Volume 3, Issue 10, October 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Software Cost Estimation in Multilayer Feed forward Network using Random Holdback Method

K. Subba Rao [*1]
[1]*IT Department,*
LBRCE , India

L.S.S Reddy [2]
[2]*CSE Department,*
KL University, India

Nagaraju Devarakonda[3], **Shaik Subhani**[4], **K. Raviteja**[5]
[3,4,5]*CSE Department,*
Acharaya Nagarajuna University, India

Abstract— Software cost and effort estimation is the most critical task in handling software projects. Since it is very difficult to bridge the gap between estimated cost and actual cost, hence the accurate cost estimation is one of the challenging tasks in maintaining software projects. The software cost estimation is now one of Centre of attention for computer software industries. As software industry runs many projects simultaneously they have to prioritize different processes based on time, cost, and number of staff, sequentially. With the increasing complexity of software, the cost of the software development is also increasing. So it is required to rely on the effective techniques for estimating software costs. Accurate cost estimation is needed because it can help to prioritize and classify development projects. Software project effort estimation is frequently seen as complex and expensive for individual software engineers. Software production is in a crisis. It agonizes from unrestrained costs. Software production is recurrently out of control. It has been propounded that software production is out of control because we do not measure. You cannot bridle what you cannot measure. During last decade, huge analysis on cost estimation has been conducted. The Holdback method has a vital role in software cost estimation studies; its importance has been ignored especially in neural network based studies. The model presented here is based on Multi-Layer Feed forward network (MLF).Another difficulty associated with the cost estimation studies is the fact that the data collection requires time and care. Neural networks can be applied in software cost estimation studies with success. Failures of software are mainly due to the faulty project management practices, which include effort estimation. Continuous changing synopsis of software development technology makes effort estimation more challenging. Accurate software cost estimates are critical to both developers and customers. They can be used for procreating request for proposals, contract negotiations, scheduling, monitoring and control. The distinct relationship between the attributes of the effort estimation is difficult to enact. A neural network is good at pioneering relationships and pattern in the data. This paper is concerned with constructing software effort estimation model based on neural networks. The test results from the trained neural network are compared with (model name) model. By comparing the results of this model, it is proven that both works better than is an optimal neural network model for software effort estimation.

Keywords— Software Cost Estimation, feed forward neural networks, Holdback, K-hold, abalone.

1. INTRODUCTION

Software cost estimation is not a new issue in IT. Ever since the complexity and size of computer programs has been growing there has been an increasing need for reliable cost estimation. Whereas the very first computer programs were extremely limited in proportions and range, current day software projects can necessitate millions of lines of code, up to a thousand simultaneous developers, and immense supporting services, framework and management teams. Such projects are expensive, thorough planning is required to guide the available resources in an flawless way. Because the financial emanation of imprecise cost predictions is considerable, there is an ample exhortation to spend effort in doing formal cost predictions. As well as running over fiscal estimate, there is the uncertainty that projects are not completed in time. Certain time critical software applications have a fixed time limit and must be delivered before the deadline has passed. Furthermore, running late on software projects may lead to entire product lines being delayed or scrapped altogether. Business opportunities are overwhelmingly time critical not delivering the right product in time may lead to a competitor taking a commanding market share, effectively eliminating the company from this particular market articulation . A massive software company may often produce hundreds of applications of varying sizes. These risks have led to software cost estimation becoming more common in any business that produces large volumes of software. Despite this need there has not been a single convincing theory for cost estimation. Most methodologies are model based and depend on the number of lines of code. Very few cost estimation methodologies have considered looking at function points as a measure of size. Today, there is no one model that can accurately predict the costs involved in any software development effort.

Software estimating is facile in concept, but tough to do properly in realism. The scope and size of projects influences the number of factors that must be reckon to get an true prediction of the contingent costs. The complication of manually doing reckoning for larger software projects usually exceeds the capabilities of most project managers. The amount of work required for activities other than coding are often misjudged and can vary greatly between different types of projects. Although automated cost estimation estimates are patchy, they are mostly more rigorous than human predictions and are far less costly. Software becomes increasingly expensive to develop and is a major cost factor in any information system budget. Software development costs often get out of control due to lack of measurement and estimation methodologies. Software cost estimation or software effort estimation is the process of predicting the effort required to develop a software system. Software engineering cost models and estimation techniques are used for a number of purposes including;

Budgeting, trade off and risk analysis, project planning and authority, and software improvement investment analysis. The accuracy of the software project cost estimation has a direct and significant impact on the quality of the firm's software investment decisions. Accurate cost estimation can reduce the unnecessary costs and increase the organization's efficiency. For this reason, many estimation models have been proposed over the last 20 years. Unfortunately, despite the large body of experience with estimation models, the perfection of these models is still far from being satisfactory. Software development effort estimation with the aid of artificial neural networks (ANN) attracted considerable research interest especially at the beginning of the nineties. A key factor in selecting a cost estimation model is the accuracy of its metrics, since these models rely on metrics as their input. Metric can be defined as a quantitative measure of the degree to which a system, intrinsic, or process possesses a given attribute. It may seem easy to think of attributes of computer software products, processes, people or programming environments that can be measured.

Software cost and effort estimate is one of the most important activities in software project management. It is the accuracy of cost and effort calculation that enable quality growth of software at a later stage. With an effective estimate of software cost and effort, software developers can efficiently decide what recourses are to be used frequently and how efficiently these resources can be utilized. For efficient software, accurate software development parameters are required, these include effort estimation, development time estimation, cost estimation, team size estimation, risk analysis, etc. Since the effort and cost estimation is done at an early stage of software development; hence a good model is required to calculate these parameters accurately. In past few decades several researchers have worked in the field of software effort estimation, and many conventional models were designed to estimate software, size and effort. These models require inputs which are difficult to obtain at early stages of software development. Moreover these models take Values of software development factors based on experience and approximation, with zero reasoning Capability. Due to few such limitations of conventional algorithmic models, non-algorithmic models based on Soft Computing came into picture, which include Neural Network, Fuzzy logic and Genetic algorithms. The non-algorithm based algorithm work with real life situations and a vast flexibility for software development factors was provided.

Software effort estimation is the process of predicting the most realistic use of effort required to develop or maintain software. Effort estimates are used to calculate effort in person-months (PM) for the Software Development work elements of the Work Breakdown Structure (WBS).Accurate Effort Estimation is important because:

- It can help to codify and compile development projects with respect to an overall business plan.
- It can be used to arbitrate what resources to commit.
- It can be used to compute the impact of changes and support re-planning.
- Projects can be simple to manage and restrict when resources are better matched to real needs.
- Customers expect absolute development costs to be in line with estimated costs.

Software cost estimation is one of the most critical tasks in managing software projects. Accurate estimation of a software development effort is critical for good management decision making. Development costs tend to increase with project complexity and hence accurate cost estimates are highly desired during the early stages of development. An important objective of the software engineering community has been to develop an appropriate model to estimate the software effort accurately. Artificial neural networks can model complex non-linear relationships and approximate any measurable function. In general, for software cost estimation, the most commonly adopted architecture, learning algorithm and activation function are the feed-forward multi-layer perceptron, the back propagation algorithm and the sigmoid function respectively. However, there are some shortcomings that prevent it from being accepted as common practice in cost estimation. The paper is organized in following sections: section 1 describes introduction, sections 2 and 3 describes existing work and neural network. Section 4 discusses the related work and proposed neural network model is described in section 5. Experimental results and evaluation criteria are shown in section 6. Section 7 ends the paper with a conclusion.

2. RELATED WORK

A Neural Network (NN) is an artificial, computational model that simulates biological neural networks. Basically, a Neural Network consists of linked, artificial neurons which are typically grouped to input, hidden, and output layers. Depending on the network structure, different network types can be identified. In contrast to recurrent networks, Feed-Forward Networks represent a directed acyclic graph. Information is forwarded in one direction only, consecutively processed by the input, hidden, and output neurons. Neural networks consist of layers of interconnected nodes, where each node produces a non-linear function of its input. The nodes in the network are divided into the ones from the input

layer going through the network to the ones at the output layer through some nodes in a hidden layer. The NN process starts by developing the structure of the network and establishing the technique used to train the network with using an actual data set. Therefore, there are three main entities:

1. Neurons (nodes),
2. Interconnection structure
3. Learning algorithm

Neural networks are the interconnection of the artificial neurons. They are used to solve the artificial intelligence problems without the need for creating a real biological model. The neural network used in our approach. The activation function is one of the key components of the perceptron as in the most common neural network architectures. It determines based on the inputs, whether the perceptron activates or not. The perceptron takes all of the weighted input values and adds them together. If the sum is above or equal to some value (called the threshold) then the perceptron fires. Otherwise, the perceptron does not.

Neural Network is used in effort estimation due to its ability to learn from previous data. It is also able to model knotty relationships between the dependent (effort) and independent variables (cost drivers). In addition, it has the ability to induce from the training data set thus enabling it to produce acceptable result for antecedent unseen data. Most of the work in the application of neural network to effort estimation made use of cascade correlation and Back-propagation algorithm. An artificial neural network comprises of eight basic components: (1) neurons, (2) activation function, (3) signal function,(4) pattern of connectivity (5) activity aggregation rule (6) activation rule (7) learning rule and(8) environment. After an NN is created it must go through the process of learning or training. The process of changing the weights in the connections between network layers with the objective of achieving the expected output is called training a network. There are two approaches for training– supervised and unsupervised .In supervised training; both the inputs and the outputs are provided. The network then processes the inputs, compares its accurate outputs against the desired outputs and error is calculated. In unsupervised training, the network is provided with inputs but not with desired outputs. The system itself must then decide what features it will use to group the input data. Most of the work in the application of neural network to effort estimation made use of Back-propagation algorithm and cascade correlation. Many different models of neural nets have been proposed for solving many complex real life problems.

3. COST ESTIMATION METHODS

Cost estimation methods are basically of two types: algorithmic and non-algorithmic. Algorithmic methods use a formula to calculate the software cost estimate. The formula is evolved from models which are created by combining related cost factors. In addition, the analytical method is used for model fabrication. Non-algorithmic methods do not use a formula to calculate the software cost estimate.

3.1 Non Algorithmic Based Estimation Methods

3.1.1 Expert Judgment Method

Expert judgment techniques involve consulting with software cost estimation expert or a group of the experts to use their experience and understanding of the proposed project to arrive at an estimate of its cost. It is the most usable methods for the software cost estimation. Mostly companies use this method to bring about the cost of the product. Generally speaking, a group consensus technique, Delphi technique, is the best way to be used. To provide a satisfactorily broad communication bandwidth for the experts to exchange the volume of information necessary to calibrate their estimates with those of the other experts, a wideband Delphi technique is introduced over standard Delphi technique. The estimating steps using this method are as follows:

1. Coordinator presents each expert with a specification and an estimation form.
2. Experts fill out forms anonymously
3. Coordinator calls a group meeting in which the experts discuss estimation issues with the coordinator and each other.
4. Coordinator prepares and distributes a summary of the estimation on an iteration form.
5. Experts fill out forms, again enormously, and steps 4 and 6 are iterated for as many rounds as appropriate.

The wideband Delphi Technique has subsequently been used in a number of studies and cost estimation activities. It has been highly efficacious in linking the free discuss advantages of the group meeting technique.

3.1.2 Estimating by Analogy

Estimating by analogy means comparing the proposed project to previously completed similar project where the project development information id known. Actual data from the completed projects are extrapolated to estimate the proposed project. This method can be used at system-level or at the component-level. The estimating steps using this method are as follows:

1. Find out the characteristics of the proposed project.
2. Select the most similar completed projects whose characteristics have been stored in the historical data base.

3. Find out the estimate for the proposed project from the most similar completed project by analogy.

The process involved to predict software cost estimates can be broken down into the following steps according to

1. Selection of analogies
2. Assessing similarities and differences
3. Assessment of analogy quality
4. Consideration of any special cases
5. Creating the estimate

3.1.3 Top-Down Estimating Method

Top-down estimating method is also called Macro Model. Using top-down estimating method, an overall cost estimation for the project is derived from the global properties of the software project, and then the project is partitioned into various low-level mechanism or components. The leading method using this approach is Putnam model. This method is more applicable to early cost estimation when only global properties are known. In the early phase of the software development, it is very useful because there is no detailed information available.

3.1.4 Bottom-up Estimating Method

Using bottom-up estimating method, the cost of each software components is estimated and then combines the results to arrive at an estimated cost of overall project. It aims at constructing the estimate of a system from the knowledge accumulated about the small software components and their interactions. The leading method using this approach is COCOMO's detailed model.

3.1.5 Parkinson's Law

Using Parkinson's principle ―work expands to fill the available volume, the cost is determined (not estimated) by the available resources rather than based on an objective assessment. If the software has to be delivered in 12 months and 5 people are available, the effort is estimated to be 60 person-months. Although it sometimes gives good estimation, this method is not recommended as it may provide very unrealistic estimates. Also, this method does not promote good software engineering practice.

3.1.6 Price-to-win

The software cost is estimated to be the best price to win the project. The estimation is based on the customer's budget instead of the software functionality. For example, if a reasonable estimation for a project costs 100 person-months but the customer can only afford 60 person-months, it is common that the estimator is asked to modify the estimation to fit 60 person months 'effort in order to win the project. This is again not a good practice since it is very likely to cause a bad delay of delivery or force the development team to work overtime.

3.2 Algorithmic Method

3.2.1 Basic cost model

$$\text{Effort} = A * \text{Size}^{B} * m(X)$$

Size: Some measurement of the software size
A: Constant factor that depends on Organizational practices Type of software
B: Usually lies between 1 and 1.5
X: Vector of cost factors
m: Adjustment multiplier

The algorithmic method is designed to provide some mathematical equations to perform software estimation. These mathematical equations are based on research and historical data and use inputs such as Source Lines of Code (SLOC), number of functions to perform, and other cost drivers such as language, design methodology, skill-levels, risk assessments, etc. The algorithmic methods have been largely studied and many models have been developed, such as COCOMO models, Putnam model, and function points based models [11] [28].

3.2.2 COCOMO Model

Constructive Cost Model (COCOMO) is widely used algorithmic software cost model [2][4]. It has following hierarchy-

Model 1 (Basic COCOMO Model):- The basic COCOMO model computes software development effort and cost as a function of program size expressed in estimated lines of code (LOC) [6] [23]. The basic steps in this Model are:-

a. Obtain an initial estimate of the development effort from the estimate of thousands of delivered lines of source code (KLOC).
b. Determine a set of 15 multiple factors from different attributes of the project.
c. Adjust the effort estimate by multiplying the initial estimate with all the multiplying factors.

The initial estimate (also called nominal estimate) is determined by an equation of the form used in the static single variable models, using KLOC as the measure of size. To determine the initial effort in person-months the equation used is of the type

$$\text{EFFORT} = a * (\text{KLOC})^{b} \qquad (1)$$

The value of constants a and b depend on the project type

Model 2 (Intermediate COCOMO Model):- Intermediate COCOMO Model computes software development effort as a function of program size and set of ―cost drivers‖ that include subjective assessment of the products, hardware, personnel and project attributes.

1. Product attributes
- a. Required software reliability
- b. Size of application data base
- c. Complexity of the product

2. Hardware attributes
- a. Run-time performance constraints
- b. Memory constraints
- c. Volatility of the virtual machine environment
- d. Required turnaround time

3. Personnel attributes
- a. Analyst capability
- b. Software engineer capability
- c. Applications experience
- d. Virtual machine experience
- e. Programming language experience

4. Project attributes
- a. Use of software tools
- b. Application of software engineering methods

- c. Required development schedule each of the 15 attributes is rated on a 6 point scale that ranges from ―very low‖ to ―extra high‖ (in importance or value). Based on the rating, an effort multiplier is determined from tables

  published by Boehm, and the product of all effort multipliers results is an *effort adjustment factor* (EAF). Typical values for EAF range from 0.9 to 1.4 [4].

The intermediate COCOMO model takes the form:

$$EFFORT = a^* (KLOC)^b * EAF \quad (2)$$

Where effort in person-months and *KLOC* is the estimated number of delivered lines of code for the project

Model 3 (Detailed COCOMO Model):- The detailed COCOMO Model incorporates all characteristics of the intermediate version with an assessment of the cost driver‘s impact on each step (analysis, design, etc) of the software engineering process.

2. COCOMO II model

It is a collection of three variants, Application composition model, Early design model, and Post architecture model. This is an extension of intermediate COCOMO model [14] and defined as:-

$$EFFORT = 2.9 (KLOC)^{1.10} \quad (3)$$

3.2.3 SEL – Model

Estimation of effort according to SEL model is defined as follows[18]

$$EFFORT = 1.4 * (Size)^{0.93}$$
$$Duration\ D = 4.6 (KLOC)^{0.26} \quad (4)$$

Effort (Person-Months) and lines of code (size in thousands of lines of code i.e. KLOC) are used as predictors.

3.2.4 Walston-Felix Model

This model is developed from various aspects of the software development environment such as user database of sixty projects collected in IBM's Federal Systems division. It provides a relationship between delivered lines of source code. This model constitutes participation, customer-oriented changes, memory constraints etc. According to Walston and Felix model, effort is computed by [26] [14]:

$$EFFORT = 5.2 (KLOC)^{0.91}$$
$$Duration\ D = 4.1 (KLOC)^{0.36} \quad (5)$$

3.2.5 Bailey-Basil Model

This model developed by Bailey-Basil between delivered lines of source code and formulates a relation [12]:

$$EFFORT = 5.5 (KLOC)^{1.16} \quad (6)$$

3.2.6 Halstead Model

This model developed by Halstead between delivered lines of source code and formulates a relation [18]

$$EFFORT = 0.7 (KLOC)^{1.50} \quad (7)$$

3.2.7 Doty (for KLOC > 9)

This model developed by Doty between delivered lines of source code and formulates a relation [18]

$$EFFORT = 5.288 (KLOC)^{1.047} \quad (8)$$

3.2.8 Putnam Model

The Putnam model is an empirical software effort estimation model. Putnam used his observations about productivity levels to derive the software equation:

Technical constant C= size * B1/3 * T4/3

Total Person Months B=1/T4 *(size/C)3

T= Required Development Time in years

Size is estimated in LOC
Where: C is a parameter dependent on the development environment and is determined on the basis of historical data of the past projects[31].

Rating: C=2,000 (poor), C=8000 (good) C=12,000 (excellent).

The Putnam model is very sensitive to the development time: decreasing the development time can greatly increase the person-months needed for development [22] [8].

3.2.9 Function Point Analysis

The Function Point Analysis is another method of quantifying the size and complexity of a software system in terms of the functions that the systems deliver to the user. A number of proprietary models for cost estimation have adopted a function point type of approach, such as ESTIMACS and SPQR/20 [5]. The total number of function points depends on the counts of distinct (in terms of format or processing logic) types.

There are two steps in counting function points:

a. Counting the user functions:- The raw function counts are arrived at by considering a linear combination of five basic software components: external inputs, external outputs, external inquiries, logic internal files, and external interfaces, each at one of three complexity levels: simple, average or complex. The sum of these numbers, weighted according to the complexity level, is the number of function counts (FC).
b. Adjusting for environmental processing complexity:- The final function points is arrived at by multiplying FC by an adjustment factor that is determined by considering 14 aspects of processing complexity. This adjustment factor allows the FC to be modified by at most 35% or -35% [31].

3.2.10 Bailey & Basili

This model is proposed to be used to express the size of the project with some measures like Line of code, executable statement, machine instructions, number of modules and a base line equation to relate this size to effort [34]. It was discovered by authors that rather considering only the total lines or only new lines to determine the size of a project; a better way is to use an algorithm to combine these sizes into one measure. By keeping this in mind they suggested that a baseline relationship of lower standard error can be derived by computing the effective size in lines to be equal to total number of new lines plus 20% of old lines used in the project [34]. They called this new size measure developed lines *DL* and they adopted same approach to measure developed modules. The equation provided by authors with 1.25 standard error estimate, further discussed at [37], is given as:

$$Effort = 0.73* DL^{1.16} *3.5$$

3.2.11 Farr & Zagorski

Probably this is the earliest known model proposed around 1965 at ICC symposium on economics of automatic data processing [35]. This linear model proposed productivity factors i-e delivered statements, size of data base etc that are determined by regression analysis of three linear equations resulting into the effort required in man months to complete the system. [35][36][90]. Coefficients and Cost Drivers in the Farr-Zagorski. The equation that can be used to estimate effort in MM and cost is given as [36].

$$MM = \Sigma x_i j_i \qquad \text{Total Cost} = MM(\text{Labor rate})$$

3.2.12 Nelson model

It is a linear model that was developed in 1966 .During the development of this model, Nelson studied 169 databases of different projects. These projects were selected from different field's i.e Business, Scientific, Computer software and other. Initially Nelson identified 104 attributes and later only most significant 14 cost drivers were used in estimation process [38]. In his work, Nelson proposed equations for estimating manpower, computer usage, and months elapsed. These equations were proposed only for the Computer Program Design, Code and Test activity. For the total sample of 169 projects, Nelson has computed Total Man months, Total computer Hours and Total Elapsed time as follows:

Total Man month = $-33.63 + 9.15 X_3 + 10.73 X_8 + .51 X_{26} + .46 X_{30} + .40 X_{41} + 7.28 X_{42} - 21.45 X_{48.1} + 13.53 X_{48.5} + 12.35 X_{51} + 58.82 X_{53} + 30.61 X_{56} + 29.55 X_{72} + .54 X_{75} - 25.20 X_{76}$

Total Computer Hours $= +80.0 + 105 X^{8} + 2.70 X^{35} + 21.2 X^{37} + .158 X^{46} + 10.8 X^{47..1} - 6.85 X^{47.3} + 292 X^{48.3} - 63.3 X^{52} + 266 X^{56} + 3.59 X^{57} - 3.95 X^{64} + 240 X^{72} - 173 X^{76}$

Total Months Elapsed = $1.31 + 1.31 X_8 + 0.020 X_{19} + .845 X_{37} + 2.37 X_{40} + .037 X_{41} + .098 X_{47.1} - .079 X_{47.3} + .860 X_{48.3} + 2.05 X_{51} - 2.65 X_{65} + 3.63 X_{72} + .039 X_{75} - 1.86 X_{76}$

"+" sign in above equations show that the resource varies directly with the variable. If the variable is increased then the amount of resource will also increase. On the other hand the "-" sign represents that a resource varies inversely with variable [38]. "X" represents the dependent variables or factors that Nelson identified in his statistical experiments that finally led to the numeric values and equations. Nelson also applied these equations on the subset of total sample i-e to obtain total man months, total computer hours and total elapsed time for Business, Scientific, Computer software. Interested readers may see [38] In our next sections we will use Nelson equation for computing purpose. In order to avoid the lengthy equation we adopt a general form to represent Nelson model as follows so that we may use it with ease.

$$\text{Effort} = \text{Constant value} + \Sigma c_i x_i$$

Another name that is more in practice for Nelson model is System Development Corporation model SDC[36][39]. However, the number of predictors used in [36] and [39] are 115 and in general form the equation may be given as

$MM = \Sigma ciji$
Cost = MM (Labor rate per year/12)

3.2.13 Boeing Model [39]

This model was developed by Boeing computer services in 1976 under the contract to US air force. The model has provided the estimates of total person-months of development, person months of development during each phase and estimates of computer time. The inputs to the model were

(i) Number of lines of deliverable source code
(ii) percentage of statements
(iii) the six defined phases and
(iv) the 8 environmental factors and each factor has six associated weight that are used to adjust and the environmental factors.

The model estimates development effort for each of the six phases and amount of computer time required for development. Development time estimate is computed through percentages of statements and number of lines of code estimate. The estimate is then adjusted by environmental factors to have final estimate.

3.2.14 Aron model

It was proposed by J.D. Aron that attempts to provide pragmatic determined productivity rates. Aron's model served as a basis for productivity rate estimation in software estimation field [41]. It is a quantitative method of estimating manpower resources for large programming jobs. This method is based on the available historical data on productivity rate of programmers and later this data is used to adjust other non programming attributes of the system [42][35][41]. Aron used following expression to calculate the man-months for any individual task. The total effort in man months is the sum of all individual man-months for each task[42].

Man Months = (Deliverable Instructions)/ (Instruction per Man Month)

Aron model is considered to be the pioneer of quantitative modeling in estimation as it has provided a good guide line for estimating man power for large systems but ,It is not being presented as a precise method and, in fact, it is not as good as an estimate based on sound experience[42].

3.2.15 Wolverton(TRW) Model

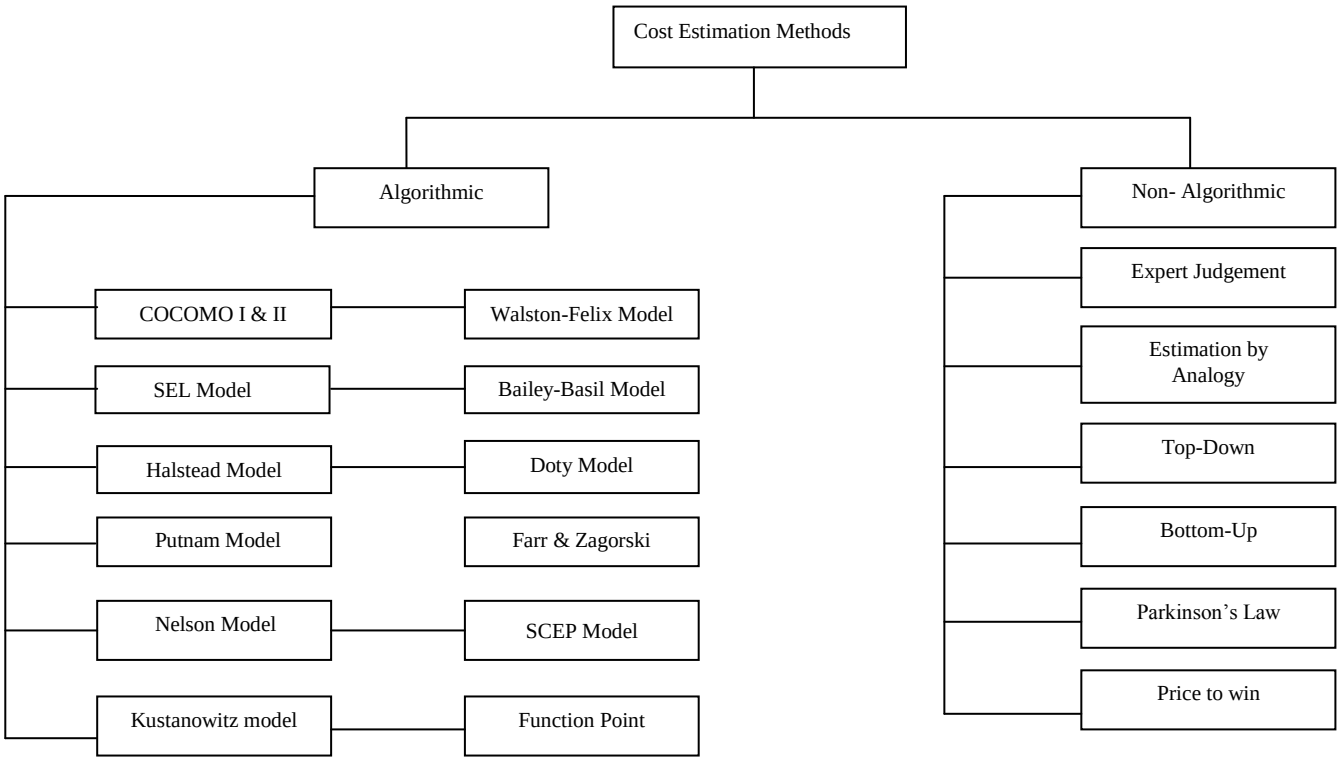
Wolverton model was an algorithm based estimation model relied on the assumption that costs is proportional to the number of instructions. The different values that are related to the routine category and complexity level are used to convert routine size to costs. The degree of complexity or difficulty was classified as easy, medium or hard on the basis of its newness or already gone through in some previous project. Total development cost may be obtained by multiplying the cost per instruction of single routine to number of object instruction and finally summing up individual results as a single effort value. The equation used for computing effort is given as [44] .

$$\text{Cost} = \sum_{M=1}^{k} Effort(m)$$

Wolverton model also suggested the use of effort adjustment multipliers. A total of six multipliers 1) control, (2) input/output, (3) pre/post processor, (4) algorithm, (5) data management, and (6) time critical were used with the complexity scales proposed as old-easy, old-medium, old-hard, new-easy, new-medium, and new-hard[44].

3.2.16 SCEP Model [39]

TRW software cost estimating program (SCEP) was developed in 1978 to estimate person months to complete different phases (preliminary design, detailed design, coding and unit test, integration and final test) of one or more user defined subsystems.. A database of 20 completed projects [9] was used to produce different equations and determine different factors that collectively yielded development time estimates in person months. This model was a typical work break down structure approach to assure accurate estimation and enhancing traceability. However, the limited availability and limited estimation to only few phases of life cycle has lessened the utilization of this model.



3.2.16 Kustanowitz model [91]

This model is proposed by Alvin Kustanowitz in 1977 to estimate the man power required for software implementation and this technique is famously known as SLICE (Software Lifecycle Estimation). The model proposed following stepwise approach for estimation [41].

Step 1: From the all possible activities in life cycle of software a list comprising a project profile which is created and modified according to the environment. According to kustanowitz a typical project profile involves 6-10 phases or steps (Functional Requirements Definition, Conceptual Systems Design, System Design, Program Specification, Coding, Unit Test, System Test).

Step 2: On the available historical data, the percentages were assigned to the phases of project profile.

Step 3 & 4 : The determination of the productivity rate in the form of average number of instructions per day on the grounds of historical data and the determination that whether the productivity rate is applicable for any particular phase or not. This applicability was considered because productivity rate is determined to be LOC per day and can only be appropriate for programming activities.

Step 5: Estimate the total number of instructions in the system.

Step6: The estimated LOC were divided by productivity factor to get required technical man-days. The result of this division was again divided by results of step 4 to determine total number of man-days required for the project.

Step 7: The determination of the manpower loading and schedule by finding the square root of the required man-months. The productivity factors in Kustanowitz model are defined on the basis of experience, environment and programming language [36]. Authors in [49] suggested that the productivity factor for real time system is 48 to 96 instruction/ man month. [36] Used 100 as productivity factor for mathematical systems. The general form of the expression to compute Cost and time by Kustanowitz model is given as:

Total Cost = MM (Labor rate per year/12)

Where MM is the effort in man month which is obtained by dividing the *number of instructions* by the appropriate *productivity factor.* The expected development time in man months is the square root of Man months.

Expected Development Time = $(MM)^{1/2}$

3.3 Machine learning Models

Most techniques about software cost estimation use statistical methods, which are not able to present reason and strong results. Machine learning approaches could be appropriate at this filed because they can increase the accuracy of estimation by training rules of estimation and repeating the run cycles [50]. Machine learning deals with the issue of how to build computer programs that improve their performance at some task through experience [51]. Machine learning algorithms have been utilized in:

(1) data mining problems where large databases may contain valuable implicit regularities that can be discovered automatically;
(2) poorly understood domains where humans might not have the knowledge needed to develop effective algorithms;
(3) domains where programs must dynamically adapt to changing conditions [51]

3.3.1 Fuzzy Method

All systems, which work based on the fuzzy logic try to simulate human behavior and reasoning. In many problems, where decision making is very difficult and conditions are vague, fuzzy systems are an efficient tool in such situations. This technique always supports the facts that may be ignored. There are four stages in the fuzzy approach:
Stage 1: Fuzzification: to produce trapezoidal numbers for the linguistic terms.
Stage 2: to develop the complexity matrix by producing a new linguistic term.
Stage 3: to determine the productivity rate and the attempt for the new linguistic terms.
Stage 4: Defuzzification: to determine the effort required to complete a task and to compare the existing method. [50].

3.3.2 CASE BASED REASONING

Case based reasoning is a method of storing observations on previous projects. Such as effort required to implement the project, programming platform and so forth. When faced with a new observation it is then possible to retrieve the closest stored observations[10].

ASSESSMENT OF CBR SYSTEMS

The performance of CBR systems have shown encouraging results. The performance of the CBR system exceeded both algorithmic models and was extremely close to the level of the expert. As CBR systems use stored cases, it is apparent that as observations are built up the CBR system would have a large knowledge base to compare new observations against. This would suggest CBR systems will get more accurate than experts, as experts have an inability to recall all of their experience and are potentially biased [10].

Advantages

No expert is required.
The CBR process is more akin to human thinking.
They can handle failed cases (i.e. those case for which an accurate prediction was not made).

Drawbacks

Case data can be hard to gather.
Predictions are limited to the cases that have been observed.
Case-base reasoning systems are intended to mimic the process of an expert making decisions based on their experience. The CBR technique is therefore very similar to the analogy software cost estimation technique [10].

3.3.3 RULE-BASED SYSTEMS

Rule-based systems have been used in very few cases of effort prediction. A rule-based system compares facts about the new project that are stored in a knowledge base against a stored set of rules. When a match is made the rule will fire, which can create a chaining effect with one rule enabling another rule to fire. The process continues until there is no more rules to fire and the output is presented to the user. A typical rule for the rule-base might look like this[10]

IF module_size > 100 AND interface_size > 5 THEN
    error_prone_module

IF error_prone_module AND developer experience < 2 THEN
redesign_required

If the first rule fires above then this will enable a new fact the error_prone_module. The new fact can then be used as a premise for the second rule. Thus creating a chaining effect.

Assessment of Rule-Based Systems

Rule-based systems are at a disadvantage, compared to fuzzy systems. As there is no degree of truth involved. All input variables must be either true or false.

Advantages

Simplicity of input variables.

Drawbacks:

Difficult to derive rules.
No degree of truth.
Very few cases of being used for effort prediction.
Hard to maintain a complex rule-base[10].

3.3.4 REGRESSION TREES

There have been a number of attempts to use regression trees in effort prediction [29]. A regression tree is created by examining what attributes can classify the data and then an algorithm constructs the tree, splitting nodes where appropriate. The example is very simple as the tree only ever splits into two leaf nodes, this is the most common format as it is easy to comprehend and check for logical errors[10].

Assessment of Regression Trees

Advantages

Can resist data outliers.
Can be adjusted for new data.
Easy to understand.
Suitable for non-numerical variables.

Drawbacks

Sensitive to the algorithm used to construct the tree and tree depth.
Cannot eliminate a value outside the range supplied in the training data

3.3.5 Artificial Neural Networks (ANNs)

ANNs are parallel systems inspired by the architecture of biological neural networks, comprising simple interconnected units, called artificial neurons. The neuron computes a weighted sum of its inputs and generates an output activated by a stepped function. This output is excitatory (positive) or inhibitory (negative) input fed to other neurons in the network. Feed-forward multi-layer perceptron are the most commonly used form of ANN, although many more neural network architectures have been proposed. The ANN is initialised with random weights. The network then 'learns' the relationships implicit in a data set by adjusting the weighs when presented with a combination of inputs and outputs that are known as the training set. Studies concerned with the use of ANNs to predict software development effort focus on comparative accuracy with algorithmic models, rather than on the suitability of the approach for building software effort prediction systems[28][20].A number of experiments were needed to determine an appropriate number of hidden nodes. Other parameters such as learning rate were set to values chosen by experience and experimentation. It was found that nearby values for parameters only slightly affected convergence performance [7]. Neural networks include several layers which each layer is composed of several elements called neuron. Neurons, by investigating the weights defined for inputs, produce the outputs. Outputs will be the actual effort, which is the main goal of estimation. Back propagation neural network is the best selection for software estimation problem because it adjusts the weights by comparing the network outputs and actual results. In addition, training is done effectively. Majority of researches on using the neural networks for software cost estimation, are focused on modelling the COCOMO method [50].

3.3.6 Genetic Programming

The software effort estimation problem may be modelled as a symbolic regression problem, which means, given a number of sets of data values for a set of input parameters and one output parameter, construct an expression of the input parameters which best predicts the value of the output parameter for any set of values of the input parameters. For example, a nearest neighbour (NN) system needs parameters to decide the weight of the variables and the method for determining closeness and combining neighbours to form a solution. Koza [21] lists the number of control parameters for genetic programming as being 20 as compared to 10 for neural networks, but it is not always easy to count what constitutes a control parameter. However, it is not so much the number of the parameters as the sensitivity of the accuracy of the solutions to their variation. It is possible to search using different parameters but this will depend on their range and the granularity of search. In order to determine the ease of configuration for a genetic program, we test empirically whether parameter values suggested by Koza[21] offer sufficient guidance to locate suitably accurate estimators[24].

3.3.7 SUPPORT VECTOR MACHINE (SVM)

Support vector machine constructs a hyper plane or set of hyper planes in a high or infinite dimensional space, which can be used for classification, regression, or other tasks. The main purpose of SVM is to differentiate two different types of samples, meanwhile insuring classification of the largest interval and the smallest error rate, [55]. SVM has a strong advantage in assessment, through selecting a appropriate parameter C, ε, constructing a greatest and optimal hyper-plane to contains as much support vector as possible, thus avoiding the local optimal solution problem of neural network and ensuring the generalization ability of SVM, [52]. A step in SVM classification involves identification as which are intimately connected to the known classes. This is called feature selection or feature extraction. Feature selection and SVM classification together have a use even when prediction of unknown samples is not necessary. They can be used to identify key sets which are involved in whatever processes distinguish the classes. The major strengths of SVM are the training is relatively easy. No local optimal, unlike in neural networks. It scales relatively well to high dimensional data and the trade-off between classifier complexity and error can be controlled explicitly. In the case of support vector machines, a data point is viewed as a p-dimensional vector (a list of p numbers), and we want to know whether we can separate such points with a $(p - 1)$ dimensional hyper plane. This is called a linear classifier. Intuitively, a good separation is achieved by the hyper plane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier. SVM can estimate the functional dependence of the variable y on a set of independent variables x. It presumes, that the relationship between the independent and dependent variables is given by a deterministic function f plus the addition of some additive noise [53] $y=f(x)+noise$

3.3.8 K-NEAREST NEIGHBORS' (KNN)

The k-nearest neighbors' algorithm (k-NN) is a method for classifying objects based on closest training examples in the feature space. k-NN is a type of instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until classification. The k-nearest neighbor algorithm is amongst the simplest of all machine learning algorithms: an object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its k nearest neighbors (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of its nearest neighbor. The same method can be used for regression, by simply assigning the property value for the object to be the average of the values of its k nearest neighbors. It can be useful to weight the contributions of the neighbors, so that the nearer neighbours contribute more to the average than the more distant ones. (A common weighting scheme is to give each neighbor a weight of 1/d, where d is the distance to the neighbor. This scheme is a generalization of linear interpolation.). The neighbors are taken from a set of objects for which the correct classification (or, in the case of regression, the value of the property) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required [54]. The k nearest neighbor algorithm is sensitive to the local structure of the data. Nearest neighbour rules in effect compute the decision boundary in an implicit manner. It is also possible to compute the decision boundary itself explicitly, and to do so in an efficient manner so that the computational complexity is a function of the boundary complexity. The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples. In the classification phase, k is a user-defined constant, and an unlabelled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point. Usually Euclidean distance is used as the distance metric; however this is only applicable to continuous variables [53][54].

3.3.9 Bayesian belief networks (BBN)

A BBN is a graphical network that represents probabilistic relationships among variables. BBNs enable reasoning under uncertainty and combine the advantages of an intuitive visual representation with a sound mathematical basis in Bayesian probability. With BBNs, it is possible to articulate expert beliefs about the dependencies between different variables and to propagate consistently the impact of evidence on the probabilities of uncertain outcomes, such as 'future system reliability' [33]. BBNs allow an injection of scientific rigour when the probability distributions associated with individual nodes are simply 'expert opinions'. A BBN will derive all the implications of the beliefs that are input to it; some of these will be facts that can be checked against the project observations, or simply against the experience of the decision makers themselves. There are many advantages of using BBNs, the most important being the ability to represent and manipulate complex models that might never be implemented using conventional methods. Because BBNs have a rigorous, mathematical meaning there are software tools that can interpret them and perform the complex calculations needed in their use [33]. Bayesian Belief Networks (BBN) prove to be a very useful approach to the software defect prediction problem. A BBN represents the *joint probability distribution* for a set of variables. This is accomplished by specifying (a) a directed acyclic graph (DAG) where nodes represent variables and arcs correspond to *conditional independence* assumptions (causal knowledge about the problem domain), and (b) a set of local conditional probability tables (one for each variable) [47, 51]. A BBN can be used to infer the probability distribution for a target variable (e.g., "Defects Detected"), which specifies the probability that the variable will take on each of its possible values (e.g., "very low", "low", "medium", "high", or "very high" for the variable "Defects Detected") given the observed values of the other variables. In general, a BBN can be used to compute the probability distribution for any subset of variables given the values or distributions for any subset of the remaining variables. When using a BBN for a decision support system such as software defect prediction, the steps below should be followed.

1. Identify variables in the BBN. Variables can be: (a) *hypothesis* variables for which the user would like to find out their probability distributions (hypothesis variable are either unobservable or too costly to observe), (b) *information* variables that can be observed, or (c) *mediating* variables that are introduced for certain purpose (help reflect independence properties, facilitate acquisition of conditional probabilities, and so forth). Variables should be defined to reflect the life-cycle activities (specification, design, implementation, and testing) and capture the multi-facet nature of software defects (perspectives from size, testing metrics and process quality). Variables are denoted as nodes in the DAG.
2. Define the proper causal relationships among variables. These relationships also should capture and reflect the causality exhibited in the software life-cycle processes. They will be represented as arcs in the corresponding DAG.
3. Acquire a probability distribution for each variable in the BBN. Theoretically well- founded probabilities, or frequencies, or subjective estimates can all be used in the BBN. The result is a set of conditional probability tables one for each variable. The full joint probability distribution for all the defect-centric variables is embodied in the DAG structure and the set of conditional probability tables.

3.3.10 Instance-Based Learning (IBL)

How to estimate the cost for a software project is a very important issue in the software project management. Most of the existing work is based on algorithmic models of effort [60]. A viable alternative approach to the project effort prediction is instance-based learning. IBL yields very good performance for situations where an algorithmic model for the prediction is not possible. In the framework of IBL, the prediction process can be carried out as follows.

1. Introduce a set of features or attributes (e.g., number of interfaces, size of functional requirements, development tools and methods, and so forth) to characterize projects. The decision on the number of features has to be judicious, as this may become the cause of the *curse of dimensionality* problem that will affect the prediction accuracy.
2. Collect data on completed projects and store them as instances in the case base.
3. Define *similarity* or *distance* between instances in the case base according to the symbolic representations of instances (e.g., Euclidean distance in an n-dimensional space where n is the number of features used).
4. Given a query for predicting the effort of a new project, use an algorithm such as *K- Nearest Neighbour*, or, *Locally Weighted Regression* to retrieve similar projects and use them as the basis for returning the prediction result.

4 PROPOSED METHOD

Dataset

The data set provides 8 physical characteristics of abalones (sex, length, diameter, height, whole weight, shucked weight, viscera weight, shell weight) and the number of shell rings. The age of the abalone is (number of rings + 1.5) years. The number of rings vary from 1 to 29.There are a total of 4177 samples.

Holdback Method

Holdback is a validation method that randomly divides the original data into the training and validation data sets. The Validation Portion on the platform launch window is used to specify the proportion of the original data to use as the validation data set (holdback).

Kfold Method

Divides the original data into K subsets. In turn, each of the K sets is used to ratify the model fit on the rest of the data, fitting a total of K models. The model giving the best validation statistic is chosen as the final model. KFold validation can be used only with the Decision Tree method. This method is best for small data sets, because it makes efficacious use of limited amounts of data. However, when used as a stopping rule, KFold tends to split too many times, thereby over fitting.

For abalone data set

Table 4.1 comparison of K-fold and Holdback Cross validation methods with different parameters

Method	No. of tours	RSquare	RMSE	Mean Abs Dev	Log Likelihood	SSE	Sum Freq
K-fold	1	0.6118789	2.0669044	1.4715815	1791.0671	3567.1982	835
Hold back	1	0.6375238	1.9231054	1.3904381	866.46377	1545.9038	418

In the above, comparing the two methods with RMSE and SSE parameters, it can conclude that Holdback is an efficient method.

Table 4.2 Parameter values of different datasets

S.no	datasets	features	Instance	RSquare	RMSE	Mean Abs Dev	Log Likelihood	SSE	Sum Freq
1	treasury	15	1050	0.9963585	0.2004213	0.1328788	-65.9383	14.059043	350
2	house	16	22784	0.4875257	36879.557	20180.769	90629.463	1.033	7594
3	Wizmir	9	1463	0.9941817	1.0994879	0.7777274	737.21237	588.72151	487
4	plastic	3	1651	0.8067171	1.5021119	1.2062271	1004.1958	1240.9871	550
5	pole	26	12000	0.8986296	13.278537	6.782505	16020.35	705278.14	4000
6	ailerons	40	13750	0.8501247	0.0001555	0.0001155	-33684.44	0.0001108	4583
7	California	8	20640	0.7117987	61752.02	44274.612	85654.765	2.624	6880
8	concrete	8	1030	0.8548478	6.4026876	4.8515426	1126.8258	14102.076	344
9	baseball	16	337	0.6990982	622.47446	427.77019	887.34845	437845	113
10	evaluator	18	16599	0.8919414	0.002193	0.0016347	-026024.71	0.0266097	5533

Table 4.3 sample dataset abalone

Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
3	0.4	0.305	0.1	0.3415	0.176	0.0625	0.0865	7
2	0.635	0.5	0.15	1.376	0.6495	0.361	0.31	10
3	0.37	0.27	0.09	0.1855	0.07	0.0425	0.065	7
1	0.68	0.54	0.155	1.534	0.671	0.379	0.384	10
3	0.375	0.285	0.09	0.2545	0.119	0.0595	0.0675	6
2	0.58	0.475	0.155	0.974	0.4305	0.23	0.285	10
2	0.525	0.38	0.14	0.6065	0.194	0.1475	0.21	14
2	0.63	0.495	0.19	1.1655	0.536	0.2115	0.1625	10
1	0.565	0.44	0.125	0.802	0.3595	0.1825	0.215	9
1	0.605	0.465	0.165	1.056	0.4215	0.2475	0.34	13
3	0.215	0.15	0.03	0.0385	0.0115	0.005	0.01	5
2	0.58	0.45	0.235	1.071	0.3	0.206	0.395	14
1	0.5	0.4	0.125	0.5975	0.27	0.1275	0.166	9
3	0.61	0.49	0.16	1.1545	0.5865	0.2385	0.2915	11
2	0.565	0.45	0.175	1.2365	0.5305	0.2455	0.308	10
3	0.475	0.36	0.11	0.492	0.211	0.11	0.15	8
2	0.45	0.35	0.125	0.4435	0.185	0.09	0.145	11
2	0.38	0.29	0.105	0.257	0.099	0.051	0.085	10
3	0.5	0.37	0.115	0.5745	0.306	0.112	0.141	7
1	0.59	0.485	0.155	1.0785	0.4535	0.2435	0.31	9

5 RESULTS AND DISCUSSION

A Neural Network is a function of a set of derived inputs, called hidden nodes. The hidden nodes are nonlinear functions of the original inputs. You can stipulate up to two layers of hidden nodes, with each layer containing as many hidden nodes as you want. Neural shows a two-layer neural network with three X variables and one Y variable. The functions applied at the nodes of the hidden layers are called activation functions. The activation function is a transformation of a linear combination of the X variables. For more details about the activation functions, the function applied at the response is a linear combination (for continuous responses), or a logistic transformation (for nominal or ordinal responses).The main advantage of a neural network model is that it can efficiently model different response surfaces. Given enough hidden nodes and layers, any surface can be verged on to any incisiveness.

In this example, the first layer has eight nodes, and each node is a function of all three nodes in the second layer. The second layer has three nodes, and all nodes are a function of the eight X variables. The predicted Y variable is a function of both nodes in the first layer.

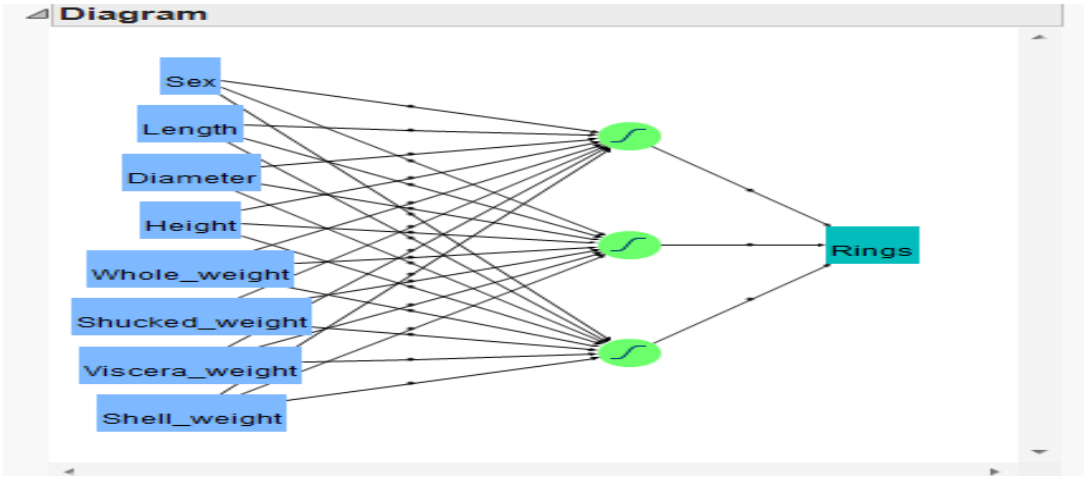


Figure 5.1 Neural Network Diagram of abalone dataset

Launching the Neural platform is a two-step process. First, enter your variables on the neural launch window. Second, specify your options in the Model Launch

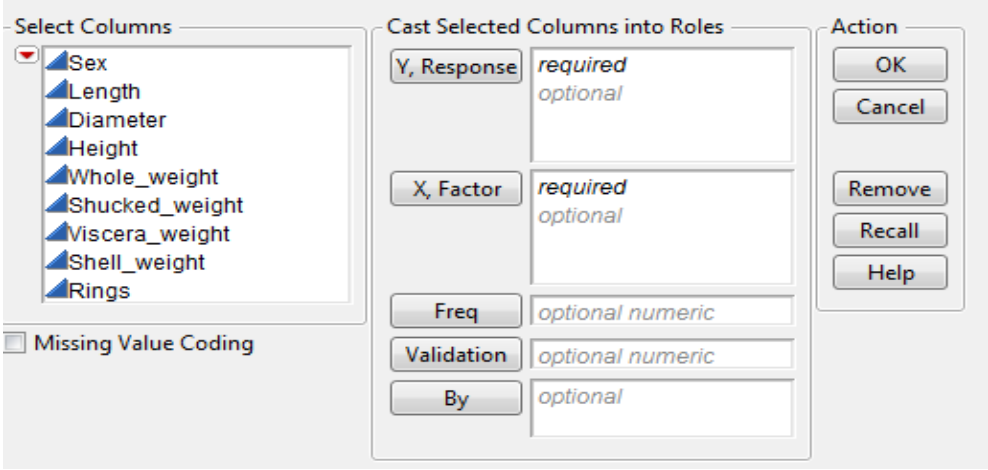


Figure 5.2 The Neural Launch Window of abalone dataset

Use the Neural launch window to specify X and Y variables, a validation column, and to enable missing value coding. Y, Response Choose the response variable. When multiple responses are specified, the models for the responses share all parameters in the hidden layers and X, Factor Choose the input variables.

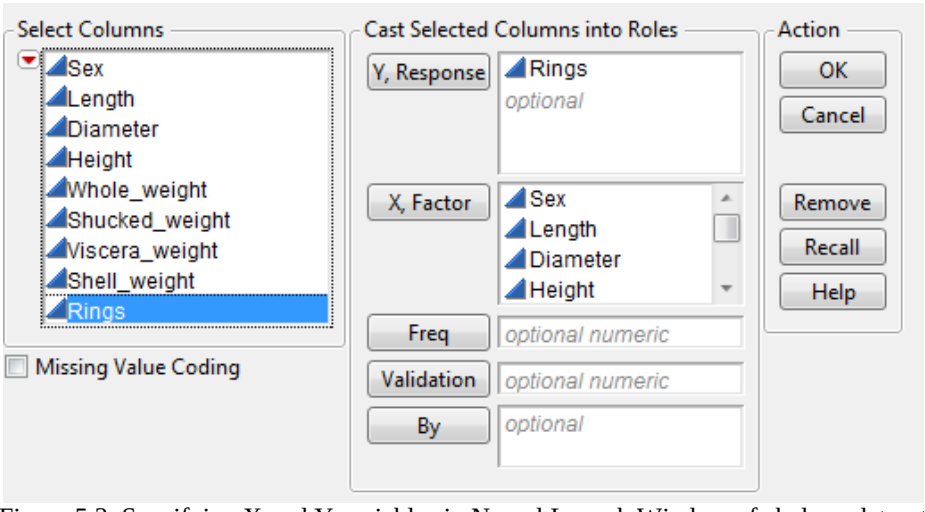


Figure 5.3 Specifying X and Y variables in Neural Launch Window of abalone dataset

The Model Launch window is used to specify the validation method, the structure of the hidden layer, whether to use gradient boosting. Validation Method, Select the method that you want to use for model validation. The different validation methods are 1).K-hold 2).Holdback 3). Excluded Rows. Holdback method randomly divides the original data into the training and validation sets. Excluded Rows uses row states to subset the data. Rows that are unexcluded are used as the training set, and excluded rows are used as the validation set. K-Fold method divides the original data into K subsets. In turn, each of the K sets is used to validate the model fit on the rest of the data, fitting a total of K models. The model giving the best validation statistic is chosen as the final model. This method is best for small data sets, because it makes efficient use of limited amounts of data.

Hidden Layer Structure, Specify the number of hidden nodes of each type in each layer and Boosting, Specify options for gradient boosting. The functions applied at the nodes of the hidden layers are called activation functions. An activation function is a transformation of a linear combination of the X variables. Figure 5.10 Actual by Predicted Plot of abalone dataset describes the three types of activation functions: 1). TanH 2). Linear 3). Gaussian TanH is a sigmoid function. TanH transforms values to be between -1 and 1, and is the centered and scaled version of the logistic function. The hyperbolic tangent function is: e^{2x}-1 / e^{2x} +1 where x is a linear combination of the X variables. The Linear activation function is most often used in conjunction with one of the non-linear activation functions. This function is placed in the second layer, and the non-linear activation functions are placed in the first layer. This is useful if you want to first reduce the dimensionality of the X variables, and then have a nonlinear model for the Y variables. For a continuous Y variable, if only linear activation functions are used, the model for the Y variable reduces to a linear combination of the X variables. For a nominal or ordinal Y variable, the model reduces to a logistic regression. The Gaussian function is used for radial basis function behavior, or when the response surface is Gaussian (normal) in shape. The Gaussian function is: e^{-x2} where x is a linear combination of the X variables.

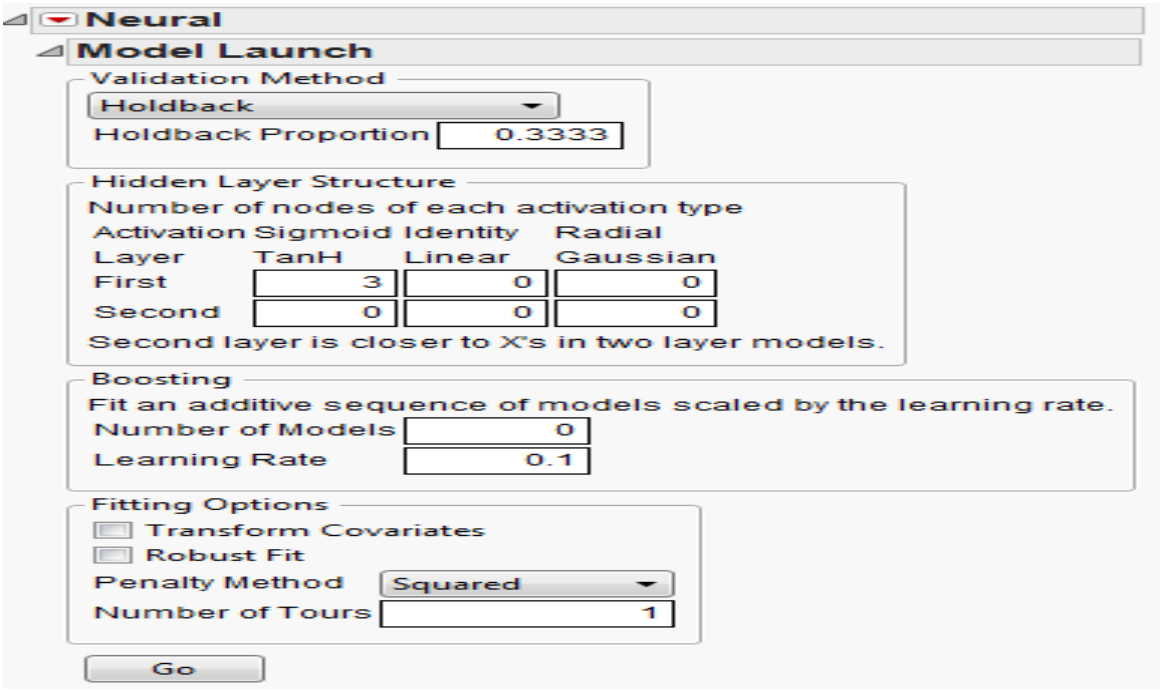


Figure 5.4 The Neural model Launch Window of abalone dataset

A model report is created for every neural network model. Measures of fit appear for the training and validation sets.

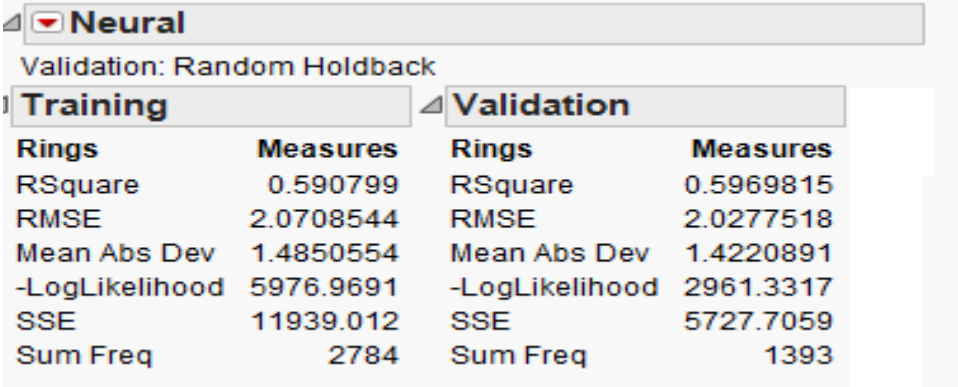


Figure 5.5 Model Report of abalone dataset

A generalization of the Rsquare measure that simplifies to the regular Rsquare for continuous responses. Entropy RSquare, Compares the log-likelihoods from the fitted model and the constant probability model. Appears only when the response is nominal or ordinal. Rsquare gives the Rsquare for the model. RMSE, Gives the root mean square error. When the response is nominal or ordinal, the differences are between 1 and p (the fitted probability for the response level that actually occurred).Mean Abs Dev, The average of the absolute values of the differences between the response and the predicted response. When the response is nominal or ordinal, the differences are between 1 and p (the fitted probability for the response level that actually occurred).Log Likelihood, Gives the negative of the log likelihood.SSE, Gives the error sums of squares. Available only when the response is continuous. Sum Freq, Gives the number of observations that are used. If you specified a Freq variable in the neural launch window, Sum Freq gives the sum of the frequency column.

For nominal or ordinal responses, each response level is represented by a separate row in the Prediction Profiler. The following figure 5.6 shows prediction profiler

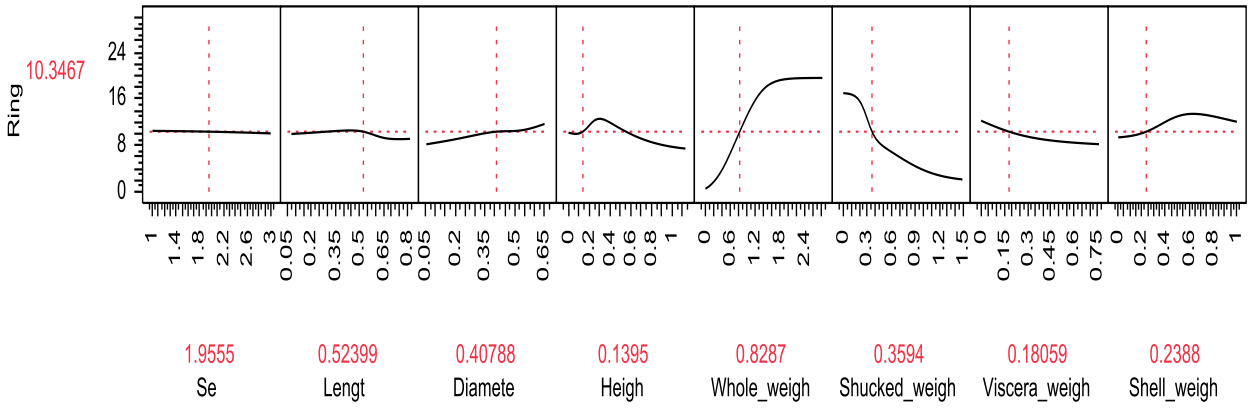


Figure 5.6 Prediction profiler of abalone dataset

Contour Profiler

This is available only when the model contains more than one continuous factor. The following figure 5.7 shows contour profiler

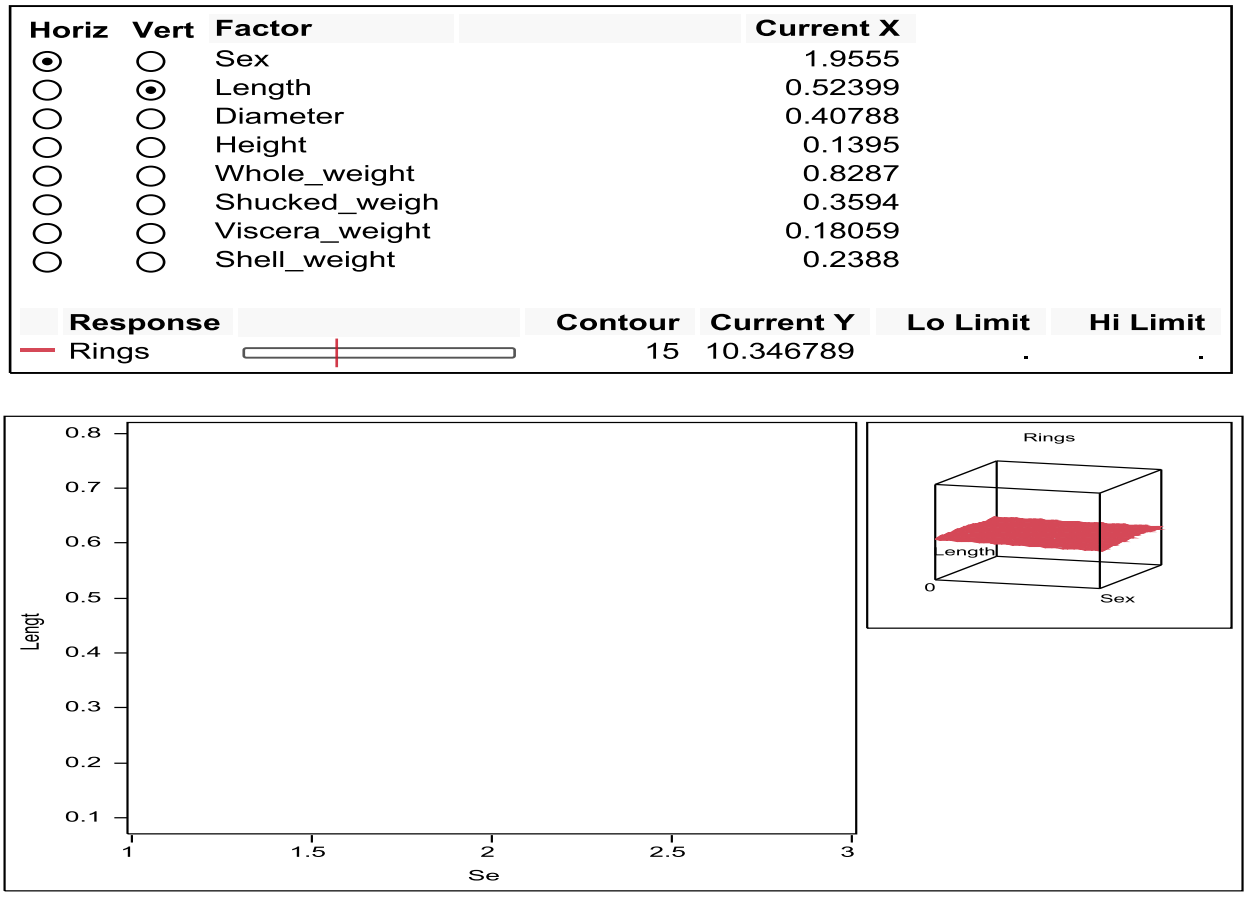


Figure5.7 Contour profiler of abalone dataset

The following figures 5.8 and 5.9 Shows the parameter estimates and rings in a report.

Estimates

Parameter	**Estimate**
H1_1:Sex	-0.05038
H1_1:Length	3.249323
H1_1:Diameter	1.495004
H1_1:Height	-3.96622
H1_1:Whole_weight	4.044512
H1_1:Shucked_weight	-3.60998
H1_1:Viscera_weight	0.415567
H1_1:Shell_weight	-2.48174
H1_1:Intercept	-2.26872
H1_2:Sex	-1.13022
H1_2:Length	20.54197
H1_2:Diameter	19.81917
H1_2:Height	13.17524
H1_2:Whole_weight	-4.18745
H1_2:Shucked_weight	-5.65906
H1_2:Viscera_weight	3.068512
H1_2:Shell_weight	-4.92809
H1_2:Intercept	-13.6171
H1_3:Sex	-0.40922
H1_3:Length	-3.43777
H1_3:Diameter	5.715831
H1_3:Height	19.32787
H1_3:Whole_weight	4.506278
H1_3:Shucked_weight	-19.0967
H1_3:Viscera_weight	-6.1584
H1_3:Shell_weight	8.459773
H1_3:Intercept	-1.40558
Rings_1:H1_1	5.166962
Rings_2:H1_2	-1.16174
Rings_3:H1_3	3.786147
Rings_4:Intercept	9.489055

Figure5.8 Parameter estimates of abalone dataset

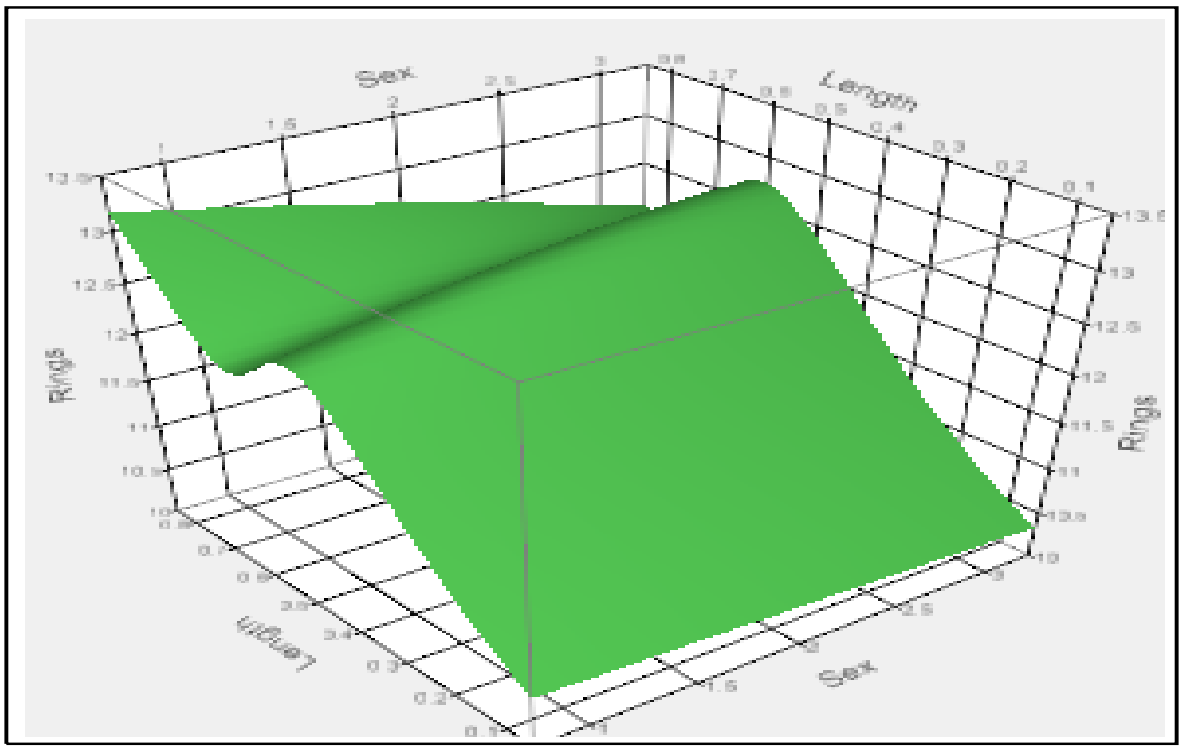


Figure5.9 Rings of abalone dataset

Actual by Predicted Plot

The following 5.10 shows Plots the actual versus the predicted response. Available only for continuous responses.

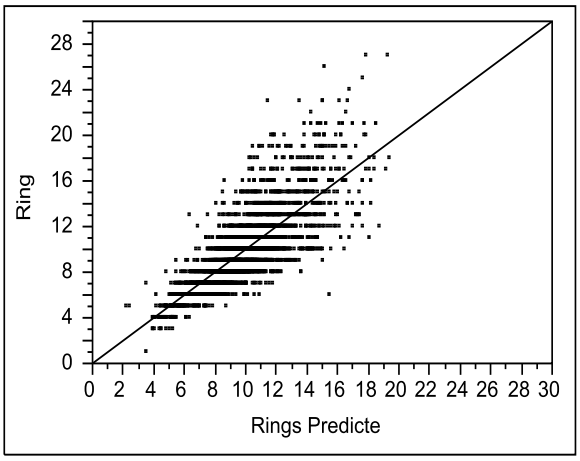
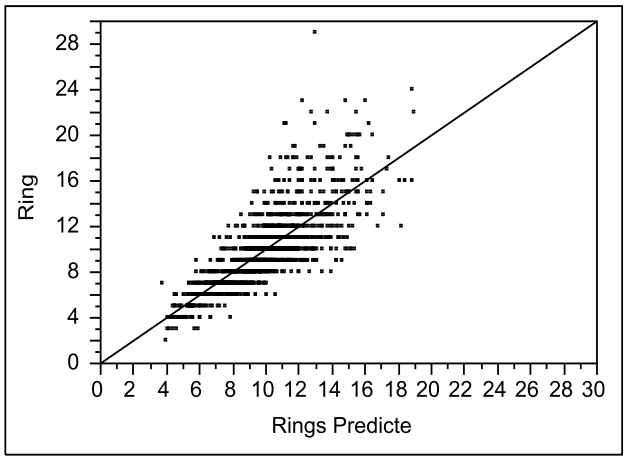


Figure 5.10 Actual by Predicted Plot of abalone dataset



Residual by Predicted Plot

The following figure shows Plots the residuals versus the predicted responses. Available only for continuous responses.

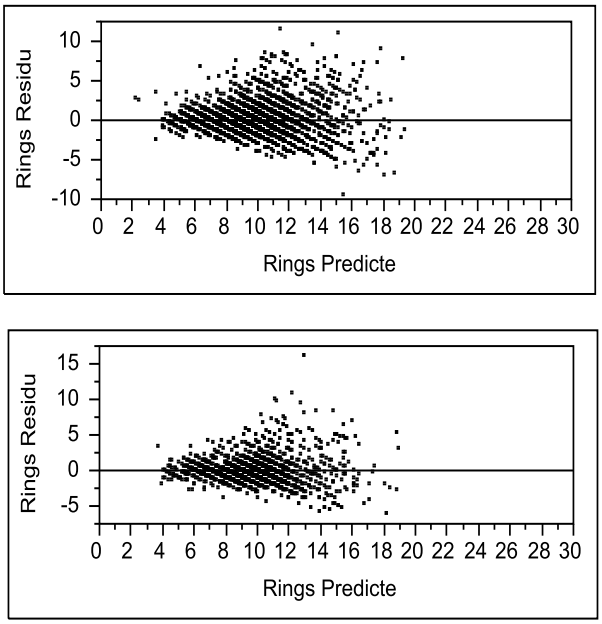


Figure 5.11 Residual by Predicted Plot of abalone dataset

6 CONCLUSIONS

The accurate and reliable estimation of effort is very important for software project development. In this paper, we have constructed an enhanced model of Holdback through neural network. The model will take the advantage of Holdback for effort estimation and neural network for learning capability. Here two most popular approaches were suggested to predict the software effort estimation. One is the K-fold cross validation method which has been already proven and successfully applied in the software effort estimation field and the other is Holdback cross validation algorithm in neural network that has been extensively used in lieu of cost estimation and have demonstrated their strength in predicting problem. Final results were shown and also calculated the root mean square error, sum of square error. The use of artificial neural network algorithm for the proposed model and the cost estimation algorithms are an efficient to find the values of project estimates.

REFERENCES

[1] G.N. Parkinson, ―Parkinson‘s Law and Other Studies in Administration‖, Houghton-Miffin, Boston, 1957.

[2] R.K.D. Black, R. P. Curnow, R. Katz and M. D. Gray, BCS Software Production Data, Final Technical Report, RADC-TR-77-116, Boeing Computer Services, Inc., March 1977.

[3] L.H. Putnam, ―A general empirical solution to the macro software sizing and estimation problem,‖ IEEE Transactions on Software Engineering, pp. 345–361, , July 1978.

[4] B.W. Boehm, ―Software Engineering Economics,‖ Prentice- Hall, Englewood Cliffs, NJ, USA, 1981.

[5] A.J. Albrecht and J.E. Gaffney, ―Software function, source lines of code, and development effort prediction: a software science validation,‖ IEEE Transactions on Software Engineering, , pp. 639–647, 1983.

[6] Chris F.K, ―An Empirical Validation of Software Cost Estimation Models‖, Management Of Computing-Communications of ACM, Vol: 30, No. 5, pp.416-429, , May 1987.

[7] Martin Lefley and Martin J. Shepperd "Using Genetic Programming to Improve Software Effort Estimation Based on General Data Sets" E. Cantú-Paz et al. (Eds.): GECCO 2003, LNCS 2724, pp. 2477–2487, 2003.© Springer-Verlag Berlin Heidelberg 2003

[8] Liming Wu ―The Comparison of the Software Cost Estimating Methods‖ University of Calgary.

[9] B. W. Boehm, Early experiences in software economics, Springer-Verlag New York, Inc. New York, 2002.

[10] Abdulbasit S. Banga "Software Estimation Techniques" Proceedings of National Conference; INDIACom-2011Computing for Nation Development, March 10 – 11, 2011,BharatiVidyapeeth University Institute of Computer Applications and Management, New Delhi

[11] COCOMO II Model definition manual‖, version 1.4, University of Southern California.

[12] S. Devnani-Chulani, "Bayesian Analysis of Software Cost and Quality Models‖. Faculty of the Graduate School, University Of Southern California May 1999.

[13] http://sunset.usc.edu/cse/pub/research/AgileCOCOMO/AgileCOCOMOII/Main.htm

[14] Y. Singh, K.K. Aggarwal, Software Engineering Third edition, New Age International Publisher Limited New Delhi.

[15] C. J. Burgess and M. Lefley, "Can genetic programming improve software effort estimation? A comparative evaluation," *Information & Software Technology*, vol. 43, pp. 863–873, 2001

[16] MuraliChemuturi, **"**Analogy based Software Estimation,**"** Chemuturi Consultants.

[17] S. Vicinanza, M. J. Prietula, and T. Mukhopadhyay, "Case-based reasoning in effort estimation," presented at 11th Intl. Conf. on Info. Syst., 1990

[18] O. Benediktsson and D. Dalcher, ―Effort Estimation in incremental Software Development,‖ *IEEE Proc. Software*, Vol. 150, no. 6, pp. 351-357, December 2003.

[19] R. Bisio and F. Malabocchia, "Cost estimation of software projects through case base reasoning," presented at 1st Intl. Conf. on Case-Based Reasoning Research & Development,1995.

[20] M. J. Shepperd and C. Schofield, "Estimating software project effort using analogies, "*IEEE Transactions on Software Engineering*, vol. 23, pp. 736–743, 1997.

[21] J. R. Koza, *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press, 1992.

[22] Caper Jones, ―Estimating software cost‖ tataMc- Graw -Hill Edition 2007.

[23] Magne J and Martin S, ― A Systematic Review of Software Development Cost Estimation Studies ‖, IEEE Transactions On Software Engineering, Vol. 33, No. 1, pp. 33-53, January 2007.

[24] J. J. Dolado, "On the problem of the software cost function," *Information & Software Technology*, vol. 43, pp. 61–72, 2001

[25] K. Vinaykumar, V. Ravi, M. Carr and N. Rajkiran, ―Software cost estimation using wavelet neural networks,Journal of Systems and Software, pp. 1853-1867, , 2008.

[26] Pressman. Software Engineering - a Practitioner‘s Approach. 6th EddditionMcGraw Hill international Edition, Pearson education, ISBN 007 - 124083 – 7.

[27] E. Stensrud and I. Myrtveit, "Human performance estimating with analogy and regression models: an empirical validation," presented at 5th Intl. Metrics Symp., Bethesda, MD,1998.

[28] G. Wittig and G. Finnie, "Estimating software development effort with connectionists models," *Information & Software Technology*, vol. 39, pp. 469–476, 1997.

[29] K. Srinivasan and D. Fisher, "Machine Learning Approaches to Estimating Software Development Effort," IEEE

Transactions on Software Engineering, vol.21, Feb.1995.
[30] M.V. Deshpande and S.G. Bhirud, ―Analysis of Combining Software Estimation Techniques, *International Journal of Computer Applications* (0975 – 8887) Volume 5 – No.3, 2010 .
[31] SwetaKumari ,ShashankPushkarPerformance Analysis of the Software Cost Estimation Methods: A Review International Journal of Advanced Research in Computer Science and Software Engineering Volume 3, Issue 7, July 2013 ISSN: 2277 128X
[32] E. E. Mills, "Software Metrics," SEI Curriculum Module SEI-CM- 12-1.1, Carnegie Mellon University, Pittsburgh, 1988
[33] Martin Neil & Norman Fenton Predicting Software Quality using Bayesian Belief Networks Proceedings of 21st Annual Software Engineering Workshop NASA/Goddard Space Flight Centre, December 4-5, 1996.
[34] J. J. Bailey and V. R. Basili, "A meta-model for software development resource expenditures," in Proc. 5th Int. Conf. Software Eng., IEEE/ACM/NBS, 1981, pp. 107-116.
[35] B. Londeix, Cost estimation for software development, Addison-Wesley, 1987.
[36] S. N. Mohanty, "Software cost estimation: Present and future," Software—Practice and Experience Vol.11, no.2, pp.103–121, 1981.
[37] B. W. Boehm, "Software Engineering Economics,",IEEE trans. On soft. Eng. Vol.10, no.1, pp 7-19, 1984
[38] E. A. Nelson, Management handbook for the estimation of computer programming costs, Systems Development Corporation, AD-A648750 , Oct. 1966.
[39] D. D. Galorath and D. J. Reifer, "Final Report: Analysis of the state of the Art of Parametric Software Cost Modeling," software management consultants, California, Tech. Rep. Aug. 1980
[40] H. Leung and Z. Fan, Software Cost Estimation Handbook of Software Engineering, Hong Kong Polytechnic University, 2002.
[41] N. B. Armstrong, "Software Estimating: A Description and Analysis of Current Methodologies with Recommendation on Appropriate Techniques For Estimating RIT Research Corporation Software Projects," M. Eng. Thesis, Wallace Memorial Library, RIT, Dec. 1987.
[42] J. N. Buxton and B. Randell, "Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee," Scientific Affairs Division, NATO, tech. rep. 27-31 Oct. 1969
[43] R. W. Wolverton, "The cost of developing large-scale software," IEEE Trans. Comput., pp. 615-636, June 1974.
[44] R. B. Gradey.(1998) CS course webpages.[Online]. Available: http://courses.cs.tamu.edu/lively/606_Simmons/ Textbook/CH07_6.doc
[45] S. D. Chulani, Bayesian Analysis of the Software Cost and Quality Models. PhD thesis, faculty of the Graduate School University of Southern California, 1999
[46] C. E. Walston and C. P. Felix, "A Method of Programming measurement and Estimation," IBM Systems Journal, Vol. 16, no. 1, pp. 54-73, 1977
[47] S. D. Chulani, Bayesian Analysis of the Software Cost and Quality Models. PhD thesis, faculty of the Graduate School University of Southern California, 1999
[48] L. Farr and J. H. Zagorski, "Quantitative Analysis of Programming Cost Factors: A Progress Report," in Economics of Automatic Data Processing, ICC Symposium Proceedings, A.B. Frielink, (ed.), North-Holland, 1965.
[49] E. B. Daly, "Management of software development," in *Proc. IEEE Transactions on Software Engineering*, SE-3, no. 3, 1977..
[50] VahidKhatibi, Dayang N. A. Jawawi"Software Cost Estimation Methods: A Review"Journal of Emerging Trends in Computing and Information Sciences Volume 2 No. 1 ISSN 2079-8407
[51] T. Mitchell, *Machine Learning*, McGraw-Hill, 1997.
[52] ZhengyuanJia, Lihua Gong, Jia Han , "Application of Support VectorMachine based on Rough Sets to Project Risk Assessment (RS-SVM) "International Conference on Computer Science and SoftwareEngineering , vol. 1, pp. 508-511, 2008
[53] GeetaSikka ,ArvinderKaur, MoinUddin , " Estimating Functionpoints: Using Machine Learning and Regression Models " 2nd International Conference on Education Technology and Computer(ICETC) , vol. 3, pp. 52-56.
[54] Amanjot Singh Klair and RaminderPreetKaur Software Effort Estimation using SVM and kNN International Conference on Computer Graphics, Simulation and Modeling (ICGSM'2012) July 28-29, 2012 Pattaya (Thailand).
[55] H. Du, S. Teng and Q. Zhu, Fast SVM Incremental Learning Based onClustering Algorithm, IEEE International Conference on IntelligentComputing and Intelligent Systems, 2009, vol. 1, pp. 13-17, November2009.
[56] I. Sommerville, *Software Engineering*, Addison-Wesley, 1996.
[57] H. Zhu, "A formal analysis of the subsume relation between software test adequacy criteria," *IEEE Trans. SE*, Vol.22, No.4, April 1996, pp.248-255.
[58] N. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Trans. SE*, Vol. 25, No. 5, Sept. 1999, pp. 675-689.
[59] F.V. Jensen, *An Introduction to Bayesian Networks*, Springer, 1996.
[60] Shepperd, M., Schofield, C. Estimating software project effort using analogies. IEEE Transactions on Software Engineering, vol. 23, no. 12 (November 1997) 736-743.

