CHAPTER 7

ESTIMATION OF SOFTWARE PROJECT EFFORT USING NEURAL NETWORKS

7.1 INTRODUCTION

The most common application of software metrics is to develop models that predict the effort (often measured in person days) The estimate for the manpower that is required for software produced, is a major constituent in cost estimation. Along with schedule estimate, it determines the team size. This is needed before development is initiated. Cost estimation is the next step.

In order to properly estimate the effort and cost, different estimation models have been proposed in literature [21,139,149,166,178,189,206,208]. Various models based on mathematical functions or other techniques such as regression trees, analogy based reasoning, rule induction models could be used to predict the effort. All of these models are based on measuring certain size or function related attributes of the software and relating these measurements to the cost or effort necessary for its development. The effort estimation is based on mainly three parameters – system size, complexity and developer characteristics as shown in Fig 7.1. [7]
Figure 7.1: Software effort estimation model

Some models use the system size, the system complexity and the developer ability to predict the effort required. These models use linear regression analysis on available historical data for similar project or use other techniques like regression trees, case based reasoning etc. [190,191,192] but the major problem that exists with such models is that the project managers cannot specify the exact values of parameters used as inputs to the models. The actual value is only known after completion of the project.

In this chapter we propose an artificial neural network based model for effort estimation using four cost drivers of the Model as inputs. The COCOMO model has been chosen as a base for two reasons. First the COCOMO model is one of the most frequently referred models in cost estimation. Second, COCOMO is
based on log-linear formula considered the most plausible for software cost modeling. [37]

7.2 COCOMO MODEL

COCOMO stands for Constructive Cost Model, and is one of the most widely used software estimation models in the world. Barry Boehm developed COCOMO in 1981, and it predicts the effort and schedule for software product development based on inputs relating to the size of the software and a number of cost drivers that affect productivity [111].

7.2.1 BASIC COCOMO MODEL

Basic COCOMO model estimates the software development effort using only a single predictor variable (size in DSI) and three software development modes. The equations for the basic COCOMO model are shown below:

\[ E = a \ (KLOC)^b \] \hspace{1cm} (7.1)

\[ D = c \ (E)^d \] \hspace{1cm} (7.2)

Where E is effort applied in Person-Months, and D is the development time in months. The coefficients a, b, c, d are given in Table 7.1.
<table>
<thead>
<tr>
<th>Project</th>
<th>A</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organic</td>
<td>2.4</td>
<td>1.06</td>
<td>2.5</td>
<td>0.38</td>
</tr>
<tr>
<td>Semidetached</td>
<td>3.0</td>
<td>1.12</td>
<td>2.5</td>
<td>0.36</td>
</tr>
<tr>
<td>Embedded</td>
<td>3.6</td>
<td>1.20</td>
<td>2.5</td>
<td>0.32</td>
</tr>
</tbody>
</table>

Table 7.1: Basic COCOMO coefficients

The multipliers and exponent of the equations are different depending on the development mode of the project. Once the effort is calculated, it can be used to estimate other project attributes such as schedule. Basic COCOMO is good for quick, early, rough order of magnitude estimates of software costs. The accuracy of this model is necessarily limited because of its lack of factors, which have a significant influence on software costs. The Basic COCOMO estimates are within a factor of 1.3 only 29% of the time, and within a factor of 2 only 60% of the time.
7.2.1 INTERMEDIATE COCOMO MODEL

The Intermediate Model estimates the software development effort by using fifteen cost driver variables besides the size variable used in Basic COCOMO. The cost drivers are attributes of the project, which can be used as multipliers. These attributes are grouped by category below.

Product Attributes

- **RELY:** Required Software Reliability
  
The extent to which the software product must perform its intended functions satisfactorily over a period of time.

- **DATA:** Data Base Size
  
The degree of the total amount of data to be assembled for the database.

- **CPLX:** Software Product Complexity
  
The level of complexity of the product to be developed.

Computer Attributes

- **TIME — Execution Time Constraint**
  
The degree of the execution constraint imposed upon a software product.
• **STOR --- Main Storage Constraint**
  The degree of main storage constraint imposed upon a software product.

• **VIRT --- Virtual Machine Volatility**
  The level of the virtual machine underlying the product to be developed.

• **TURN --- Computer Turnaround Time**
  The level of computer response time experienced by the project team developing the product.

**Personnel Attributes**

• **ACAP: Analyst Capability**
  The level of capability of the analysts working on a software product.

• **AEXP: Applications Experience**
  The level of applications experience of the project team developing the software product.

• **PCAP: Programmer Capability**
  The level of capability of the programmers working on the software product.

• **VEXP: Virtual Machine Experience** The level of virtual machine experience of the project team developing the product.
• **LEXP: Programming Language Experience**

  The level of programming language experience of the project team developing the product.

**Project Attributes**

• **MODP: Use of Modern Programming Practices**

  The degree to which modern programming practices (MPPs) are used in developing software product.

• **TOOL: Use of Software Tools**

  The degree to which software tools are used in developing the software product.

• **SCED: Schedule Constraint**

  The level of schedule constraint imposed upon the project team developing the software product.

Table of Effort Multipliers for each of the Cost Drivers is provided with ranges depending on the ratings. A table is provided in [37] which explains in more detail how to determine what rating (high, very high, etc.) to give the cost drivers in your project or product. The equations for the Intermediate COCOMO model are given below:

\[ E = a(KLOC)^b \times EAF \quad \text{(7.3)} \]
\[ D = c (E)^d \] \hspace{1cm} (7.4)

The coefficients a, b, c, d are given in Table 7.2.

<table>
<thead>
<tr>
<th>Project</th>
<th>A</th>
<th>B</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organic</td>
<td>3.2</td>
<td>1.05</td>
<td>2.5</td>
<td>0.38</td>
</tr>
<tr>
<td>Semidetached</td>
<td>3.0</td>
<td>1.12</td>
<td>2.5</td>
<td>0.35</td>
</tr>
<tr>
<td>Embedded</td>
<td>2.8</td>
<td>1.20</td>
<td>2.5</td>
<td>0.32</td>
</tr>
</tbody>
</table>

Table 7.2: Intermediate COCOMO coefficients

The Intermediate Model can be applied across the entire software product easily for rough cost estimation during the early stage or it can be applied at the software product component level for more accurate cost estimation in more detailed stages. The Intermediate Model estimates are within 20% of the actual values 68% of the time [111]. Some limitations of the Intermediate model are that its effort multipliers are phase-insensitive. The Detailed COCOMO model addresses this. Another limitation is that it can be very tedious to use on a product with many components.

7.2.2 DETAILED COCOMO MODEL

Two major areas where the detailed COCOMO model is different from the intermediate model are:
Phase-sensitive Effort Multipliers

The detailed model uses different effort multipliers for every cost drivers depending on the phase of software development

- **Module-Subsystem-System Hierarchy**

  The software product is estimated in a three level hierarchical decomposition. The fifteen cost drivers are related to module or subsystem level

  The Module-Subsystem-System Hierarchy can be illustrated by the association of cost drivers to each of the levels this indicates that depending on the level that you are attempting to estimate, certain cost drivers are more applicable, and have more effect on the cost estimation. This can be summarized as:

- **Module level**

  Cost drivers tend to vary at the lowest level.

  Example: CPLX, PCAP, VEXP, LEXP

- **Subsystem Level**

  Cost drivers tend to vary from subsystem to subsystem, but are the same for modules in a sub-system.

  Example: RELY, DATA, TIME, STOR, VIRT
System Level

Overall project relations such as nominal effort and schedule equations

The Detailed Model uses the same equations for estimations as the Intermediate Model; however, it uses a more complex procedure to calculate estimation. The procedure uses the DSI's for subsystems and modules, and module level and subsystem level effort multipliers as inputs.

The Detailed Model can estimate the staffing, cost, and duration of each of the development phases, subsystems, and modules. It also allows one to experiment with different development strategies, to find the plan that best suit one’s needs and resources. Detailed Model estimates are within 20% of the actual values 70% of the time [111].

A limitation of the Detailed COCOMO model is that it requires substantially more time and effort to calculate estimates than previous models.

7.3 EXPERIMENT DESIGN

The intermediate COCOMO model by Boehm proposed fifteen predictors called cost drivers to take into account the software development process namely the system complexity and the developer characteristics as shown in Fig 7.1 [8, 37]

Of course, the size is the most important input. Out of the fifteen cost drivers, four
important cost drivers namely reliability, complexity of the software, developer’s experience, developer’s ability were considered along with size of the software as inputs to the neural network. The target output is effort. Each of these five inputs could be low, medium and high. The ranges as classified by expert opinion are shown in Table 7.3. The values of size are normalized to have values between 0 and 1. All possible combination of the five inputs was considered. These resulted in a total of two hundred and forty three cases.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>0 - 0.5</td>
<td>.51 - .7</td>
<td>.71 - 1</td>
</tr>
<tr>
<td>Complexity</td>
<td>0 - 0.4</td>
<td>.41 - .7</td>
<td>.71 - 1</td>
</tr>
<tr>
<td>Experience</td>
<td>0 - 0.2</td>
<td>.21 - .5</td>
<td>.51 - 1</td>
</tr>
<tr>
<td>Ability</td>
<td>0 - 0.3</td>
<td>.31 - .7</td>
<td>.71 - 1</td>
</tr>
<tr>
<td>Size</td>
<td>0 - 0.01 (≤ 10 K)</td>
<td>.02 - .50 (10 K - 500 K)</td>
<td>.51 - 1 (500 - 1000 K)</td>
</tr>
</tbody>
</table>

Table 7.3: Range for inputs

Using the Neural network tool in MATLAB, the neural network was trained for 115 combinations resulting in various effort values. The test cases were given in numeric format. For example if reliability is low, this should be given as any numeric value between .0-0.50. While selecting these 115 test cases, it was made sure that border values are included.
The efforts i.e. the target output of the neural network was classified into four types as shown in Table 7.4.

<table>
<thead>
<tr>
<th>Output</th>
<th>Effort</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Low</td>
<td>P1</td>
</tr>
<tr>
<td>0 1</td>
<td>Medium</td>
<td>P2</td>
</tr>
<tr>
<td>1 0</td>
<td>High</td>
<td>P3</td>
</tr>
<tr>
<td>1 1</td>
<td>V. High</td>
<td>P4</td>
</tr>
</tbody>
</table>

Table 7.4: Classification of effort

A supervised algorithm called error back propagation was used to perform the experiment.

We used a sigmoid feed forward network with a single hidden layer. There are three nodes in the input layer. In the first (hidden) layer neurons were varied from two to fifteen and in the second (output) layer two neurons were kept to represent the four target values of effort as shown in Table 7.4.

Experiment was done with an ensemble of networks starting from network with two hidden neurons to fifteen hidden neurons, to find out which of these networks gives the best performance, keeping the output layer neurons constant in each of the networks i.e. two, which represents the four target values.
The MATLAB training function used was 'trainlm', the adaptation learning function selected for this experiment was 'learnngdm'. Performance function used was mean square Error (MSE).

Transfer functions used were 'log sigmoid' in both the layers. The network with nine neurons in hidden layer was found to be most appropriate for further study, as it's performance was found to be the best. Taking the network with nine hidden neurons, experimentation was done with same parameter values except that epochs were increased from 100 to 10000 and the goal was kept as 0.05. The performance varied from 0.012 to 0.0445. The training set consisted of one hundred and fifteen exemplars, thirty-five each from P1, P2 & P3 categories and ten from P4 category. These were chosen randomly.

After training with one hundred and fifteen test cases, simulation was done on the network. The criterion for misclassification was that if the desired value was say (0,0), then any value different from (0,0) is a misclassification. Also values from 0 to 0.35 were treated as zero and similarly values from .65 to 1 were treated as 1.

Next test data for two sets A & B as shown in the next section were provided as inputs to the neural network, which was already trained with a goal of 0.05. The neural network was then simulated and the output for these cases was analysed.
7.4 RESULTS & DISCUSSION

The experiment results after training are as shown in Table 7.5 and the simulation results are summarized in Tables 7.6 and 7.7

<table>
<thead>
<tr>
<th>No. Of training cases</th>
<th>Misclassification</th>
<th>% Of misclassification</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 35</td>
<td>P2 35</td>
<td>P3 35</td>
</tr>
</tbody>
</table>

Table 7.5: Training Results (115 Cases)

<table>
<thead>
<tr>
<th>Misclassification (Mean)</th>
<th>Misclassification (Std. Dev)</th>
<th>Percentage of Misclassification (Mean)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 2</td>
<td>P2 4</td>
<td>P3 5</td>
</tr>
</tbody>
</table>

Table 7.6: Simulation Results- Set A having 5 sub sets of data (each of seventy test cases)

Simulation in Table 7.6 is done for 70 test cases 20 each of P1 P2 and P3 and 10 test cases of P4. Percentage of misclassifications is 13.77 %.

<table>
<thead>
<tr>
<th>Misclassification (Mean)</th>
<th>Misclassification (Std. Dev)</th>
<th>Percentage of Misclassification (Mean)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 2</td>
<td>P2 3</td>
<td>P3 4</td>
</tr>
</tbody>
</table>

Table 7.7: Simulation Results- Set B having 5 sub sets of data (each of hundred test cases)
Simulation in Table 7.7 is done for 100 test cases, 30 each of P1, P2, P3 and 10 test cases of P4 categories. Total percentage of misclassification is 7.5%.

After training, it was observed that misclassification occurred only in the P4 category (i.e. effort= very high) to the extent of 1%. No misclassification occurred in P1, P2 and P3 category.

After simulation with seventy test cases in set A and set B, it was observed that, maximum classification occurred in P3 category.

Simulation was done with two different sets so as to see the difference in misclassification (If any). It was found that the difference was significant. In case of simulation done with seventy exemplars the misclassification was 13.7% while with hundred exemplars it improved to 7.5%.

7.5 CONCLUSION

In this chapter, the possibility of use of neural networks for estimating effort of software projects was explored. It is observed from the experimental work, that neural networks are very well used for estimating the effort.
However to be able to utilize this technique in practice, the training data and test data must be given in a numeric format. However, at present, there is no mechanism to interpret the architecture of the neural network. This is one of the weaknesses of the neural network approach.