks can be at least as good as linear regression in developing predictive models and suffer less from degradation in the

face of noisy data. As such, neural networks deserve further study in the field of educational modeling and should be considered as tool in that regard. Neural networks may provide avenues to more complex studies involving subtle or unknown relationships that then direct researchers to other methods of study on more narrow bands of data.

Neural networks also demonstrate remarkable flexibility. Researchers have the ability to shape the input data by transforming it from one representation to another as needed by the input layer. Studies may also investigate a variety of network topologies in seeking the most appropriate modeling architecture for a given problem.

Furthermore, the initial investment in training a neural network is not lost when a model is used to make predictions. If the accuracy of the network begins to decrease over time, small re-training sessions with new data can realign the existing model without the need for the full training that was done initially.

Neural networks also have the ability to predict multiple dependent variables with a single network. In this study, math and reading scores were predicted simultaneously with a single neural network. In terms of data manipulation this has the potential to reduce the burden on researchers in terms of the number of instruments that must be managed.

Limitations of the Research

This study employed a single neural network model, and thus the results are applicable only to three-layer, feed-forward, back-propagation networks using the sigmoid transfer function in the hidden and output layers. Other network topologies utilizing a different number of layers and/or different transfer functions may perform

differently. Alternate schemes for encoding the independent variables for presentation to the input layer may also affect the performance of the network.

Finally, the results of this study are only representative of this data set. Therefore, the results may be generalizable to data gathered from other 10^{th} graders in public, private and Catholic schools, but it is unknown if the results would be the same for data from students in other grades or other types of institutions.

Future Research

Given the virtually identical performance of both models in this study, and the resistance to degradation shown by the neural network when presented with noisy data, it would be of value to compare model performance when training with imperfect data. Recall that during the case selection process, any cases with invalid or missing values were discarded. This resulted in training data that was free from noise. This perfect training data was then used to develop the regression equations and train the neural network.

Noisy or imperfect data, however, is more the norm than the exception in typical field research. Noisy data are often handled through various imputation processes to fill in the blanks. If neural networks could be shown to accurately model noisy data researchers would have a powerful new tool that did not require the present data imputation techniques.

In the near future this study could be repeated using the same data, software, and procedures with a single modification in the case selection process: selecting from the entire sample population and not just the cases that were free from noise. This would

result in training data and cross-validation data that contained noisy samples. The comparisons in this study would then demonstrate whether or not neural networks have any advantage over linear regression when starting with imperfect data.

What's more, it would be of value to study what characteristics, if any, of a given set of data lend themselves to neural network modeling. For instance, future researchers could perform correlations between the independent and dependent variables and then test neural network modeling to see if higher correlation values imply that the data are better suited to neural network modeling. The converse may also be interesting if future research found that neural networks were able to model un-correlated data.

Future research might also explore neural networks that dummy code nominal and ordinal input data as opposed to converting such data to a scalar format. This necessarily expands the size of the input layer and places additional burdens on the training data, but it would be valuable to explore what impact, if any, dummy coding has on neural network performance.

Clearly, neural network modeling shows promise in the areas of modeling and prediction. Additional study is needed, however, to understand its full utility in educational research and the circumstances under which neural network modeling is most effective. It is possible that such research could lead to pre-configured modeling software specifically for educational researchers to further enhance their ability to understand the complex landscape of the teaching and learning enterprise.

References

Baker, B. D., & Richards, C. E. (1999). A Comparison of Conventional Linear Regression Methods and Neural Networks for Forecasting Educational Spending. *Economics of Education Review, 18*, 405-415.

Bandura, A. (1986). *Social Foundations of Thought and Action: A Social Cognitive Theory*. Englewood Cliffs, NJ: Prentice Hall.

Bandura, A. (1997). *Self-Efficacy: The Exercise of Control*. New York: W.H. Freeman and Co.

Blum, A. (1992). *Neural Networks in C++*. Wiley.

Boger, Z., Kuflik, T., Shoval, P., & Shapira, B. (2000). Automatic Keyword Identification by Artificial Neural Networks Compared to Manual Identification by Users of Filtering Systems. *Information Processing and Management, 37*(2), 187-198.

Bridglall, B. L. & Gordon, E. W. (2003). *Raising Minority Academic Achievement: The Department of Defense Model* (Report No. EDO-UD-03-8). Washington D. C.: Department of Education. (ERIC Number ED480919).

Chen, C., Campbell, V. C., & Suleiman, A. (2001). *Predicting Student Performance at a Minority Professional School.* Paper presented at the Annual Meeting of the Association for Institutional Research, Long Beach, CA.

Cohen, J. (1977). *Statistical Power Analysis for the Behavioral Sciences.* New York: Academic Press.

Crawford, L., Tindal, G., & Stieber, S. (2001). Using Oral Reading Rate to Predict Student Performance on Statewide Assessment Tests. *Educational Assessment, 7*, 303-323.

Comrie, A. C. (1997). Comparing Neural Networks and Regression Models for Ozone Forecasting. *Journal of Air & Waste Management Association*, *47*, 653-663.

Gonzalez, J. M. B. & DesJardins, S. L. (2001). *Artifical Neural Networks: A New Approach to Predicting Application Behavior. AIR 2001 Annual Forum Paper.* Paper presented at the Annual Forum for the Association for Institutional Research, Long Beach. CA.

Gonzalez, J. M. B. & DesJardins, S. L. (2002). Artifical Neural Networks: A New Approach to Predicting Application Behavior. *Research in Higher Education*, *43*, 235-258.

Green, P.J., Dugoni, B.L., Ingels, S.J., and Camburn, E. (1995). *A Profile of the American High School Senior in 1992* (NCES 95–384). U.S. Department of Education, National Center for Education Statistics. Washington, DC: U.S. Government Printing Office.

Hammouri, H.A.M. (2004). Attitudinal and motivational variables related to mathematics achievement in Jordan: Findings from the Third International Mathematics and Science Study. *Educational Research*, 46, 214-257.

Holbrook, R. G. (2003). Impact of selected non-curricular variables on regular education student achievement as measured by the 2001-2002 reading and mathematics PSSA scores. *Dissertation Abstracts International, 64*(12), 4289A. (UMI No. 3116331)

Hoefer, P., & Gould, J. (2000). Assessment of Admission Criteria for Predicting Students' Academic Performance in Graduate Business Programs. *Journal of Education for Business, 75*, 225-229.

House, D. J., & Keeley, E. J. (1993). *Differential Prediction of Graduate Student Achievement from Miller Analogies Test Scores.* Paper presented at the Annual Meeting of the Illinois Association for Institutional Research, Oakbrook, IL.

Ince, H., & Trafalis, T. B. (2005). A Hybrid Model for Exchange Rate Prediction. *Decision Support Systems, 42,* 1054-1062.

Ingels, S. J., Chen, X., and Owings, J. A. (2005). A Profile of the American Highschool Sophomore in 2002 (NCES 2005-338). *U.S. Department of Education. National Center for Education Statistics.* Washington, DC: U.S. Government Printing Office.

Ladd, G.W., and Birch, S.H. (1997). The Teacher-Child Relationship and Children's Early School Adjustment. *Journal of School Psychology*, *35*(1), 61–79.

Lafee, S. (2002). Data-Driven Districts. *School Administrator*, *59*(11), 6-7, 9-10, 12, 14-15.

Lashway, L. (2002). Data Analysis for School Improvement. *Research Roundup*, *19*(2), 1-4.

Luan, J. (2002, June). *Data Mining and Knowledge Management in Higher Education – Potential Applications*. Paper presented at the Annual Forum for the Association for Institutional Research, Toronto, Ontario, Canada.

Mandelbrot, B. B., & Hudson, R. L. (2004). *The (mis)Behavior of Markets*. New York: Basic Books.

Mandic, D. P., & Chambers, J. A. (2001). Recurrent *Neural Networks for Prediction*. New York: Wiley.

Martynenko, A. I., & Yang, S. X. (2006). Biologically Inspired Neural Computation for Gunseng Drying Rate. *Biosystems Engineering*, *95*, 385-396.

Murat, Y. S., & Ceylan, H. (2006). Use of Artificial Neural Networks for Transport Energy Demand Modeling. *Energy Policy*, 34, 3165-3172.

Naik, B., & Ragothaman, S. (2004). Using Neural Networks to Predict MBA Student Success. *College Student Journal, 38*(1), 143-149.

Odom, M.D., & Sharda, R. (1990). *A Neural Network for Banruptcy Prediction*. Paper presented at the1990 IJCNN International Joint Conference on Neural Networks, San Diego, CA.

Osterman, K. F. (2000). Students' Need for Belonging in the School Community. *Review of Educational Research*, *70*, 323–367.

Pajares, F., & Graham, L. (1999). Self-efficacy, motivation constructs, and mathematics performance of entering middle school students. *Contemporary Educational Psychology*, 24, 124-139.

Pintrich, P.R. & De Groot E. (1990). Motivational and self-regulated learning components of classroom academic performance. *Journal of Educational Psychology*, 82(1), pp. 33-50.

Reed, R. D., & Marks, R. J, II. (1999). *Neural Smithing: Supervised Learning in Feedforward Artifical Neural Networks.* Cambridge: The MIT Press.

Russell, U., & Zhang, S. (2006). *The Role of Demographic Factors in Predicting Student Performance on a State Reading Test.* Paper presented at the Annual Meeting of the American Edicational Research Association, San Francisco, CA.

Saad, E. W., Prokhorov, D. V., & Wunsch, D. C. II. (1998). Comparative Study of Stock Trend Prediction Using Time Delay, Recurrent and Probabilistic Neural Networks. *IEEE Transactions on Neural Networks,* 9, 1456-1470.

Sahoo, G. B., Ray, C., & De Carlo, E. H. (2006). Use of Neural Network to Predict Flash Flood and Attendant Water Qualities of a Mountainous Stream on Oahu, Hawaii. *Journal of Hydrology, 327*, 525-538.

Satinover, S. (2001). *The Quantum Brain*. New York: Wiley.

Skidmore, R. L. (2003). *Predicting Final Examination Grades in a Self-Paced Introductory Psychology Course: The Role of Motivational Orientation, Learning Strategies, Procrastination, and Perception of Daily Hassles.* Paper presented at the Annual Meeting of the Mid-South Educational Research Association, Biloxi, MS.

Streifer, P. A., & Shumann, J. A. (2005) Using Data Mining to Identify Actional Information: Breaking New Group in Data-Driven Decision Making. *Journal of Education for Students Placed At Risk, 10*, 281-293.

Snyder, R. M. (1996). *Neural Networks for the Beginner.* Paper presented at the Association for Small Computer Users in Education, Myrtle Beach, SC.

Tell, C. A., & McDonald, D. (2003). *The First Year: Students' Performance on 10th Grade Standards and Subsequent Performance in the First Year of College (2001-02).* Paper presented at the Annual Meeting of the Association for the Student of Higher Education, Portland, OR.

Tokar, A. S., & Johnson, P. A. (1999). Rainfall-Runoff Modeling Using Artificial Neural Networks. *Journal of Hydrological Engineering, 4*, 232-239.

Xuefeng, L., Lu, L., Lihua, W., & Zhang, Z. (2006). Predicting the Final Prices of Online Auction Items. *Expert Systems with Applications, 31*, 542-550.

Yi, P. J. (1996). A Neural Network Model Forecasting for Prediction of Daily Maximum Ozone Concentrations in an Industrialized Urban Area. *Environmental Pollution, 92*, 349-357.

APPENDIX A

Inventory of Variables

Table A-1

Independent Variable Inventory

Variable	Description
BYFCOMP	A nominal measure of the family configuration with values (1) mother and father, (2) mother and male guardian, (3) father and guardian, (4) two guardians, (5) mother only, (6) father only, (7) female guardian only, (8) male guardian only, and (9) parent or guardian lives with student less than ½ time.
FATHED	A nominal measure of the father's highest level of education with values (1) did not finish high school, (2) graduated from high school, (3) attended 2-year school, no degree, (4) graduated from 2-year school, (5) attended college, no 4-year degree, (6) graduated from college, (7) completed master's degree, and (8) completed PhD, MD, or other advanced degree.
MOTHED	A nominal measure of the mother's highest level of education with the same possible values as FATHED.
PARED	A nominal measure similar to FATHED and MOTHED reporting the highest level of education attained by either parent.
SES1QU	An ordinal measure of the student's socioeconomic status as classified in one of four quartiles with values (1) lowest quartile, (2) second quartile, (3) third quartile, and (4) highest quartile.

Table A-1 *(continued).*

STEXPECT	An ordinal measure of how far the student believes they will go in school with values (-1) don't know, (1) less than high school graduation, (2) high school graduation or GED, (3) attend and/or complete 2-year college/school, (4) attend college, 4-year degree incomplete, (5) graduate from college, (6) obtain master's degree or equivalent, and (7) obtain PhD, MD, or other advanced degree.
BYBASEBL	An ordinal measure of the student's participation in interscholastic baseball with values (1) no interscholastic team, (2) did not participate, (3) participated at the junior varsity level, (4) participated at the varsity level, and (5) participated as varsity captain.
BYSOFTBL	An ordinal measure of the student's participation in interscholastic softball with values (1) no interscholastic team, (2) did not participate, (3) participated at the junior varsity level, (4) participated at the varsity level, and (5) participated as varsity captain.
BYBSKTBL	An ordinal measure of the student's participation in interscholastic basketball with values (1) no interscholastic team, (2) did not participate, (3) participated at the junior varsity level, (4) participated at the varsity level, and (5) participated as varsity captain.
BYFOOTBL	An ordinal measure of the student's participation in interscholastic football with values (1) no interscholastic team, (2) did not participate, (3) participated at the junior varsity level, (4) participated at the varsity level, and (5) participated as varsity captain.

Table A-1 *(continued).*

BYSOCCER	An ordinal measure of the student's participation in interscholastic soccer with values (1) no interscholastic team, (2) did not participate, (3) participated at the junior varsity level, (4) participated at the varsity level, and (5) participated as varsity captain.
BYTEAMSP	An ordinal measure of the student's participation in other interscholastic team sports with values (1) no interscholastic team, (2) did not participate, (3) participated at the junior varsity level, (4) participated at the varsity level, and (5) participated as varsity captain.
BYSOLOSP	An ordinal measure of the student's participation in interscholastic baseball with values (1) no interscholastic team, (2) did not participate, (3) participated at the junior varsity level, (4) participated at the varsity level, and (5) participated as varsity captain.
BYSCTRL	An ordinal measure of school control with values (1) public, (2) catholic, and (3) other private.
BYURBAN	An ordinal measure of the school's locale with values (1) urban, (2) suburban, and (3) rural.
BYREGION	An ordinal measure of the schools region with values (1) northeast, (2) midwest, (3) south, and (4) west.
BYS20A	An ordinal measure of the students' agreement with the statement that they get along with their teachers with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS20B	An ordinal measure of the students' agreement with the statement that there is school spirit with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.

Table A-1 *(continued).*

BYS20C	An ordinal measure of the students' agreement with the statement that they are friendly with other racial groups with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS20E	An ordinal measure of the students' agreement with the statement that the teaching is good with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS20F	An ordinal measure of the students' agreement with the statement that teachers are interested in the students with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS20G	An ordinal measure of the students' agreement with the statement that teachers praise effort with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS20J	An ordinal measure of the students' agreement with the statement that they do not feel safe at school with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS20M	An ordinal measure of the students' agreement with the statement that there are gangs at school with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS20N	An ordinal measure of the students' agreement with the statement that racial/ethnic groups often fight with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.

Table A-1 *(continued).*

BYS21A	An ordinal measure of the students' agreement with the statement that everyone knows the school rules with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS21B	An ordinal measure of the students' agreement with the statement that school rules are fair with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS21C	An ordinal measure of the students' agreement with the statement that punishment is the same no matter who you are with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS21D	An ordinal measure of the students' agreement with the statement that school rules are strictly enforced with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS21E	An ordinal measure of the students' agreement with the statement that student know punishment for broken rules with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS22A	An ordinal measure of whether the student had anything stolen at school with values (1) never, (2) once or twice, and (3) more than twice.
BYS22B	An ordinal measure of whether the student was ever offered drugs at school with values (1) never, (2) once or twice, and (3) more than twice.
BYS22C	An ordinal measure of whether anyone ever threatened to hurt the student with values (1) never, (2) once or twice, and (3) more than twice.

Table A-1 *(continued).*

BYS22D	An ordinal measure of whether the student ever got in to a physical fight with values (1) never, (2) once or twice, and (3) more than twice.
BYS22E	An ordinal measure of whether the student was ever hit with values (1) never, (2) once or twice, and (3) more than twice.
BYS22F	An ordinal measure of whether anyone ever forced money or items from the student with values (1) never, (2) once or twice, and (3) more than twice.
BYS22G	An ordinal measure of whether anyone ever damaged the student's belongings with values (1) never, (2) once or twice, and (3) more than twice.
BYS22H	An ordinal measure of whether anyone ever bullied or picked on the student with values (1) never, (2) once or twice, and (3) more than twice.
BYS24B	A nominal measure of how many times the student cut classes with values (1) never, (2) 1-2 times, (3) 3-6 times, (4) 7-9 times, (5) 10 or more times.
BYS26	An ordinal measure of the student's high school program with values (1) general, (2) college prepatory/academic, (3) vocational including technical/business.
BYS27A	An ordinal measure of the students' agreement with the statement that classes are interesting and challenging with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.

Table A-1 *(continued).*

BYS27B	An ordinal measure of the students' agreement with the statement that they are satisfied by doing what is expected in class with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS27C	An ordinal measure of the students' agreement with the statement that they have nothing better to do than school with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS27D	An ordinal measure of the students' agreement with the statement that education is important to get a job later with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS27E	An ordinal measure of the students' agreement with the statement that school is a place to meet friends with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS27F	An ordinal measure of the students' agreement with the statement that they attend school to play a sport or attend a club with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS27G	An ordinal measure of the students' agreement with the statement that they learn job skills in school with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS27H	An ordinal measure of the students' agreement with the statement that teachers expect success in school with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.

Table A-1 *(continued).*

BYS27I	An ordinal measure of the students' agreement with the statement that parents expect success in school with values (1) strongly agree, (2) agree, (3) disagree, and (4) strongly disagree.
BYS28	An ordinal measure of how much the student likes school with values (-1) don't know, (1) not at all, (2) somewhat, (3) a great deal.
BYS34A	A nominal measure of how many hours/week the student spent on homework in school with values (-1) don't know, (0) 0 hours, (1) 1 hour, (2) 2 hours, (3) 3 hours, (4) 4 hours, (5) 5 hours, (6) 6 hours, (7) 7 hours, (8) 8 hours, (9) 9 hours, (10) 10 hours, (11) 11 hours, (12) 12 hours, (13) 13 hours, (14) 14 hours, (15) 15 hours, (16) 16 hours, (17) 17 hours, (18) 18 hours, (19) 19 hours, (20) 20 hours, and (21) 21 or more hours.
BYS34B	A nominal measure of how many hours/week the student spent on homework out of school with values (0) 0 hours, (1) 1 hour, (2) 2 hours, (3) 3 hours, (4) 4 hours, (5) 5 hours, (6) 6 hours, (7) 7 hours, (8) 8 hours, (9) 9 hours, (10) 10 hours, (11) 11 hours, (12) 12 hours, (13) 13 hours, (14) 14 hours, (15) 15 hours, (16) 16 hours, (17) 17 hours, (18) 18 hours, (19) 19 hours, (20) 20 hours, (21) 21 hours, (22) 22 hours, (23) 23 hours, (24) 24 hours, (25) 25 hours, and (26) 26 or more hours.

Table A-1 *(continued).*

BYS35A	A nominal measure of how many hours/week the student spent on math homework in school with values (-1) don't know, (0) 0 hours, (1) 1 hour, (2) 2 hours, (3) 3 hours, (4) 4 hours, (5) 5 hours, (6) 6 hours, (7) 7 hours, (8) 8 hours, (9) 9 hours, (10) 10 hours, (11) 11 hours, (12) 12 hours, (13) 13 hours, (14) 14 hours, (15) 15 hours, (16) 16 hours, (17) 17 hours, (18) 18 hours, (19) 19 hours, (20) 20 hours, and (21) 21 or more hours.
BYS35B	A nominal measure of how many hours/week the student spent on math homework out of school with values (-1) don't know, (0) 0 hours, (1) 1 hour, (2) 2 hours, (3) 3 hours, (4) 4 hours, (5) 5 hours, (6) 6 hours, (7) 7 hours, (8) 8 hours, (9) 9 hours, (10) 10 hours, (11) 11 hours, (12) 12 hours, (13) 13 hours, (14) 14 hours, (15) 15 hours, (16) 16 hours, (17) 17 hours, (18) 18 hours, (19) 19 hours, (20) 20 hours, and (21) 21 or more hours.
BYS36A	A nominal measure of how many hours/week the student spent on English homework in school with values (-1) don't know, (0) 0 hours, (1) 1 hour, (2) 2 hours, (3) 3 hours, (4) 4 hours, (5) 5 hours, (6) 6 hours, (7) 7 hours, (8) 8 hours, (9) 9 hours, (10) 10 hours, (11) 11 hours, (12) 12 hours, (13) 13 hours, (14) 14 hours, (15) 15 hours, (16) 16 hours, (17) 17 hours, (18) 18 hours, (19) 19 hours, (20) 20 hours, and (21) 21 or more hours.

Table A-1 *(continued).*

BYS36B	A nominal measure of how many hours/week the student spent on English homework out of school with values (-1) don't know, (0) 0 hours, (1) 1 hour, (2) 2 hours, (3) 3 hours, (4) 4 hours, (5) 5 hours, (6) 6 hours, (7) 7 hours, (8) 8 hours, (9) 9 hours, (10) 10 hours, (11) 11 hours, (12) 12 hours, (13) 13 hours, (14) 14 hours, (15) 15 hours, (16) 16 hours, (17) 17 hours, (18) 18 hours, (19) 19 hours, (20) 20 hours, and (21) 21 or more hours.
BYS37	An ordinal measure of the importance of good grades to the student with values (1) not important, (2) somewhat important, (3) important, and (4) very important.
BYS38B	An ordinal measure of how often the student goes to class without books with values (1) never, (2) seldom, (3) often, and (4) usually.
BYS39A	An ordinal measure of whether the student played intramural baseball with values (1) school doesn't have team, (2) no, and (3) yes.
BYS39B	An ordinal measure of whether the student played intramural softball with values (1) school doesn't have team, (2) no, and (3) yes.
BYS39C	An ordinal measure of whether the student played intramural basketball with values (1) school doesn't have team, (2) no, and (3) yes.
BYS39D	An ordinal measure of whether the student played intramural football with values (1) school doesn't have team, (2) no, and (3) yes.
BYS39E	An ordinal measure of whether the student played intramural soccer with values (1) school doesn't have team, (2) no, and (3) yes.

Table A-1 *(continued).*

BYS39F	An ordinal measure of whether the student played another intramural team sport with values (1) school doesn't have team, (2) no, and (3) yes.
BYS39G	An ordinal measure of whether the student played an individual intramural sport with values (1) school doesn't have team, (2) no, and (3) yes.
BYS39H	An ordinal measure of whether the student participated on an intramural cheerleading/drill team with values (1) school doesn't have team, (2) no, and (3) yes.
BYS41A	An ordinal measure of whether the student participated in band or chorus with values (-1) don't know, (1) no, and (2) yes.
BYS41B	An ordinal measure of whether the student participated in a school play or musical with values (1) no and (2) yes.
BYS41C	An ordinal measure of whether the student participated in student government with values (1) no and (2) yes.
BYS41D	An ordinal measure of whether the student participated in an academic honor society with values (1) no and (2) yes.
BYS41E	An ordinal measure of whether the student participated in the school yearbook or newspaper with values (1) no and (2) yes.
BYS41F	An ordinal measure of whether the student participated in school service clubs with values (-1) don't know, (1) no, and (2) yes.
BYS41G	An ordinal measure of whether the student participated in school academic clubs with values (-1) don't know, (1) no, and (2) yes.

Table A-1 *(continued).*

BYS41H	An ordinal measure of whether the student participated in school hobby clubs with values (-1) don't know, (1) no, and (2) yes.
BYS41I	An ordinal measure of whether the student participated in school vocational clubs with values (-1) don't know, (1) no, and (2) yes.
BYS42	A nominal measure of the hours/week the student spent on extracurricular activities with values (-1) don't know, (0) 0 hours, (1) 1 hour, (2) 2 hours, (3) 3 hours, (4) 4 hours, (5) 5 hours, (6) 6 hours, (7) 7 hours, (8) 8 hours, (9) 9 hours, (10) 10 hours, (11) 11 hours, (12) 12 hours, (13) 13 hours, (14) 14 hours, (15) 15 hours, (16) 16 hours, (17) 17 hours, (18) 18 hours, (19) 19 hours, (20) 20 hours, and (21) 21 or more hours.
BYS43	A nominal measure of the hours/week the student spent reading outside of school with values (-1) don't know, (0) 0 hours, (1) 1 hour, (2) 2 hours, (3) 3 hours, (4) 4 hours, (5) 5 hours, (6) 6 hours, (7) 7 hours, (8) 8 hours, (9) 9 hours, (10) 10 hours, (11) 11 hours, (12) 12 hours, (13) 13 hours, (14) 14 hours, (15) 15 hours, (16) 16 hours, (17) 17 hours, (18) 18 hours, (19) 19 hours, (20) 20 hours, and (21) 21 or more hours.
BYS45A	An ordinal measure of how often the student uses computer for fun with values (1) never, (2) rarely, (3) less than once a week, (4) once or twice a week, and (5) everyday or almost everyday.

Table A-1 *(continued).*

BYS45B	An ordinal measure of how often the student uses computer for school work with values (1) never, (2) rarely, (3) less than once a week, (4) once or twice a week, and (5) everyday or almost everyday.
BYS45C	An ordinal measure of how often the student uses computer to learn on his own with values (1) never, (2) rarely, (3) less than once a week, (4) once or twice a week, and (5) everyday or almost everyday.
BYS46A	A nominal measure of how many hours/day the student uses a computer for school work with values (0) 0 hours, (1) 1 hour, (2) 2 hours, (3) 3 hours, (4) 4 hours, (5) 5 hours, and (6) 6 or more hours.
BYS46B	A nominal measure of how many hours/day the student uses a computer for things other than school work with values (0) 0 hours, (1) 1 hour, (2) 2 hours, (3) 3 hours, (4) 4 hours, (5) 5 hours, and (6) 6 or more hours.
BYS54A	An ordinal measure of how important being successful at work is to the student with values (1) not important, (2) somewhat important, and (3) very important.
BYS54B	An ordinal measure of how important marrying the right person/having a happy family is to the student with values (-1) don't know, (1) not important, (2) somewhat important, and (3) very important.
BYS54C	An ordinal measure of how important having lots of money is to the student with values (-1) don't know, (1) not important, (2) somewhat important, and (3) very important.

Table A-1 *(continued).*

BYS54D	An ordinal measure of how important having strong friendships is to the student with values (1) not important, (2) somewhat important, and (3) very important.
BYS54E	An ordinal measure of how important being able to find steady work is to the student with values (-1) don't know, (1) not important, (2) somewhat important, and (3) very important.
BYS54F	An ordinal measure of how important helping others in the community is to the student with values (1) not important, (2) somewhat important, and (3) very important.
BYS54G	An ordinal measure of how important giving children better opportunities is to the student with values (1) not important, (2) somewhat important, and (3) very important.
BYS54H	An ordinal measure of how important living close to parents/relatives is to the student with values (1) not important, (2) somewhat important, and (3) very important.
BYS54I	An ordinal measure of how important getting away from this area is to the student with values (-1) don't know, (1) not important, (2) somewhat important, and (3) very important.
BYS54J	An ordinal measure of how important working to correct inequalities is to the student with values (-1) don't know, (1) not important, (2) somewhat important, and (3) very important.
BYS54K	An ordinal measure of how important having children is to the student with values (-1) don't know, (1) not important, (2) somewhat important, and (3) very important.

Table A-1 *(continued).*

BYS54L	An ordinal measure of how important having leisure time is to the student with values (-1) don't know, (1) not important, (2) somewhat important, and (3) very important.
BYS54N	An ordinal measure of how important being an expert in a field of work is to the student with values (-1) don't know, (1) not important, (2) somewhat important, and (3) very important.
BYS54O	An ordinal measure of how important getting a good education is to the student with values (1) not important, (2) somewhat important, and (3) very important.
BYS57	An ordinal measure of the student's plan to continue education after high school with values (-1) don't know, (1) yes, right after high school, (2) yes, after being out of high school 1 year, (3) yes, after being out of high school more than 1 year, (4) yes, but don't know when, and (5) no, don't plan to continue education.
BYS60	An ordinal measure of whether the student want to play athletics in college with values (0) no and (1) yes.
BYS61	An ordinal measure of whether the student hopes to receive an athletic scholarship for college with values (0) no and (1) yes.
BYS66A	An ordinal measure of the mother's desire for the 10^{th} grader after high school with values (-1) don't know, (1) go to college, (2) get a full-time job, (3) enter trade school or apprenticeship, (4) enter military service, (5) get married, and (6) whatever the student wants to do.

Table A-1 *(continued).*

BYS66B	An ordinal measure of the father's desire for the 10th grader after high school with values (-1) don't know, (1) go to college, (2) get a full-time job, (3) enter trade school or apprenticeship, (4) enter military service, (5) get married, and (6) whatever the student wants to do.
BYS66F	An ordinal measure of the favorite teacher's desire for the 10th grader after high school with values (-1) don't know, (1) go to college, (2) get a full-time job, (3) enter trade school or apprenticeship, (4) enter military service, (5) get married, and (6) whatever the student wants to do.
BYS72	An ordinal measure of whether the student ever worked for pay not around the house with values (1) no, (2) yes, currently employed, and (3) yes, not currently employed.

APPENDIX B

Case Selection Source Listing

```
import java.util.ArrayList;
import java.util.Random;
import java.io.*;

/**
 * This program selects cases from the total population of 15,362 cases to create
 * a 10% (N=1536) sample for training purposes, to train the neural net as well as
 * to perform the linear regression, and a 10% (N=1536) sample to test the accuracy
 * of the neural net and the linear regression equation.
 *
 * This program also removes all cases with missing data so that the selection is
 * made from cases with no missing values. This may result in training and test
 * populations less than N=1536
 *
 * Note: After cleaning the data, 3,068 cases remain so the total number of cases in
 * each 10% sample is actually 1,534
 *
 */
public class CaseSelector {

    private static int[][]inputFeatures = new int[103][];
    private static String[] inputNames = new String[103];
    private static ArrayList dummyFreqs;
    private static int dummyCount = 0;
    private static int categoryCount = 0;

    private static ArrayList inputCases = new ArrayList();
    private static int[][] dummyCoding;
    private static int[][] NNCoding;
    private static String[] mathTargets;
    private static String[] readingTargets;

    //The position of the reading score in the input data
    private static int READING_SCORE_INDEX = 8;
    //The position of the math score in the input data
    private static int MATH_SCORE_INDEX = 7;

    // Used to randomly select input variables to code as "missing" to generate
    // noisy test files.
    private static int[] noise = new int[0];

    public static void main(String[] args) {

        // Initialize the input features
        setupInputFeatures();

        // Determine the total number of Dummy Variables
        for (int i=0;i<inputFeatures.length;i++) {
            dummyCount+=inputFeatures[i].length-1; // The total number of dummy vars
            categoryCount += inputFeatures[i].length; // The total number of categories
        }
        System.out.println("Total number of Dummy Variables = "+dummyCount);

        loadInputCases("ELS2002_RawDataSet.dat");

        // Perform dummy coding of input cases into the dummyCoding array
        dummyCoding = new int[inputCases.size()][dummyCount];
        NNCoding = new int[inputCases.size()][inputFeatures.length];
        mathTargets = new String[inputCases.size()];
        readingTargets = new String[inputCases.size()];
        performDummyCoding();

        // Now create the dummyFreqs array to determine which case #'s have 1
        // for each of the dummy variables. dummyFreqs if an ArrayList that contains
        // ArrayLists. The dummyFreqs ArrayList indexes all of the dummy variables.
        // The inner ArrayLists each contain a list of the case #'s that have a 1 for
        // that Dummy Variable.
        performDummyFrequencies();

        // Now peform the case selection.
        // This will return a 2 dimensional array representing two lists of case #'s
```

```
    int [][] selectedCases = performCaseSelection();

    // Now check to ensure we didn't select any duplicates
    if (!performCrossCheck(selectedCases)) {
        System.out.println("Error during cross check - duplicates found.");
        System.exit(-1);
    } else {
        System.out.println("Cross check OK");
    }

    //Finally, write out 4 files. Two containing the dummy coded training cases and
    //dummy coded test cases and two containing NN coded training and test cases
    writeOutputFile("ELS2002_DummyCoded_Training.txt", selectedCases[0]);
    writeNNOutputFile("ELS2002_NNCoded_Training.txt", selectedCases[0]);
    writeOutputFile("ELS2002_DummyCoded_Test.txt", selectedCases[1]);
    writeNNOutputFile("ELS2002_NNCoded_Test.txt", selectedCases[1]);

    // Now, generate progressively more and more noise, and write out
    // dummy coded and NN coded files with more and more variables flagged as missing
    noise = generateNoise(noise,10); // 10 input vars
    performDummyCoding();
    writeOutputFile("ELS2002_DummyCoded_Test_Noise_"+noise.length+".txt",
        selectedCases[1]);
    writeNNOutputFile("ELS2002_NNCoded_Test_Noise_"+noise.length+".txt",
        selectedCases[1]);
    writeNoisyVariableInventory("ELS2002_NoisyVariables_"+noise.length+".txt");

    noise = generateNoise(noise,5); // 15 input vars
    performDummyCoding();
    writeOutputFile("ELS2002_DummyCoded_Test_Noise_"+noise.length+".txt",
        selectedCases[1]);
    writeNNOutputFile("ELS2002_NNCoded_Test_Noise_"+noise.length+".txt",
        selectedCases[1]);
    writeNoisyVariableInventory("ELS2002_NoisyVariables_"+noise.length+".txt");

    noise = generateNoise(noise,5); // 20 input vars
    performDummyCoding();
    writeOutputFile("ELS2002_DummyCoded_Test_Noise_"+noise.length+".txt",
        selectedCases[1]);
    writeNNOutputFile("ELS2002_NNCoded_Test_Noise_"+noise.length+".txt",
        selectedCases[1]);
    writeNoisyVariableInventory("ELS2002_NoisyVariables_"+noise.length+".txt");

    noise = generateNoise(noise,5); // 25 input vars
    performDummyCoding();
    writeOutputFile("ELS2002_DummyCoded_Test_Noise_"+noise.length+".txt",
        selectedCases[1]);
    writeNNOutputFile("ELS2002_NNCoded_Test_Noise_"+noise.length+".txt",
        selectedCases[1]);
    writeNoisyVariableInventory("ELS2002_NoisyVariables_"+noise.length+".txt");

    System.out.println("Finished");
}


/**
 * Writes out a text file containing the variable index and name of the variables
 * tagged as noise
 * @param path The path to the file of noisy variables
 */
public static void writeNoisyVariableInventory(String path) {
    System.out.println("Writing Noisy Variable Inventory - "+path);
    File f = new File(path);
    try {
        FileWriter fw = new FileWriter(f);
        BufferedWriter bw = new BufferedWriter(fw);
        for (int i=0;i<noise.length;i++) {
            bw.write(noise[i]+","+inputNames[i]);
            bw.newLine();
        }
        bw.flush();
```

```
        bw.close();

    } catch (java.io.IOException e) {
        System.out.println("Error writing noisy variable inventory - "+
            e.getMessage());
    }

}

/**
 * Produces an array consisting of the entries in the <code>existingNoise</code>
 * array plus
 * <code>howMany</code> additional entries. Used to randomly select larger and larger
 * pools
 * of input variables so that we can generate noisy test files with missing entries
 * to test the robustness of the Regression Equation and the Neural Network
 *
 * @param existingNoise An array of already selected variables indexes [0,102]
 * @param howMany How many additional variables to select
 * @return A new array containing the original array plus <code>howMany</code>
 * additional variables indices
 */
public static int[] generateNoise(int[]existingNoise, int howMany) {
    System.out.println("Selecting "+(existingNoise.length+howMany)+" ("+howMany
            +" new) input variables to flag as missing.");
    int[] newNoise = new int[existingNoise.length+howMany];
    for (int i=0;i<existingNoise.length;i++) {
        newNoise[i] = existingNoise[i];
    }
    Random random = new Random(System.currentTimeMillis()); //Initialize a pseudo-
                                                            //random number generator
    for (int i=0;i<howMany;i++) {
        boolean ok = false;
        int variableIndex = 0;
        while (!ok) {
            ok = true; // Assume we're good, unless we picked a dupe index
            variableIndex = random.nextInt(inputFeatures.length); // Select a random
                                                                  // index into the
                                                                  // input vars
            // Check to see that we haven't already selected this one.
            for (int j=0;j<existingNoise.length+i;j++) {
                // Loop through the variable indices selected so far
                if (newNoise[j]==variableIndex) {
                    // Found a dupe.
                    ok = false;
                    break;
                }
            }
        }
        newNoise[existingNoise.length+i] = variableIndex;
    }
    return newNoise;
}

/**
 * Writes an array of selected cases to an output file. The first line contains the
 * dummy variable
 * names and the remaining lines contain comma separated list of dummy coded values
 * as well as the
 * target math and reading scores, scaled to the range 0-1
 * @param path The path to the output file
 */
public static void writeOutputFile(String path, int[] cases) {
    System.out.print("Writing output file - "+path);
    try {
        File f = new File(path);
        FileWriter fw = new FileWriter(f);
        BufferedWriter bw = new BufferedWriter(fw);
        // First write out the variable names
        bw.write("MathScore,ReadingScore");
        for (int i=0;i<inputNames.length;i++) {
```

```
            for (int j=0;j<inputFeatures[i].length-1;j++) { // length-1 for C-1
                                                            // dummy coding
                bw.write(","+inputNames[i]+"_D"+(j+1));
            }
        }
        bw.newLine();
        for (int i=0;i<cases.length;i++) {
            int caseIndex = cases[i];
            String outLine = mathTargets[caseIndex]+","+readingTargets[caseIndex];
            for (int j=0;j<dummyCoding[caseIndex].length;j++) {
                outLine+=","+dummyCoding[caseIndex][j];
            }
            bw.write(outLine);
            bw.newLine();
        }
        bw.flush();
        bw.close();
    } catch (java.io.IOException e) {
        System.out.println("Error writing output file - "+e.getMessage());
    }
    System.out.println(" - "+cases.length+" cases");
}

 /**
 * Writes an array of selected cases to an output file for use by the NN.
 * Each line contains the integer responses to the ELS2002 Survey questions
 * for the 103 variables under study
 *
 * @param path The path to the output file
 */
public static void writeNNOutputFile(String path, int[] cases) {
    System.out.print("Writing Neural Net output file - "+path);
    try {
        File f = new File(path);
        FileWriter fw = new FileWriter(f);
        BufferedWriter bw = new BufferedWriter(fw);

        for (int i=0;i<cases.length;i++) {
            int caseIndex = cases[i];
            String outLine = mathTargets[caseIndex]+","+readingTargets[caseIndex];
            for (int j=0;j<NNCoding[caseIndex].length;j++) {
                outLine+=","+NNCoding[caseIndex][j];
            }
            bw.write(outLine);
            bw.newLine();
        }
        bw.flush();
        bw.close();
    } catch (java.io.IOException e) {
        System.out.println("Error writing output file - "+e.getMessage());
    }
    System.out.println(" - "+cases.length+" cases");
}

/**
 * Checks to make sure that there are no duplicates in the arrays case[0] and
 *cases[1]
 * @param cases A 2 dimensional array of selected cases.
 * @return true if there are no duplicate cases, false otherwise
 */
public static boolean performCrossCheck(int[][] cases) {
    System.out.println("Performing cross check validation of the selected cases");
    boolean ok = true; // Assume we're fine until we find a duplicate.

    for (int i=0;i<cases[0].length;i++) {
        for (int j=0;j<cases[1].length;j++) {
            if (cases[0][i]==cases[1][j]) {
                // We found a duplicate, bail out.
                return false;
            }
        }
```

```
    }
    return ok;
}

/**
 * Generates 2 lists of case #'s. One for training the NN/Regresion Equation and one
 * for
 * testing the NN/Regression equation. See comments in the method for details on the
 * selection
 * algorithm. Essentially we want to mutually exclusive lists and we want to select
 * cases
 * that
 * will result in all dummy variables being used.
 */
public static int[][] performCaseSelection() {
    System.out.println("Performing case selection");
    Random random = new Random(System.currentTimeMillis()); //Initialize a pseudo-
                                                            //random number generator

    //The final selection of cases in two lists of 1536 (10%) samples
    //In actuality, after cleaning we will only have two 1534 case samples
    int[][] selectedCases = new int[2][];
    selectedCases[0]=new int[1534]; // The training data
    selectedCases[1]=new int[1534]; // The test data

    int[] tempSelectedCases = new int[2*1534]; // 20% of our population (10%
                                               // training, 10% test)
    int numSelectedCases = 0;

    // Keep track of which dummy's we've used in this iteration
    ArrayList usedDummyVariables = new ArrayList();
    //We need to keep going until we select 20% the total cases
    System.out.print("Cases selected so far:");
    while (numSelectedCases < (2*1534)) {
        // Select a dummy variable at random until we've selected each dummy variable
        // one, then start over.
        boolean ok = false;
        int selectedDummyVar = 0;
        while (!ok) {
            selectedDummyVar = random.nextInt(categoryCount); // Select a random in
                                                              // from
                                                              // 0-(categoryCount-1)
            //Assume the selected category is OK, unless we found we've used it
            ok = true;
            // Check to see if we've exhausted all possible categories. If so, clear
            // the used category arraylist and start over.
            if (usedDummyVariables.size()==categoryCount) {
                usedDummyVariables.clear();
            }
            // Did we already use this dummy variable.
            for (int i=0;i<usedDummyVariables.size();i++) {
                int dummyIdx = ((Integer)usedDummyVariables.get(i)).intValue();
                if (dummyIdx==selectedDummyVar) {
                    ok = false; // Have to select another one
                    break; // exit this loop;
                }
            }
        }
        // Add this selected dummy to the list of used dummy variables
        usedDummyVariables.add(new Integer(selectedDummyVar));

        // Now that we've selected the category, randomly select one of the cases
        // that has a 1 for this category. If none of the cases had a 1 for this
        // category
        // or if there are no remaining cases to choose from for this category, just
        // continue
        ArrayList dummyCases = (ArrayList)dummyFreqs.get(selectedDummyVar);
        if (dummyCases.size()>0) {
            // Randomly select one of the remaining cases
            int selectedCaseIndex = random.nextInt(dummyCases.size());
            int selectedCase =
```

```
                    ((Integer)dummyCases.get(selectedCaseIndex)).intValue();
                tempSelectedCases[numSelectedCases] = selectedCase;
                numSelectedCases++;
                if (numSelectedCases %500==0) System.out.print(" "+numSelectedCases);
                // Now, remove this case from every array list in which it appears
                // so that we don't select this case again.
                for (int i=0;i<dummyFreqs.size();i++) {
                    ArrayList cases = (ArrayList)dummyFreqs.get(i);
                    for (int j=0;j<cases.size();j++) {
                        int caseIndex = ((Integer)cases.get(j)).intValue();
                        if (caseIndex==selectedCase) {
                            cases.remove(j);
                            break;
                        }
                    }
                }
            }

        }
        System.out.println();
        System.out.println("Selected a total of "+numSelectedCases+" cases");

        // Finally, randomly split the (2*1534) cases into 2 lists of 1534 & 1534 cases
        // respectively and return the resulting 2 dimensional array
        ArrayList selectedCasesArrayList = new ArrayList();
        // Temporarily store the 2*1534 cases in an ArrayList for easier handling
        for (int i=0;i<tempSelectedCases.length;i++) {
            selectedCasesArrayList.add(new Integer(tempSelectedCases[i]));
        }
        // Randomly select 1534 cases, removing each selected case from the array list as
        // it's added to the
        // selectedCases array. First, select the 10% training sample
        for (int i=0;i<(1534);i++) {
            int selectedCaseIndex = random.nextInt(selectedCasesArrayList.size());
            int selectedCase =
                ((Integer)selectedCasesArrayList.get(selectedCaseIndex)).intValue();
            selectedCases[0][i]=selectedCase;
            selectedCasesArrayList.remove(selectedCaseIndex);
        }
        // Now, there should be only 1534 items left in the selectedCasesArrayList, just
        // dump them into the second list of selected cases
        for (int i=0;i<1534;i++) {
            int selectedCase = ((Integer)selectedCasesArrayList.get(i)).intValue();
            selectedCases[1][i]=selectedCase;
        }

        return selectedCases;
    }

    /**
     *  Determines which cases have a 1 for each dummy variable and populates an
     *  ArrayList
     *  representing the dummy variables with another ArrayList containing the case #'s.
     */
    public static void performDummyFrequencies() {
        System.out.println("Calculating dummay variable frequencies");
        // Create the ArrayList of ArrayLists
        dummyFreqs = new ArrayList();
        for (int i=0;i<categoryCount;i++) {
            dummyFreqs.add(new ArrayList());
        }
        // Loop over all dummycoded test cases and then loop over all values for each
        // case
        // and add this case # to each DummyFreqs ArrayList where this case has a value
        // of 1
        for (int i=0;i<dummyCoding.length;i++) {
            // For each dummy case, determine which categories are represented for each
            // input
            // variable. For categories 1-(C-1) this is easy, just check the dummy
            // variable
            // for a value of 1, for category C, though, we have to check all dummies
```

```
        // related to that input and see if they are all 0.
        int cMinusOneIndex = 0;
        int categoryIndex = 0;

        for (int j=0;j<inputFeatures.length;j++) { // Loop over all input features
                                                   // (input vars)
            boolean categoryC = true; // flag that tells us if dummies 1-(C-1)
                                      // were all 0
            for (int k=0;k<inputFeatures[j].length-1;k++) { // Look for 1's in
                                                            // categories
                                                            // 1-(C-1)
                if (dummyCoding[i][cMinusOneIndex]==1) {
                    categoryC = false;
                    ArrayList cases = (ArrayList)dummyFreqs.get(categoryIndex);
                    cases.add(new Integer(i));
                }
                cMinusOneIndex++;
                categoryIndex++;
            }
            // Now check category C, which is denoted by category dummies 1-(C-1)
            // being 0
            if (categoryC) {
                // Dummies 1-(C-1) were all 0, so it was category C
                ArrayList cases = (ArrayList)dummyFreqs.get(categoryIndex);
                cases.add(new Integer(i));
            }
            categoryIndex++; // Advance the category index
        }
    }
}

/**
 * Encodes the cases from ArrayList inputCases into a two dimensional
 * array dummyCoding where the 1st dimension represents the case # and the
 * 2nd dimension represents each dummy variable
 * Also extracts the math and reading scores, scales them to the range 0-1,
 * and stored them in the arrays mathTargets and readingTargets
 */
public static void performDummyCoding() {
    System.out.println("Performing dummy coding for "+inputCases.size()+
        " input cases (Noisy inputs = "+noise.length+")");
    String noisyVars = "";
    for (int i=0;i<inputCases.size();i++) {
        noisyVars = "";
        String inputCase = (String)inputCases.get(i);
        String[] inputValues = inputCase.split("\t");
        int dummyIndex = 0;
        int inputValueIndex = 0;
        for (int j=1;j<inputValues.length;j++) { // start at j=1 to skip student id
            if (j==READING_SCORE_INDEX || j==MATH_SCORE_INDEX) {
                // Store the math and reading score values
                //in the mathTargets and readingTargets arrays

                //We're scaling the math and reading scores, which are in the range
                //0-100
                //down to the range 0-1. So, we simply extract the decimal point from
                //the score, e.g. 45.33, and put it in front, e.g. .4533
                String[] scoreParts = inputValues[j].split("\\.");
                String scaledScore = ".";
                for (int k=0;k<scoreParts.length;k++) {
                    scaledScore+=scoreParts[k];
                }
                if (j==READING_SCORE_INDEX) {
                    // Store the reading score target
                    readingTargets[i] = scaledScore;
                } else {
                    // it must be a math score
                    mathTargets[i] = scaledScore;
                }
            } else {
                //Figure out which dummy variable gets the 1 and set the others to 0
```

```
                //If this input value is not in the list of allowable values, set
                //all dummy variables to 0
                int intValue = Integer.parseInt(inputValues[j]);

                // But check to see if we have generated any noise, and if so, if
                // this
                // input variable has been flagged as noise, mark it as missing
                // (which will code it as all 0's)
                for (int l=0;l<noise.length;l++) {
                    if (noise[l]==inputValueIndex) {
                        // This input var has been marked as noise, flag is as
                        // missing.
                        intValue = -9;
                        noisyVars += inputValueIndex+" ";
                        break;
                    }
                }

                // Store the quantative value for use by the NN
                NNCoding[i][inputValueIndex] = intValue;

                // We dummy coding C-1 categories, so we check to see if the value
                // is one of the categories from 1 to C-1 and set that dummy to 1
                // If it's the last Category, all dummies are set to 0
                for (int k=0;k<inputFeatures[inputValueIndex].length-1;k++) {
                    if (intValue==inputFeatures[inputValueIndex][k]) {
                        dummyCoding[i][dummyIndex]=1;
                    } else {
                        dummyCoding[i][dummyIndex]=0;
                    }
                    dummyIndex++;
                }
                inputValueIndex++;
            }
        }
    }
    System.out.println("Noisy Variable Indicies = "+noisyVars);
}

/**
 * Read the full data set into a vector for processing.
 * During the load, the input variables are checked and cases that have invalid
 * inputs (missing, multiple answer, etc) are dicarded. This ensures that the
 * training and test data are pure.
 *
 * @param path The path to the full data set.
 */
public static void loadInputCases(String path) {
    System.out.println("Loading input cases from "+path);
    try {
        File f = new File(path);
        FileReader fr = new FileReader(f);
        BufferedReader br = new BufferedReader(fr);
        String inputCase = br.readLine();
        int discardedCases = 0;
        while (inputCase!=null) {
            String[] inputValues = inputCase.split("\t");
            boolean discardCase = false; // Assume the case is OK until we find
                                         // otherwise
            int inputValueIndex = 0;
            for (int i=1;i<inputValues.length;i++) { // Start at 1 to skip student id
                // Skip the math & reading scores
                if (i==MATH_SCORE_INDEX || i==READING_SCORE_INDEX) continue;
                boolean missingValue = true;
                int intValue = Integer.parseInt(inputValues[i]);
                for (int j=0;j<inputFeatures[inputValueIndex].length;j++) {
                    if (intValue==inputFeatures[inputValueIndex][j]) {
                        missingValue = false;
                        break; // Exit the loop
                    }
                }
```

```
                if (missingValue) {
                    discardCase = true;
                    break; // Exit the loop
                }
                inputValueIndex++;
            }
            if (!discardCase) {
                inputCases.add(inputCase);
            } else {
                discardedCases++;
            }
            inputCase = br.readLine();
        }
        br.close();
        System.out.println("Discarded "+discardedCases+" cases");
    } catch (java.io.IOException e) {
        System.out.println("Error reading the input cases - "+e.getMessage());
    }
}

/**
 * Initializes the arrays containing the root names for the dummy variables
 * as well as the valid values for each dummy variable
 */
public static void setupInputFeatures() {

    //BYFCOMP
    inputNames[0]="BYFCOMP";
    inputFeatures[0]=new int[]{1,2,3,4,5,6,7,8,9};
    //PARED
    inputNames[1]="PARED";
    inputFeatures[1]=new int[]{1,2,3,4,5,6,7,8};
    //MOTHED
    inputNames[2]="MOTHED";
    inputFeatures[2]=new int[]{1,2,3,4,5,6,7,8};
    //FATHED
    inputNames[3]="FATHED";
    inputFeatures[3]=new int[]{1,2,3,4,5,6,7,8};
    //SES1QU
    inputNames[4]="SES1QU";
    inputFeatures[4]=new int[]{1,2,3,4};
    //STEXPECT
    inputNames[5]="STEXPECT";
    inputFeatures[5]=new int[]{-1,1,2,3,4,5,6,7};
    //BYBASEBL
    inputNames[6]="BYBASEBL";
    inputFeatures[6]=new int[]{1,2,3,4,5};
    //BYSOFTBL
    inputNames[7]="BYSOFTBL";
    inputFeatures[7]=new int[]{1,2,3,4,5};
    //BYBASKTBL
    inputNames[8]="BYBASKTBL";
    inputFeatures[8]=new int[]{1,2,3,4,5};
    //BYFOOTBL
    inputNames[9]="BYFOOTBL";
    inputFeatures[9]=new int[]{1,2,3,4,5};
    //BYSOCCER
    inputNames[10]="BYSOCCER";
    inputFeatures[10]=new int[]{1,2,3,4,5};
    //BYTEAMSP
    inputNames[11]="BYTEAMSP";
    inputFeatures[11]=new int[]{1,2,3,4,5};
    //BYSOLOSP
    inputNames[12]="BYSOLOSP";
    inputFeatures[12]=new int[]{1,2,3,4,5};
    //BYSCTRL
    inputNames[13]="BYSCTRL";
    inputFeatures[13]=new int[]{1,2,3};
    //BYURBAN
    inputNames[14]="BYURBAN";
    inputFeatures[14]=new int[]{1,2,3};
```

```
//BYREGION
inputNames[15]="BYREGION";
inputFeatures[15]=new int[]{1,2,3,4};
//BYS20A
inputNames[16]="BYS20A";
inputFeatures[16]=new int[]{1,2,3,4};
//BYS20B
inputNames[17]="BYS20B";
inputFeatures[17]=new int[]{1,2,3,4};
//BYS20C
inputNames[18]="BYS20C";
inputFeatures[18]=new int[]{1,2,3,4};
//BYS20E
inputNames[19]="BYS20E";
inputFeatures[19]=new int[]{1,2,3,4};
//BYS20F
inputNames[20]="BYS20F";
inputFeatures[20]=new int[]{1,2,3,4};
//BYS20G
inputNames[21]="BYS20G";
inputFeatures[21]=new int[]{1,2,3,4};
//BYS20J
inputNames[22]="BYS20J";
inputFeatures[22]=new int[]{1,2,3,4};
//BYS20M
inputNames[23]="BYS20M";
inputFeatures[23]=new int[]{1,2,3,4};
//BYS20N
inputNames[24]="BYS20N";
inputFeatures[24]=new int[]{1,2,3,4};
//BYS21A
inputNames[25]="BYS21A";
inputFeatures[25]=new int[]{1,2,3,4};
//BYS21B
inputNames[26]="BYS21B";
inputFeatures[26]=new int[]{1,2,3,4};
//BYS21C
inputNames[27]="BYS21C";
inputFeatures[27]=new int[]{1,2,3,4};
//BYS21D
inputNames[28]="BYS21D";
inputFeatures[28]=new int[]{1,2,3,4};
//BYS21E
inputNames[29]="BYS21E";
inputFeatures[29]=new int[]{1,2,3,4};
//BYS22A
inputNames[30]="BYS22A";
inputFeatures[30]=new int[]{1,2,3};
//BYS22B
inputNames[31]="BYS22B";
inputFeatures[31]=new int[]{1,2,3};
//BYS22C
inputNames[32]="BYS22C";
inputFeatures[32]=new int[]{1,2,3};
//BYS22D
inputNames[33]="BYS22D";
inputFeatures[33]=new int[]{1,2,3};
//BYS22E
inputNames[34]="BYS22E";
inputFeatures[34]=new int[]{1,2,3};
//BYS22F
inputNames[35]="BYS22F";
inputFeatures[35]=new int[]{1,2,3};
//BYS22G
inputNames[36]="BYS22G";
inputFeatures[36]=new int[]{1,2,3};
//BYS22H
inputNames[37]="BYS22H";
inputFeatures[37]=new int[]{1,2,3};
//BYS24B
inputNames[38]="BYS24B";
```

```
inputFeatures[38]=new int[]{1,2,3,4,5};
//BYS26
inputNames[39]="BYS26";
inputFeatures[39]=new int[]{1,2,3};
//BYS27A
inputNames[40]="BYS27A";
inputFeatures[40]=new int[]{1,2,3,4};
//BYS27B
inputNames[41]="BYS27B";
inputFeatures[41]=new int[]{1,2,3,4};
//BYS27C
inputNames[42]="BYS27C";
inputFeatures[42]=new int[]{1,2,3,4};
//BYS27D
inputNames[43]="BYS27D";
inputFeatures[43]=new int[]{1,2,3,4};
//BYS27E
inputNames[44]="BYS27E";
inputFeatures[44]=new int[]{1,2,3,4};
//BYS27F
inputNames[45]="BYS27F";
inputFeatures[45]=new int[]{1,2,3,4};
//BYS27G
inputNames[46]="BYS27G";
inputFeatures[46]=new int[]{1,2,3,4};
//BYS27H
inputNames[47]="BYS27H";
inputFeatures[47]=new int[]{1,2,3,4};
//BYS27I
inputNames[48]="BYS27I";
inputFeatures[48]=new int[]{1,2,3,4};
//BYS28
inputNames[49]="BYS28";
inputFeatures[49]=new int[]{-1,1,2,3};
//BYS34A
inputNames[50]="BYS34A";
inputFeatures[50]=new int[]{
    -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
//BYS34B
inputNames[51]="BYS34B";
inputFeatures[51]=new int[]{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
    21,22,23,24,25,26};
//BYS35A
inputNames[52]="BYS35A";
inputFeatures[52]=new int[]{
   -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
//BYS35B
inputNames[53]="BYS35B";
inputFeatures[53]=new int[]{
    -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
//BYS36A
inputNames[54]="BYS36A";
inputFeatures[54]=new int[]{
    -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
//BYS36B
inputNames[55]="BYS36B";
inputFeatures[55]=new int[]{
    -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
//BYS37
inputNames[56]="BYS37";
inputFeatures[56]=new int[]{1,2,3,4};
//BYS38B
inputNames[57]="BYS38B";
inputFeatures[57]=new int[]{1,2,3,4};
//BYS39A
inputNames[58]="BYS39A";
inputFeatures[58]=new int[]{1,2,3};
//BYS39B
inputNames[59]="BYS39B";
inputFeatures[59]=new int[]{1,2,3};
//BYS39C
```

```
inputNames[60]="BYS39C";
inputFeatures[60]=new int[]{1,2,3};
//BYS39D
inputNames[61]="BYS39D";
inputFeatures[61]=new int[]{1,2,3};
//BYS39E
inputNames[62]="BYS39E";
inputFeatures[62]=new int[]{1,2,3};
//BYS39F
inputNames[63]="BYS39F";
inputFeatures[63]=new int[]{1,2,3};
//BYS39G
inputNames[64]="BYS39G";
inputFeatures[64]=new int[]{1,2,3};
//BYS39H
inputNames[65]="BYS39H";
inputFeatures[65]=new int[]{1,2,3};
//BYS41A
inputNames[66]="BYS41A";
inputFeatures[66]=new int[]{-1,0,1};
//BYS41B
inputNames[67]="BYS41B";
inputFeatures[67]=new int[]{0,1};
//BYS41C
inputNames[68]="BYS41C";
inputFeatures[68]=new int[]{0,1};
//BYS41D
inputNames[69]="BYS41D";
inputFeatures[69]=new int[]{-1,0,1};
//BYS41E
inputNames[70]="BYS41E";
inputFeatures[70]=new int[]{0,1};
//BYS41F
inputNames[71]="BYS41F";
inputFeatures[71]=new int[]{-1,0,1};
//BYS41G
inputNames[72]="BYS41G";
inputFeatures[72]=new int[]{-1,0,1};
//BYS41H
inputNames[73]="BYS41H";
inputFeatures[73]=new int[]{-1,0,1};
//BYS41I
inputNames[74]="BYS41I";
inputFeatures[74]=new int[]{-1,0,1};
//BYS42
inputNames[75]="BYS42";
inputFeatures[75]=new int[]{
    -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
//BYS43
inputNames[76]="BYS43";
inputFeatures[76]=new int[]{
    -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
//BYS45A
inputNames[77]="BYS45A";
inputFeatures[77]=new int[]{1,2,3,4,5};
//BYS45B
inputNames[78]="BYS45B";
inputFeatures[78]=new int[]{1,2,3,4,5};
//BYS45C
inputNames[79]="BYS45C";
inputFeatures[79]=new int[]{1,2,3,4,5};
//BYS46A
inputNames[80]="BYS46A";
inputFeatures[80]=new int[]{0,1,2,3,4,5,6};
//BYS46B
inputNames[81]="BYS46B";
inputFeatures[81]=new int[]{0,1,2,3,4,5,6};
//BYS54A
inputNames[82]="BYS54A";
inputFeatures[82]=new int[]{1,2,3};
//BYS54B
```

```
        inputNames[83]="BYS54B";
        inputFeatures[83]=new int[]{-1,1,2,3};
        //BYS54C
        inputNames[84]="BYS54C";
        inputFeatures[84]=new int[]{-1,1,2,3};
        //BYS54D
        inputNames[85]="BYS54D";
        inputFeatures[85]=new int[]{1,2,3};
        //BYS54E
        inputNames[86]="BYS54E";
        inputFeatures[86]=new int[]{-1,1,2,3};
        //BYS54F
        inputNames[87]="BYS54F";
        inputFeatures[87]=new int[]{1,2,3};
        //BYS54G
        inputNames[88]="BYS54G";
        inputFeatures[88]=new int[]{1,2,3};
        //BYS54H
        inputNames[89]="BYS54H";
        inputFeatures[89]=new int[]{1,2,3};
        //BYS54I
        inputNames[90]="BYS54I";
        inputFeatures[90]=new int[]{-1,1,2,3};
        //BYS54J
        inputNames[91]="BYS54J";
        inputFeatures[91]=new int[]{-1,1,2,3};
        //BYS54K
        inputNames[92]="BYS54K";
        inputFeatures[92]=new int[]{-1,1,2,3};
        //BYS54L
        inputNames[93]="BYS54L";
        inputFeatures[93]=new int[]{-1,1,2,3};
        //BYS54N
        inputNames[94]="BYS54N";
        inputFeatures[94]=new int[]{-1,1,2,3};
        //BYS54O
        inputNames[95]="BYS54O";
        inputFeatures[95]=new int[]{1,2,3};
        //BYS57
        inputNames[96]="BYS57";
        inputFeatures[96]=new int[]{-1,1,2,3,4,5};
        //BYS60
        inputNames[97]="BYS60";
        inputFeatures[97]=new int[]{0,1};
        //BYS61
        inputNames[98]="BYS61";
        inputFeatures[98]=new int[]{0,1};
        //BYS66A
        inputNames[99]="BYS66A";
        inputFeatures[99]=new int[]{-1,1,2,3,4,5,6};
        //BYS66B
        inputNames[100]="BYS66B";
        inputFeatures[100]=new int[]{-1,1,2,3,4,5,6};
        //BYS66F
        inputNames[101]="BYS66F";
        inputFeatures[101]=new int[]{-1,1,2,3,4,5,6,7};
        //BYS72
        inputNames[102]="BYS72";
        inputFeatures[102]=new int[]{1,2,3};

    }
}
```

APPENDIX C

Regression Cross Validation Source Listing

```
import java.io.File;
import java.io.FileReader;
import java.io.BufferedReader;
import java.math.BigDecimal;

/**
 * This program reads Dummy Variable names and coefficients from a CSV
 * (comma separated value) file exported from Excel. The variable names and
 * coefficients are the result of running a linear regression on the ELS2002
 * training data.
 *
 */
public class CrossValidator {

    private static int[][]inputFeatures = new int[103][];
    private static String[] inputNames = new String[103];
    private static int QUANT=0; // For quantitatively coded inputs
    private static int CODED=1; // For dummy coded inputs

    private static int dummyCount = 0;
    private static int[][] inputCases;
    private static String[] inputVariableNames;
    private static double[] mathTargets = new double[1534];
    private static double[] readingTargets = new double[1534];

    private static int MATH_SCORE_INDEX = 0; // The location of the math score in the
input
                                             //data
    private static int READING_SCORE_INDEX = 1; // The location of the reading score in
the
                                                // input data

    private static double[] coefficients; // The regression coefficients
    private static String[] coefficientVariables; // The variable each coefficient
applies to

    public static void main(String[] args) {
        // Initialize the input features and dummy variable names
        setupInputFeatures();

        //Calculate the total number of dummy variables
        for (int i=0;i<inputNames.length;i++) {
            dummyCount += inputFeatures[i].length;
        }

        // Set up storage for the dummy coded input cases
        // and target values (reading and math)
        inputCases = new int[1536][dummyCount];
        inputVariableNames = new String[dummyCount];

        // Load the input cases
        loadInputData("ELS2002_DummyCoded_Test.txt");

        // Now load the coefficients for Reading
        loadCoefficients("ELS2002_ReadingCoefficients.csv");

        // Now compute the predicted value for each of the test cases
        // and compute the RMSE, d1, and d2 values
        runModel(READING_SCORE_INDEX);

        // Now load the coefficients for Math
        loadCoefficients("ELS2002_MathCoefficients.csv");

        // Now compute the predicted value for each of the test cases
        // and compute the RMSE, d1, and d2 values
        runModel(MATH_SCORE_INDEX);

    }

    /**
     * Runs all of the input cases through the regression equation
```

```
 * Compares the predicted value to the actual value and computes
 * RMSE and R^2 for the regression equation
 */
public static void runModel(int whichScore) {
    System.out.print("Running the model for ");
    if (whichScore==MATH_SCORE_INDEX) {
        System.out.println("math scores");
    } else {
        System.out.println("reading scores");
    }
    // Compute the mean actual value
    double totalActual = 0;
    for (int i=0;i<inputCases.length;i++) {
        if (whichScore==MATH_SCORE_INDEX) {
            totalActual += mathTargets[i];
        } else {
            totalActual += readingTargets[i];
        }
    }
    double meanActual = totalActual/(double)inputCases.length;

    System.out.println("Mean score = "+meanActual);
    // Compute SST sum( (actual-mean)^2 )
    double SST = 0;
    for (int i=0;i<inputCases.length;i++) {
        double actual = 0;
        if (whichScore==MATH_SCORE_INDEX) {
            actual = mathTargets[i];
        } else {
            actual = readingTargets[i];
        }
        double diff = actual - meanActual;
        SST = SST+(diff*diff);
    }

    double d1Numerator = 0.0;
    double d1Denominator = 0.0;
    double d2Numerator = 0.0;
    double d2Denominator = 0.0;

    // Loop over all input cases
    double SSE = 0.0;
    double SSR = 0.0;
    for (int i=0;i<inputCases.length;i++) {
        double result = computePredictedValue(inputCases[i]);
        double target = 0;
        if (whichScore == MATH_SCORE_INDEX) {
            // We're running the model for math scores
            target = mathTargets[i];
        } else {
            // We're running the model for reading scores
            target = readingTargets[i];
        }

        double diff = new BigDecimal(Double.toString(target-
             result)).setScale(6,BigDecimal.ROUND_HALF_UP).doubleValue();
        SSE = new BigDecimal(Double.toString(SSE+Math.pow(diff,2))).
                      setScale(6,BigDecimal.ROUND_HALF_UP).doubleValue();

        // Calculate the index of agreement numerators and denominators
        d1Numerator += Math.abs(diff); // Sum( |Ypred-Y| )
        d2Numerator += (diff*diff);    // Sum( (|YPred-Y|)^2 )
        d1Denominator += (Math.abs(result-meanActual)+Math.abs(target-meanActual));
        d2Denominator += Math.pow((Math.abs(result-meanActual)+Math.abs(target-
                               meanActual)),2);
    }
    SSR = SST-SSE;
    double RMSE = new BigDecimal(Double.toString(Math.sqrt(new
                      BigDecimal(Double.toString(SSE/(double)inputCases.length)).
                      setScale(6,BigDecimal.ROUND_HALF_UP).doubleValue()))).
                      setScale(6,BigDecimal.ROUND_HALF_UP).doubleValue();
```

```
    double R2 = new BigDecimal(Double.toString(SSR/SST)).
                    setScale(4,BigDecimal.ROUND_HALF_UP).doubleValue();
    System.out.println("RMSE (least squares curve fit) = "+RMSE);

    double d1 = 1.0-(d1Numerator/d1Denominator);
    double d2 = 1.0-(d2Numerator/d2Denominator);
    System.out.println("Indices of Agreement: d1="+d1+", d2="+d2);

}

/**
 * Takes an array of input values and computes the output of the regression
 * equeation and returns that value.
 *
 * @param values An array of input values
 * @return The predicted value of the regression equation
 */
public static double computePredictedValue(int[] values) {
    double result = coefficients[0]; // Start with the regression constant;
    // Start at i=1 since 0 is the regression equation constant
    for (int i=1;i<coefficients.length;i++) {
        String varName = coefficientVariables[i]; // Get the var name for this
        // coefficient find the index of this var in the list of input values
        int j =0;
        for (j=0;j<inputVariableNames.length;j++) {
            if (inputVariableNames[j].equals(varName)) {
                break; // Exit the loop, we found it.
            }
        }
        if (j<inputVariableNames.length) {
            // We found the variable name, compute it's part of the equation and
            // add the value to result
            result = result+(coefficients[i]*(double)values[j]);
        } else {
            // We couldn't find the variable name in the list of input variables
            // This is a problem. Report it and stop
            System.out.println("Unable to locate "+varName+
                " in the list of input variables");
            System.exit(-1);
        }
    }

    return new BigDecimal(Double.toString(result)).
               setScale(6,BigDecimal.ROUND_HALF_UP).doubleValue();
}

/**
 * Reads the coefficients and variables names from a CSV input file
 *
 * @param path The path to the CSV file of coefficients and variable names
 */
public static void loadCoefficients(String path) {
    System.out.println("Reading the coefficients and variable names from "+path);
    try {
        File f = new File(path);
        FileReader fr = new FileReader(f);
        BufferedReader br = new BufferedReader(fr);
        String input = br.readLine();
        // First Count the lines in the file so we can size the coefficient arrays
        int coeffCount = 0;
        while (input!=null) {
            coeffCount++;
            input = br.readLine();
        }
        coefficients = new double[coeffCount];
        coefficientVariables = new String[coeffCount];
        System.out.println(
            "Found "+coeffCount+" coefficients (including the constant)");
        // Move back to the beginning of the file
        br.close();
        fr.close();
```

```
        fr = new FileReader(f);
        br = new BufferedReader(fr);
        // Now load the coefficients and variable names
        coeffCount = 0;
        input = br.readLine();
        while (input!=null) {
            String[] values = input.split(",");
            coefficientVariables[coeffCount] = values[0];
            coefficients[coeffCount] = Double.parseDouble(values[1]);
            coeffCount++;
            input = br.readLine();
        }
        br.close();
    } catch (java.io.IOException e) {
        System.out.println("Error reading coefficients - "+e.getMessage());
    }
}

/**
 * Reads the input file and populates the inputData, mathTarget, and readingTarget
 * arrays
 *
 * @param path The path to the input file
 */
public static void loadInputData(String path) {
    System.out.println("Reading dummy variable names and input cases from "+path);
    try {
        File f = new File(path);
        FileReader fr = new FileReader(f);
        BufferedReader br = new BufferedReader(fr);
        String inputCase = br.readLine(); // Read the variable names
        String[] varNames = inputCase.split(",");
        // Start at i=2 to skip the math and reading score var names
        for (int i=2;i<varNames.length;i++) {
            inputVariableNames[i-2]=varNames[i];
        }

        inputCase = br.readLine();
        int caseCount = 0;
        while (inputCase!=null) {
            String[] inputValues = inputCase.split(",");
            // Start at i=2 to skip the math and reading scores.
            for (int i=2;i<inputValues.length;i++) {
                inputCases[caseCount][i-2] = Integer.parseInt(inputValues[i]);
            }
            // Now grab the math and reading scores
            mathTargets[caseCount] =
                Double.parseDouble(inputValues[MATH_SCORE_INDEX]);
            readingTargets[caseCount] =
                Double.parseDouble(inputValues[READING_SCORE_INDEX]);
            inputCase = br.readLine();
            caseCount++;
        }
        br.close();
        System.out.println("Read "+caseCount+" input cases");
    } catch (java.io.IOException e) {
        System.out.println("Error reading input file - "+e.getMessage());
    }
}

 /**
 * Initializes the arrays containing the root names for the dummy variables
 * as well as the valid values for each dummy variable
 */
public static void setupInputFeatures() {

     int i = 0; // Index into the input arrays

     //BYFCOMP
     inputNames[i]="BYFCOMP";
     inputFeatures[i]=new int[]{1,2,3,4,5,6,7,8,9};
```

```
 i++;
 //PARED
 inputNames[i]="PARED";
 inputFeatures[i]=new int[]{1,2,3,4,5,6,7,8};
 i++;
 //MOTHED
 inputNames[i]="MOTHED";
 inputFeatures[i]=new int[]{1,2,3,4,5,6,7,8};
 i++;
 //FATHED
 inputNames[i]="FATHED";
 inputFeatures[i]=new int[]{1,2,3,4,5,6,7,8};
 i++;
 //SES1QU
 inputNames[i]="SES1QU";
 inputFeatures[i]=new int[]{1,2,3,4};
 i++;
 //STEXPECT
 inputNames[i]="STEXPECT";
 inputFeatures[i]=new int[]{-1,1,2,3,4,5,6,7};
 i++;
 //BYBASEBL
 inputNames[i]="BYBASEBL";
 inputFeatures[i]=new int[]{1,2,3,4,5};
 i++;
//BYSOFTBL
 inputNames[i]="BYSOFTBL";
 inputFeatures[i]=new int[]{1,2,3,4,5};
 i++;
//BYBASKTBL
 inputNames[i]="BYBASKTBL";
 inputFeatures[i]=new int[]{1,2,3,4,5};
 i++;
//BYFOOTBL
 inputNames[i]="BYFOOTBL";
 inputFeatures[i]=new int[]{1,2,3,4,5};
 i++;
//BYSOCCER
 inputNames[i]="BYSOCCER";
 inputFeatures[i]=new int[]{1,2,3,4,5};
 i++;
 //BYTEAMSP
 inputNames[i]="BYTEAMSP";
 inputFeatures[i]=new int[]{1,2,3,4,5};
 i++;
//BYSOLOSP
 inputNames[i]="BYSOLOSP";
 inputFeatures[i]=new int[]{1,2,3,4,5};
 i++;
 //BYSCTRL
 inputNames[i]="BYSCTRL";
 inputFeatures[i]=new int[]{1,2,3};
 i++;
 //BYURBAN
 inputNames[i]="BYURBAN";
 inputFeatures[i]=new int[]{1,2,3};
 i++;
 //BYREGION
 inputNames[i]="BYREGION";
 inputFeatures[i]=new int[]{1,2,3,4};
 i++;
 //BYS20A
 inputNames[i]="BYS20A";
 inputFeatures[i]=new int[]{1,2,3,4};
 inputCodings[i]=QUANT;
 i++;
 //BYS20B
 inputNames[i]="BYS20B";
 inputFeatures[i]=new int[]{1,2,3,4};
 i++;
 //BYS20C
```

```
inputNames[i]="BYS20C";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS20E
inputNames[i]="BYS20E";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS20F
inputNames[i]="BYS20F";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS20G
inputNames[i]="BYS20G";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS20J
inputNames[i]="BYS20J";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS20M
inputNames[i]="BYS20M";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS20N
inputNames[i]="BYS20N";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS21A
inputNames[i]="BYS21A";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS21B
inputNames[i]="BYS21B";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS21C
inputNames[i]="BYS21C";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS21D
inputNames[i]="BYS21D";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS21E
inputNames[i]="BYS21E";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS22A
inputNames[30]="BYS22A";
inputFeatures[30]=new int[]{1,2,3};
i++;
//BYS22B
inputNames[i]="BYS22B";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS22C
inputNames[i]="BYS22C";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS22D
inputNames[i]="BYS22D";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS22E
inputNames[i]="BYS22E";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS22F
inputNames[i]="BYS22F";
inputFeatures[i]=new int[]{1,2,3};
i++;
```

```
//BYS22G
inputNames[i]="BYS22G";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS22H
inputNames[i]="BYS22H";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS24B
inputNames[i]="BYS24B";
inputFeatures[i]=new int[]{1,2,3,4,5};
i++;
//BYS26
inputNames[i]="BYS26";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS27A
inputNames[i]="BYS27A";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS27B
inputNames[i]="BYS27B";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS27C
inputNames[i]="BYS27C";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS27D
inputNames[i]="BYS27D";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS27E
inputNames[i]="BYS27E";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS27F
inputNames[i]="BYS27F";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS27G
inputNames[i]="BYS27G";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS27H
inputNames[i]="BYS27H";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS27I
inputNames[i]="BYS27I";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS28
inputNames[i]="BYS28";
inputFeatures[i]=new int[]{-1,1,2,3};
i++;
//BYS34A
inputNames[i]="BYS34A";
inputFeatures[i]=new int[]{
    -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
i++;
//BYS34B
inputNames[i]="BYS34B";
inputFeatures[i]=new int[]{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
    21,22,23,24,25,26};
i++;
//BYS35A
inputNames[i]="BYS35A";
inputFeatures[i]=new int[]{
    -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
i++;
```

```
//BYS35B
inputNames[i]="BYS35B";
inputFeatures[i]=new int[]{
    -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
i++;
//BYS36A
inputNames[i]="BYS36A";
inputFeatures[i]=new int[]{
    -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
i++;
//BYS36B
inputNames[i]="BYS36B";
inputFeatures[i]=new int[]{
    -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
i++;
//BYS37
inputNames[i]="BYS37";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS38B
inputNames[i]="BYS38B";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS39A
inputNames[i]="BYS39A";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS39B
inputNames[i]="BYS39B";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS39C
inputNames[i]="BYS39C";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS39D
inputNames[i]="BYS39D";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS39E
inputNames[i]="BYS39E";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS39F
inputNames[i]="BYS39F";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS39G
inputNames[i]="BYS39G";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS39H
inputNames[i]="BYS39H";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS41A
inputNames[i]="BYS41A";
inputFeatures[i]=new int[]{-1,0,1};
i++;
//BYS41B
inputNames[i]="BYS41B";
inputFeatures[i]=new int[]{0,1};
i++;
//BYS41C
inputNames[i]="BYS41C";
inputFeatures[i]=new int[]{0,1};
i++;
//BYS41D
inputNames[i]="BYS41D";
inputFeatures[i]=new int[]{-1,0,1};
i++;
```

```
//BYS41E
inputNames[i]="BYS41E";
inputFeatures[i]=new int[]{0,1};
i++;
//BYS41F
inputNames[i]="BYS41F";
inputFeatures[i]=new int[]{-1,0,1};
i++;
//BYS41G
inputNames[i]="BYS41G";
inputFeatures[i]=new int[]{-1,0,1};
i++;
//BYS41H
inputNames[i]="BYS41H";
inputFeatures[i]=new int[]{-1,0,1};
i++;
//BYS41I
inputNames[i]="BYS41I";
inputFeatures[i]=new int[]{-1,0,1};
i++;
//BYS42
inputNames[i]="BYS42";
inputFeatures[i]=new int[]{
    -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
i++;
//BYS43
inputNames[i]="BYS43";
inputFeatures[i]=new int[]{
   -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
i++;
//BYS45A
inputNames[i]="BYS45A";
inputFeatures[i]=new int[]{1,2,3,4,5};
i++;
//BYS45B
inputNames[i]="BYS45B";
inputFeatures[i]=new int[]{1,2,3,4,5};
i++;
//BYS45C
inputNames[i]="BYS45C";
inputFeatures[i]=new int[]{1,2,3,4,5};
i++;
//BYS46A
inputNames[i]="BYS46A";
inputFeatures[i]=new int[]{0,1,2,3,4,5,6};
i++;
//BYS46B
inputNames[i]="BYS46B";
inputFeatures[i]=new int[]{0,1,2,3,4,5,6};
i++;
//BYS54A
inputNames[i]="BYS54A";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS54B
inputNames[i]="BYS54B";
inputFeatures[i]=new int[]{-1,1,2,3};
i++;
//BYS54C
inputNames[i]="BYS54C";
inputFeatures[i]=new int[]{-1,1,2,3};
i++;
//BYS54D
inputNames[i]="BYS54D";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS54E
inputNames[i]="BYS54E";
inputFeatures[i]=new int[]{-1,1,2,3};
i++;
//BYS54F
```

```
            inputNames[i]="BYS54F";
            inputFeatures[i]=new int[]{1,2,3};
            i++;
            //BYS54G
            inputNames[i]="BYS54G";
            inputFeatures[i]=new int[]{1,2,3};
            i++;
            //BYS54H
            inputNames[i]="BYS54H";
            inputFeatures[i]=new int[]{1,2,3};
            i++;
            //BYS54I
            inputNames[i]="BYS54I";
            inputFeatures[i]=new int[]{-1,1,2,3};
            i++;
            //BYS54J
            inputNames[i]="BYS54J";
            inputFeatures[i]=new int[]{-1,1,2,3};
            i++;
            //BYS54K
            inputNames[i]="BYS54K";
            inputFeatures[i]=new int[]{-1,1,2,3};
            i++;
            //BYS54L
            inputNames[i]="BYS54L";
            inputFeatures[i]=new int[]{-1,1,2,3};
            i++;
            //BYS54N
            inputNames[i]="BYS54N";
            inputFeatures[i]=new int[]{-1,1,2,3};
            i++;
            //BYS54O
            inputNames[i]="BYS54O";
            inputFeatures[i]=new int[]{1,2,3};
            i++;
            //BYS57
            inputNames[i]="BYS57";
            inputFeatures[i]=new int[]{-1,1,2,3,4,5};
            i++;
            //BYS60
            inputNames[i]="BYS60";
            inputFeatures[i]=new int[]{0,1};
            i++;
            //BYS61
            inputNames[i]="BYS61";
            inputFeatures[i]=new int[]{0,1};
            i++;
            //BYS66A
            inputNames[i]="BYS66A";
            inputFeatures[i]=new int[]{-1,1,2,3,4,5,6};
            i++;
            //BYS66B
            inputNames[i]="BYS66B";
            inputFeatures[i]=new int[]{-1,1,2,3,4,5,6};
            i++;
            //BYS66F
            inputNames[i]="BYS66F";
            inputFeatures[i]=new int[]{-1,1,2,3,4,5,6,7};
            i++;
            //BYS72
            inputNames[i]="BYS72";
            inputFeatures[i]=new int[]{1,2,3};
            i++;
    }
}
```

APPENDIX D

Neural Network Source Listing

```
import org.joone.engine.*;
import org.joone.engine.learning.TeachingSynapse;
import org.joone.net.NeuralNet;
import org.joone.io.MemoryInputSynapse;
import org.joone.io.MemoryOutputSynapse;
import org.joone.helpers.factory.JooneTools;

import java.io.*;
import java.util.ArrayList;
import java.util.Random;

/**
 * Either loads from disk or sets up a 3 layer, feed forward, supervised learning neural
 * network utilizing the resilient back-propagation learning algorithm.
 *
 * If in training mode, reads the contents of the training file, sets aside 10% of the
 * training data for
 * validation purposes, and begins 500 training epochs during which 90% of the training
 * cases are presented to the network in order to adapt the synaptic weights.
 *
 * Every 10th epoch the NeuralNetListener object makes a copy of the NeuralNet, runs the
 * reserved 10% of the
 * training data through the network and computes the RMSE. If the RMSE is better than
 * the last validation,
 * then this copy of the network is set aside as the best network so far and training
 * continues. At the end
 * of 500 training epochs, save a copy of the best Neural Network.
 *
 * If not in training mode, load the neural network from disk, load the test data and run
 * the neural net.
 * Compute the RMSE, and Wilmott's indices of agreement d1, and d2.
 *
 */
public class ELS2002NeuralNet {

    private static int[][] inputFeatures = new int[103][];
    private static String[] inputNames = new String[103];
    private static int QUANT=0;
    private static int CODED=1;

    public static int MATH_SCORE = 0; //index to the math score in the input data
    public static int READING_SCORE = 1; // index to the reading score in the input data

    public static String inputColumns = "";

    private static NeuralNet neuralNet;

    private static int totalInputNeurons = 0;

    private static double[][] inputCases;      // The full set of input cases
    private static double[][] inputTargets;    // The full set of input target values
    private static double[][] trainingCases;   // Pulled from inputCases, used to train
                                               // the network
    public static double[][] testCases;        // Pulled from inputCases, used to test
                                               // the fitness of the network
    private static double[][] trainingTargets; // Pulled from inputTargets, used to train
                                               // the network
    public static double[][] testTargets;      // Pulled from inputTargets, used to test
                                               // the fitness of the network

    public static void main(String[] args) {

        // Populate the arrays of input variable names, valid responses, and coding
        setupInputFeatures();

        // figure out how many input neurons we'll have by examing the input variables
        totalInputNeurons = 0;
        for (int i=0;i<inputFeatures.length;i++) {
            totalInputNeurons += inputFeatures[i].length;
        }
```

```
// Create a string of the form "1,2,3,4,5...n" where n=totalInputNeurons, used
// to identify the
// columns that will be used as input to the neural network
for (int i=0;i<totalInputNeurons;i++) {
    if (i==0)
        inputColumns += (i+1);
    else
        inputColumns += ","+(i+1);
}

// Try to load a saved neural network from disk
neuralNet = loadNet();

// If the net wasn't found, create a new one
if (neuralNet==null) {
    // Construct a new neural net and train it
    LinearLayer inputLayer = new LinearLayer();
    SigmoidLayer hiddenLayer = new SigmoidLayer();
    SigmoidLayer outputLayer = new SigmoidLayer();

    System.out.println("Creating Input Layer with "+totalInputNeurons+
        " input neurons");
    inputLayer.setRows(totalInputNeurons);
    hiddenLayer.setRows(30);
    outputLayer.setRows(2);

    FullSynapse synapse_IH = new FullSynapse();
    FullSynapse synapse_HO = new FullSynapse();

    inputLayer.addOutputSynapse(synapse_IH);
    hiddenLayer.addInputSynapse(synapse_IH);
    hiddenLayer.addOutputSynapse(synapse_HO);
    outputLayer.addInputSynapse(synapse_HO);

    neuralNet = new NeuralNet();
    neuralNet.addLayer(inputLayer,NeuralNet.INPUT_LAYER);
    neuralNet.addLayer(hiddenLayer,NeuralNet.HIDDEN_LAYER);
    neuralNet.addLayer(outputLayer,NeuralNet.OUTPUT_LAYER);

    MemoryInputSynapse inputStream = new MemoryInputSynapse();
    inputStream.setInputDimension(totalInputNeurons);
    inputStream.setAdvancedColumnSelector(inputColumns);

    inputCases =getInputData("ELS2002_NNCoded_Training.txt");
    inputTargets = getTargetData("ELS2002_NNCoded_Training.txt");

    // Reserve 10% of the input cases to be used to test the accuracy of the net
    splitInputCases(inputCases, 153);

    inputStream.setInputArray(trainingCases);
    neuralNet.getInputLayer().addInputSynapse(inputStream);

    TeachingSynapse trainer = new TeachingSynapse();
    MemoryInputSynapse samples = new MemoryInputSynapse();
    samples.setOutputDimension(2);
    samples.setInputArray(trainingTargets);

    trainer.setDesired(samples);
    samples.setAdvancedColumnSelector("1,2");
    neuralNet.setTeacher(trainer);
    neuralNet.getOutputLayer().addOutputSynapse(trainer);
}

// Decide if we're training, or running cross-validation
boolean training = false;

if (training) {
    //These three lines are settings for RProp
    neuralNet.getMonitor().addLearner(0, "org.joone.engine.RpropLearner");
    neuralNet.getMonitor().setLearningMode(0);
    neuralNet.getMonitor().setLearningRate(1.0); // For the RpropLearner
```

```
    neuralNet.getMonitor().setBatchSize(trainingCases.length);
    neuralNet.getMonitor().addNeuralNetListener(
        new MyNeuralNetListener(neuralNet));
    neuralNet.getMonitor().setTrainingPatterns(trainingCases.length);
    neuralNet.getMonitor().setTotCicles(500);
    neuralNet.getMonitor().setLearning(true);
    neuralNet.go();
} else {

    // We're performing cross-validation
    String inputPath = "ELS2002_NNCoded_Test.txt";
    neuralNet.getMonitor().addNeuralNetListener(
        new MyNeuralNetListener(neuralNet));
    neuralNet.getInputLayer().removeAllInputs();
    MemoryInputSynapse testData = new MemoryInputSynapse();
    testData.setFirstRow(1);
    testData.setAdvancedColumnSelector(inputColumns);
    neuralNet.getInputLayer().addInputSynapse(testData);
    inputCases = getInputData(inputPath);
    testData.setInputArray(inputCases);
    neuralNet.getOutputLayer().removeAllOutputs();
    MemoryOutputSynapse testResults = new MemoryOutputSynapse();
    neuralNet.getOutputLayer().addOutputSynapse(testResults);
    neuralNet.getMonitor().setTotCicles(1);
    neuralNet.getMonitor().setTrainingPatterns(inputCases.length);
    neuralNet.getMonitor().setLearning(false);

    double[][] outputData = getTargetData(inputPath);

    neuralNet.go();

    File outputFile = new File("ReadingPredictions.txt");
    File outputFile2 = new File("MathPredictions.txt");

    double Math_SSResidual = 0;
    double Math_SSRegression = 0;
    double Reading_SSResidual = 0;
    double Reading_SSRegression = 0;

    // Compute the average reading score to be used in the calculation of R^2
    double totalReading = 0;
    double totalMath = 0;
    for (int i=0;i<outputData.length;i++) {
        totalReading+=outputData[i][READING_SCORE];
        totalMath+=outputData[i][MATH_SCORE];
    }
    double avgReading = totalReading/(double)outputData.length;
    double avgMath = totalMath/(double)outputData.length;
    System.out.println("Computed Avergage Reading Score = "+avgReading);
    System.out.println("Computed Avergage Math Score    = "+avgMath);

    double Math_SSTotal = 0;
    double Reading_SSTotal = 0;
    for (int i=0;i<outputData.length;i++) {
        double reading_diff = outputData[i][READING_SCORE]-avgReading;
        Reading_SSTotal += (reading_diff*reading_diff);

        double math_diff = outputData[i][MATH_SCORE]-avgMath;
        Math_SSTotal += (math_diff*math_diff);
    }

    double Math_d1Numerator = 0.0;
    double Math_d1Denominator = 0.0;
    double Math_d2Numerator = 0.0;
    double Math_d2Denominator = 0.0;

    double Reading_d1Numerator = 0.0;
    double Reading_d1Denominator = 0.0;
    double Reading_d2Numerator = 0.0;
    double Reading_d2Denominator = 0.0;
```

```
try {
    FileWriter fw = new FileWriter(outputFile);
    FileWriter fw2 = new FileWriter(outputFile2);
    BufferedWriter bw = new BufferedWriter(fw);
    BufferedWriter bw2 = new BufferedWriter(fw2);

    bw.write("Actual Reading Score,Predicted Reading Score");
    bw.newLine();
    bw2.write("Actual Math Score,Predicted Math Score");
    bw2.newLine();

    for (int i=0;i<inputCases.length;i++) {
        double[] results = testResults.getNextPattern();

        double math_err = results[MATH_SCORE]-outputData[i][MATH_SCORE];
        double reading_err = results[READING_SCORE]-
            outputData[i][READING_SCORE];

        Math_SSResidual += (math_err*math_err);
        Reading_SSResidual += (reading_err*reading_err);

        bw.write(outputData[i][READING_SCORE]+","+results[READING_SCORE]);
        bw.newLine();
        bw2.write(outputData[i][MATH_SCORE]+","+results[MATH_SCORE]);
        bw2.newLine();

        // Calculate the index of agreement numerators and denominators
        Math_d1Numerator += Math.abs(math_err);      // Sum( |Ypred-Y| )
        Math_d2Numerator += (math_err*math_err);     // Sum( (|YPred-Y|)^2 )
        Math_d1Denominator += (Math.abs(results[MATH_SCORE]-
            avgMath)+Math.abs(outputData[i][MATH_SCORE]-avgMath));
        Math_d2Denominator += Math.pow((Math.abs(results[MATH_SCORE]-
            avgMath)+Math.abs(outputData[i][MATH_SCORE]-avgMath)),2);

        Reading_d1Numerator += Math.abs(reading_err);     // Sum( |Ypred-Y| )
        Reading_d2Numerator += (reading_err*reading_err); // Sum( (|YPred-
                                                          // Y|)^2 )
        Reading_d1Denominator += (Math.abs(results[READING_SCORE]-
            avgReading)+Math.abs(outputData[i][READING_SCORE]-avgReading));
        Reading_d2Denominator += Math.pow((Math.abs(results[READING_SCORE]-
            avgReading)+Math.abs(outputData[i][READING_SCORE]-
            avgReading)),2);
    }
    Math_SSRegression = Math_SSTotal-Math_SSResidual;
    Reading_SSRegression = Reading_SSTotal-Reading_SSResidual;

    double math_rmse = Math.sqrt(Math_SSResidual/(double)inputCases.length);
    double reading_rmse =
        Math.sqrt(Reading_SSResidual/(double)inputCases.length);

    System.out.println("Math RMSE = "+math_rmse+" Reading RMSE = "+
        reading_rmse);

    double math_d1 = 1.0-(Math_d1Numerator/Math_d1Denominator);
    double math_d2 = 1.0-(Math_d2Numerator/Math_d2Denominator);

    double reading_d1 = 1.0-(Reading_d1Numerator/Reading_d1Denominator);
    double reading_d2 = 1.0-(Reading_d2Numerator/Reading_d2Denominator);

    System.out.println("Math Indices of Agreement:    d1="+math_d1+
        ", d2="+math_d2);
    System.out.println("Reading Indices of Agreement: d1="+reading_d1+
        ", d2="+reading_d2);

    bw.flush();
    bw.close();
    bw2.flush();
    bw2.close();

} catch (java.io.IOException e) {
    System.out.println("Error writing predicted results - "+e.getMessage());
```

```
        }

    }

}

/**
 * Splits the complete list of input cases into a set for test purposed and a set
 * for training purposes. Periodically the net will be tested with the test cases
 * so that we can determine if we're overfitting the net and so that we can preserve
 * the best net even if the best net is found in the middle of the training.
 *
 * @param cases A 2-D array of input cases
 * @param splitCount The number of cases to strip off to be used to test the net
 *
 */
public static void splitInputCases(double[][] cases, int splitCount) {
    System.out.println("Splitting the input cases into "+(cases.length-splitCount)+
        " training and "+splitCount+" test cases");

    ArrayList allCases = new ArrayList();
    // First, dimension the trainingCases and testCases arrays
    trainingCases = new double[cases.length-splitCount][cases[0].length];
    trainingTargets = new double[trainingCases.length][2];
    testCases = new double[splitCount][cases[0].length];
    testTargets = new double[testCases.length][2];

    // Next, copy all of the case indices into the ArrayList
    for (int i=0;i<cases.length;i++) {
        allCases.add(new Integer(i));
    }

    // Now, set up a loop to randomly select case indices from the remaining
    // indicies and populate the test case array
    Random random = new Random(System.currentTimeMillis());
    for (int i=0;i<splitCount;i++) {
        int selectedIndex = random.nextInt(allCases.size());
        int selectedCaseIndex = ((Integer)allCases.get(selectedIndex)).intValue();
        for (int j=0;j<cases[selectedCaseIndex].length;j++) {
            // Copy this case into the array of test cases
            testCases[i][j] = cases[selectedCaseIndex][j];
        }
        testTargets[i][READING_SCORE] =
            inputTargets[selectedCaseIndex][READING_SCORE];
        testTargets[i][MATH_SCORE] = inputTargets[selectedCaseIndex][MATH_SCORE];
        // Now remove this case index from the array list of all cases, so we don't
        // pick it again.
        allCases.remove(selectedIndex);
    }
    // Finally, populate the training case array with the cases that haven't been
    // selected
    for (int i=0;i<allCases.size();i++) {
        int selectedCaseIndex = ((Integer)allCases.get(i)).intValue();
        for (int j=0;j<cases[selectedCaseIndex].length;j++) {
            trainingCases[i][j] = cases[selectedCaseIndex][j];
        }
        trainingTargets[i][READING_SCORE] =
            inputTargets[selectedCaseIndex][READING_SCORE];
        trainingTargets[i][MATH_SCORE] = inputTargets[selectedCaseIndex][MATH_SCORE];
    }
}

/**
 * Used by MyNeuralNetListener to get the test cases, the reserved 10% of the
 * training data used for validation of the neural network training.
 *
 * @return 2-D array of cases in dummy coded format.
 */
public static double[][] getTestCases() {
    return testCases;
}
```

```
/**
 * Used by MyNeuralNetListener to get the targets for the test cases used for
 * validation of the neural network training.
 *
 * @return 2-D array of target values
 */
public static double[][] getTestTargets() {
    return testTargets;
}

/**
 * Used by MyNeuralNetListener to save the best NN once training is complete.
 *
 * @param theNet The NeuralNetwork object to save
 */
public static void saveNet(NeuralNet theNet) {
    try {
        System.out.println("Saving the neural network");
        JooneTools.save(theNet,"MathAndReadingNeuralNet.net");
    } catch (java.io.IOException e) {
        System.out.println("Error saving net - "+e.getMessage());
    }
}

/**
 * Reads a saved NeuralNetwork object from disk.
 *
 * @return A previously trained NeuralNetwork object or null if the file doesn't
 * exist.
 */
public static NeuralNet loadNet() {
    File f = new File("MathAndReadingNeuralNet.net");
    NeuralNet net = null;

    if (!f.exists()) {
        System.out.println("Stored Neural Net not found");
        return null;
    }

    try {
        System.out.println("Loading the neural net from disk");
        net = JooneTools.load("MathAndReadingNeuralNet.net");
    } catch (Exception e) {
        System.out.println("Error loading net - "+e.getMessage());
    }
    return net;
}

/**
 * Parses the target data from the input file. Returns a 2D array [numcases][2]
 * indexed
 * by the cases in the first dimension and containing the target values in the second
 * dimension.
 *
 * @param path Path to the file in the file system
 *
 * @return 2D array of doubles
 */
public static double[][] getTargetData(String path) {
    File f = new File(path);
    // Note - the input data must be read before the target data so that
    // the total number of input cases has been set
    int totalCases = 0;
    double[][] targets = new double[0][0];
    if (f.exists()) {
        try {
            FileReader fr = new FileReader(f);
            BufferedReader br = new BufferedReader(fr);
            String pattern = br.readLine(); // Get the first real row of data
            // Now figure out how many cases we have
```

```
            while (pattern!=null) {
                totalCases++;
                pattern = br.readLine();
            }
            br.close();
            // Now read the data
            fr = new FileReader(f);
            br = new BufferedReader(fr);
            pattern = br.readLine(); // Get the first read row of data
            targets = new double[totalCases][2];
            int rowCount = 0;
            while (pattern!=null) {
                String[] values = pattern.split(",");
                // Values scaled to [0,1]
                targets[rowCount][READING_SCORE] =
                    Double.parseDouble(values[READING_SCORE]);
                targets[rowCount][MATH_SCORE] =
                    Double.parseDouble(values[MATH_SCORE]);
                rowCount++;
                pattern = br.readLine();
            }
            br.close();
        } catch (java.io.IOException e) {
            System.out.println("Error reading target data - "+e.getMessage());
        }
    }
    System.out.println("    Read "+totalCases+" target cases from "+path);
    return targets;
}

/**
 * Reads the input data from an input file and scales the dummy coded values
 * into the range 0, 0.2-0.8
 *
 * @param path Path to the input file in the file system
 * @return  2D array of doubles indexes by case number and input neuron
 */
public static double[][] getInputData(String path) {
    File f = new File(path);
    double[][] activations = new double[0][0];
    int totalInputCases=0;

    if (f.exists()) {
        try {
            FileReader fr = new FileReader(f);
            BufferedReader br = new BufferedReader(fr);
            // First count up the total number of input rows
            String pattern = br.readLine(); // Get the first real row of data
            totalInputCases = 0;
            while (pattern!=null) {
                pattern = br.readLine();
                totalInputCases++;

            }
            br.close();
            // Now read the data
            fr = new FileReader(f);
            br = new BufferedReader(fr);
            pattern = br.readLine(); // Get the first real row of data
            activations = new double[totalInputCases][totalInputNeurons];
            int rowCount = 0;
            while (pattern!=null) {
                String[] values = pattern.split(",");
                for (int i=2;i<values.length;i++) { // Skip 0 & 1 which are the math
                                                    // & reading scores
                    int value = Integer.parseInt(values[i]);
                    double activation = 0.0;
                    // Recode to [0.2,0.8]. If the response isn't in the list of
                    // valid responses, leave as 0
                    for (int j=0;j<inputFeatures[i-2].length;j++) {
                        if (value==inputFeatures[i-2][j]) {
```

```
                            // Found it, so scale the activaton value to [0.2,0.8]
                            activation = 0.2+0.6*((double)j/
                                (double)(inputFeatures[i-2].length-1));
                            break;
                        }
                    }
                    activations[rowCount][i-2]=activation;
                }
                rowCount++;
                pattern = br.readLine();
            }
            br.close();
        } catch (java.io.IOException e) {
            System.out.println("Error reading input data - "+e.getMessage());
        }
    }
    System.out.println("    Read "+totalInputCases+" input cases from "+path);
    return activations;
}

/**
 * Initializes the arrays containing the root names for the dummy variables
 * as well as the valid values for each dummy variable
 */
public static void setupInputFeatures() {

     int i = 0; // Index into the input arrays

     //BYFCOMP
     inputNames[i]="BYFCOMP";
     inputFeatures[i]=new int[]{1,2,3,4,5,6,7,8,9};
     i++;
     //PARED
     inputNames[i]="PARED";
     inputFeatures[i]=new int[]{1,2,3,4,5,6,7,8};
     i++;
     //MOTHED
     inputNames[i]="MOTHED";
     inputFeatures[i]=new int[]{1,2,3,4,5,6,7,8};
     i++;
     //FATHED
     inputNames[i]="FATHED";
     inputFeatures[i]=new int[]{1,2,3,4,5,6,7,8};
     i++;
     //SES1QU
     inputNames[i]="SES1QU";
     inputFeatures[i]=new int[]{1,2,3,4};
     i++;
     //STEXPECT
     inputNames[i]="STEXPECT";
     inputFeatures[i]=new int[]{-1,1,2,3,4,5,6,7};
     i++;
     //BYBASEBL
     inputNames[i]="BYBASEBL";
     inputFeatures[i]=new int[]{1,2,3,4,5};
     i++;
    //BYSOFTBL
     inputNames[i]="BYSOFTBL";
     inputFeatures[i]=new int[]{1,2,3,4,5};
     i++;
    //BYBASKTBL
     inputNames[i]="BYBASKTBL";
     inputFeatures[i]=new int[]{1,2,3,4,5};
     i++;
    //BYFOOTBL
     inputNames[i]="BYFOOTBL";
     inputFeatures[i]=new int[]{1,2,3,4,5};
     i++;
    //BYSOCCER
     inputNames[i]="BYSOCCER";
     inputFeatures[i]=new int[]{1,2,3,4,5};
```

```
 i++;
 //BYTEAMSP
 inputNames[i]="BYTEAMSP";
 inputFeatures[i]=new int[]{1,2,3,4,5};
 i++;
//BYSOLOSP
 inputNames[i]="BYSOLOSP";
 inputFeatures[i]=new int[]{1,2,3,4,5};
 i++;
 //BYSCTRL
 inputNames[i]="BYSCTRL";
 inputFeatures[i]=new int[]{1,2,3};
 i++;
 //BYURBAN
 inputNames[i]="BYURBAN";
 inputFeatures[i]=new int[]{1,2,3};
 i++;
 //BYREGION
 inputNames[i]="BYREGION";
 inputFeatures[i]=new int[]{1,2,3,4};
 i++;
 //BYS20A
 inputNames[i]="BYS20A";
 inputFeatures[i]=new int[]{1,2,3,4};
 inputCodings[i]=QUANT;
 i++;
 //BYS20B
 inputNames[i]="BYS20B";
 inputFeatures[i]=new int[]{1,2,3,4};
 i++;
 //BYS20C
 inputNames[i]="BYS20C";
 inputFeatures[i]=new int[]{1,2,3,4};
 i++;
 //BYS20E
 inputNames[i]="BYS20E";
 inputFeatures[i]=new int[]{1,2,3,4};
 i++;
 //BYS20F
 inputNames[i]="BYS20F";
 inputFeatures[i]=new int[]{1,2,3,4};
 i++;
 //BYS20G
 inputNames[i]="BYS20G";
 inputFeatures[i]=new int[]{1,2,3,4};
 i++;
 //BYS20J
 inputNames[i]="BYS20J";
 inputFeatures[i]=new int[]{1,2,3,4};
 i++;
 //BYS20M
 inputNames[i]="BYS20M";
 inputFeatures[i]=new int[]{1,2,3,4};
 i++;
 //BYS20N
 inputNames[i]="BYS20N";
 inputFeatures[i]=new int[]{1,2,3,4};
 i++;
 //BYS21A
 inputNames[i]="BYS21A";
 inputFeatures[i]=new int[]{1,2,3,4};
 i++;
 //BYS21B
 inputNames[i]="BYS21B";
 inputFeatures[i]=new int[]{1,2,3,4};
 i++;
 //BYS21C
 inputNames[i]="BYS21C";
 inputFeatures[i]=new int[]{1,2,3,4};
 i++;
 //BYS21D
```

```
inputNames[i]="BYS21D";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS21E
inputNames[i]="BYS21E";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS22A
inputNames[30]="BYS22A";
inputFeatures[30]=new int[]{1,2,3};
i++;
//BYS22B
inputNames[i]="BYS22B";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS22C
inputNames[i]="BYS22C";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS22D
inputNames[i]="BYS22D";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS22E
inputNames[i]="BYS22E";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS22F
inputNames[i]="BYS22F";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS22G
inputNames[i]="BYS22G";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS22H
inputNames[i]="BYS22H";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS24B
inputNames[i]="BYS24B";
inputFeatures[i]=new int[]{1,2,3,4,5};
i++;
//BYS26
inputNames[i]="BYS26";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS27A
inputNames[i]="BYS27A";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS27B
inputNames[i]="BYS27B";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS27C
inputNames[i]="BYS27C";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS27D
inputNames[i]="BYS27D";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS27E
inputNames[i]="BYS27E";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS27F
inputNames[i]="BYS27F";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
```

```
//BYS27G
inputNames[i]="BYS27G";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS27H
inputNames[i]="BYS27H";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS27I
inputNames[i]="BYS27I";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS28
inputNames[i]="BYS28";
inputFeatures[i]=new int[]{-1,1,2,3};
i++;
//BYS34A
inputNames[i]="BYS34A";
inputFeatures[i]=new int[]{
    -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
i++;
//BYS34B
inputNames[i]="BYS34B";
inputFeatures[i]=new int[]{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
    21,22,23,24,25,26};
i++;
//BYS35A
inputNames[i]="BYS35A";
inputFeatures[i]=new int[]{
    -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
i++;
//BYS35B
inputNames[i]="BYS35B";
inputFeatures[i]=new int[]{
    -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
i++;
//BYS36A
inputNames[i]="BYS36A";
inputFeatures[i]=new int[]{
    -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
i++;
//BYS36B
inputNames[i]="BYS36B";
inputFeatures[i]=new int[]{
   -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
i++;
//BYS37
inputNames[i]="BYS37";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS38B
inputNames[i]="BYS38B";
inputFeatures[i]=new int[]{1,2,3,4};
i++;
//BYS39A
inputNames[i]="BYS39A";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS39B
inputNames[i]="BYS39B";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS39C
inputNames[i]="BYS39C";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS39D
inputNames[i]="BYS39D";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS39E
```

```
inputNames[i]="BYS39E";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS39F
inputNames[i]="BYS39F";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS39G
inputNames[i]="BYS39G";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS39H
inputNames[i]="BYS39H";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS41A
inputNames[i]="BYS41A";
inputFeatures[i]=new int[]{-1,0,1};
i++;
//BYS41B
inputNames[i]="BYS41B";
inputFeatures[i]=new int[]{0,1};
i++;
//BYS41C
inputNames[i]="BYS41C";
inputFeatures[i]=new int[]{0,1};
i++;
//BYS41D
inputNames[i]="BYS41D";
inputFeatures[i]=new int[]{-1,0,1};
i++;
//BYS41E
inputNames[i]="BYS41E";
inputFeatures[i]=new int[]{0,1};
i++;
//BYS41F
inputNames[i]="BYS41F";
inputFeatures[i]=new int[]{-1,0,1};
i++;
//BYS41G
inputNames[i]="BYS41G";
inputFeatures[i]=new int[]{-1,0,1};
i++;
//BYS41H
inputNames[i]="BYS41H";
inputFeatures[i]=new int[]{-1,0,1};
i++;
//BYS41I
inputNames[i]="BYS41I";
inputFeatures[i]=new int[]{-1,0,1};
i++;
//BYS42
inputNames[i]="BYS42";
inputFeatures[i]=new int[]{
    -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
i++;
//BYS43
inputNames[i]="BYS43";
inputFeatures[i]=new int[]{
   -1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21};
i++;
//BYS45A
inputNames[i]="BYS45A";
inputFeatures[i]=new int[]{1,2,3,4,5};
i++;
//BYS45B
inputNames[i]="BYS45B";
inputFeatures[i]=new int[]{1,2,3,4,5};
i++;
//BYS45C
inputNames[i]="BYS45C";
```

```
inputFeatures[i]=new int[]{1,2,3,4,5};
i++;
//BYS46A
inputNames[i]="BYS46A";
inputFeatures[i]=new int[]{0,1,2,3,4,5,6};
i++;
//BYS46B
inputNames[i]="BYS46B";
inputFeatures[i]=new int[]{0,1,2,3,4,5,6};
i++;
//BYS54A
inputNames[i]="BYS54A";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS54B
inputNames[i]="BYS54B";
inputFeatures[i]=new int[]{-1,1,2,3};
i++;
//BYS54C
inputNames[i]="BYS54C";
inputFeatures[i]=new int[]{-1,1,2,3};
i++;
//BYS54D
inputNames[i]="BYS54D";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS54E
inputNames[i]="BYS54E";
inputFeatures[i]=new int[]{-1,1,2,3};
i++;
//BYS54F
inputNames[i]="BYS54F";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS54G
inputNames[i]="BYS54G";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS54H
inputNames[i]="BYS54H";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS54I
inputNames[i]="BYS54I";
inputFeatures[i]=new int[]{-1,1,2,3};
i++;
//BYS54J
inputNames[i]="BYS54J";
inputFeatures[i]=new int[]{-1,1,2,3};
i++;
//BYS54K
inputNames[i]="BYS54K";
inputFeatures[i]=new int[]{-1,1,2,3};
i++;
//BYS54L
inputNames[i]="BYS54L";
inputFeatures[i]=new int[]{-1,1,2,3};
i++;
//BYS54N
inputNames[i]="BYS54N";
inputFeatures[i]=new int[]{-1,1,2,3};
i++;
//BYS54O
inputNames[i]="BYS54O";
inputFeatures[i]=new int[]{1,2,3};
i++;
//BYS57
inputNames[i]="BYS57";
inputFeatures[i]=new int[]{-1,1,2,3,4,5};
i++;
//BYS60
```

```
            inputNames[i]="BYS60";
            inputFeatures[i]=new int[]{0,1};
            i++;
            //BYS61
            inputNames[i]="BYS61";
            inputFeatures[i]=new int[]{0,1};
            i++;
            //BYS66A
            inputNames[i]="BYS66A";
            inputFeatures[i]=new int[]{-1,1,2,3,4,5,6};
            i++;
            //BYS66B
            inputNames[i]="BYS66B";
            inputFeatures[i]=new int[]{-1,1,2,3,4,5,6};
            i++;
            //BYS66F
            inputNames[i]="BYS66F";
            inputFeatures[i]=new int[]{-1,1,2,3,4,5,6,7};
            i++;
            //BYS72
            inputNames[i]="BYS72";
            inputFeatures[i]=new int[]{1,2,3};
            i++;
    }
}
```

APPENDIX E

Neural Network Listener Source Listing

```
import org.joone.engine.NeuralNetListener;
import org.joone.engine.NeuralNetEvent;
import org.joone.engine.Monitor;
import org.joone.net.NeuralNet;
import org.joone.io.MemoryInputSynapse;
import org.joone.io.MemoryOutputSynapse;

/**
 * Helper class that listens for NeuralNetwork events and performs functions such as
 * printing the current RMSE for the network, validating the network, and saving the
 * network
 */
public class MyNeuralNetListener implements NeuralNetListener {

    public NeuralNet net;
    public NeuralNet bestNet = null;
    public double bestRMSE = 0;

    public MyNeuralNetListener(NeuralNet theNet) {
        net = theNet;
    }

    public void netStarted(NeuralNetEvent nne) {
        System.out.println("Starting the net");
    }

    public void cicleTerminated(NeuralNetEvent nne) {
    }

    public void netStopped(NeuralNetEvent nne) {
        System.out.println("Stopping the Net");
        Monitor monitor = (Monitor)nne.getSource();
        if (monitor.isLearning()) {
            ELS2002NeuralNet.saveNet(bestNet);
        }
    }

    public void errorChanged(NeuralNetEvent nne) {
        Monitor monitor = (Monitor)nne.getSource();
        int currentCicle = monitor.getCurrentCicle();
        if (currentCicle%10==0 || currentCicle==1) {
            System.out.println("Total Error Changed - "+monitor.getGlobalError()+
                " - Current Cicle "+currentCicle);

            monitor.setExporting(true);
            NeuralNet newNet = net.cloneNet();
            monitor.setExporting(false);
            newNet.getInputLayer().removeAllInputs();
            MemoryInputSynapse testData = new MemoryInputSynapse();
            testData.setFirstRow(1);
            testData.setAdvancedColumnSelector(ELS2002NeuralNet.inputColumns);
            newNet.getInputLayer().addInputSynapse(testData);
            double[][] inputData = ELS2002NeuralNet.getTestCases();
            testData.setInputArray(inputData);
            newNet.getOutputLayer().removeAllOutputs();
            MemoryOutputSynapse testResults = new MemoryOutputSynapse();
            newNet.getOutputLayer().addOutputSynapse(testResults);
            newNet.getMonitor().setTotCicles(1);
            newNet.getMonitor().setTrainingPatterns(inputData.length);
            newNet.getMonitor().setLearning(false);

            double[][] outputData = ELS2002NeuralNet.getTestTargets();
            System.out.println("Running test cycle with "+inputData.length+" cases");
            newNet.go();

            double Math_SSResidual = 0; // Variability about the regression line
            double Math_SSRegression = 0;
            double Reading_SSResidual = 0;
            double Reading_SSRegression = 0;

            int READING_SCORE = ELS2002NeuralNet.READING_SCORE;
```

```
            int MATH_SCORE = ELS2002NeuralNet.MATH_SCORE;

            // Compute the average reading score to be used in the calculation of R^2
            double totalReading = 0;
            double totalMath = 0;
            for (int i=0;i<outputData.length;i++) {
                totalReading+=outputData[i][READING_SCORE];
                totalMath+=outputData[i][MATH_SCORE];
            }
            double avgReading = totalReading/(double)outputData.length;
            double avgMath = totalMath/(double)outputData.length;

            double Math_SSTotal = 0;
            double Reading_SSTotal = 0;
            for (int i=0;i<outputData.length;i++) {
                double reading_diff = outputData[i][READING_SCORE]-avgReading;
                double math_diff = outputData[i][MATH_SCORE]-avgMath;
                Reading_SSTotal += (reading_diff*reading_diff);
                Math_SSTotal += (math_diff*math_diff);
            }

            for (int i=0;i<inputData.length;i++) {
                double[] results = testResults.getNextPattern();
                double math_err = outputData[i][MATH_SCORE]-results[MATH_SCORE];
                double reading_err = outputData[i][READING_SCORE]-results[READING_SCORE];
                Math_SSResidual += (math_err*math_err);
                Reading_SSResidual += (reading_err*reading_err);
            }
            Math_SSRegression = Math_SSTotal-Math_SSResidual;
            Reading_SSRegression = Reading_SSTotal-Reading_SSResidual;

            double math_rmse = Math.sqrt(Math_SSResidual/(double)inputData.length);
            double reading_rmse = Math.sqrt(Reading_SSResidual/(double)inputData.length);

            System.out.println("    Math RMSE (Sqrt(SSResidual/n)) = "+math_rmse);
            System.out.println("    Reading RMSE                   = "+reading_rmse);
            double avg_rmse = (math_rmse+reading_rmse)/2.0;
            if (bestNet == null || avg_rmse < bestRMSE) {
                System.out.println("    New best net, average rmse = "+avg_rmse);
                bestNet = newNet;
                bestRMSE = avg_rmse;
            }
        }
    }

    public void netStoppedError(NeuralNetEvent nne, String str) {
        System.out.println("The net stopped due to an error - "+str);

    }
}
```