SOFTWARE EFFORT ESTIMATION USING ENSEMBLE OF NEURAL NETWORKS WITH ASSOCIATIVE MEMORY

by

Yiğit Kültür

B.S., Computer Engineering, Middle East Technical University, 2006

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2008

ACKNOWLEDGEMENTS

I would like to thank my supervisor Assistant Professor Ayşe B. Bener for her support and guidance throughout my research.

I would also like to thank Associate Professor Necati Aras and Professor Oğuz Tosun for kindly accepting to be in my thesis jury.

I am grateful to the members of Software Engineering Research Laboratory (SoftLab) for their friendship and support.

I would like to express my sincere gratitude to Boğaziçi University Computer Engineering faculty and assistants for strengthening my knowledge base with brilliant graduate courses.

My elementary school teacher, Mrs. Fahriye Sümer, deserves special thanks for teaching me the importance of diligence and patience. If she were not my teacher, it would have been impossible for me to reach this point.

My parents were by my side in this research as they have always been in my life. Words cannot convey how much I love them.

I would like to thank to Boğaziçi University Research Fund for supporting this research under grant number BAP-06HA104.

Finally, I would like to thank The Scientific and Technological Research Council of Turkey (TÜBİTAK) for supporting me under National Scholarship Program for MS Students.

ABSTRACT

SOFTWARE EFFORT ESTIMATION USING ENSEMBLE OF NEURAL NETWORKS WITH ASSOCIATIVE MEMORY

In software industry, most of the budget is used for project implementation. Therefore, each software company has to manage its workforce effectively. Estimating the software effort accurately is essential for workforce management.

Researchers became aware of the importance of software effort estimation in 1960's and so far they have proposed several models, some of which are learning oriented. Companies usually have a small number of completed projects and consequently limited amount of data for estimating the effort of new projects. It is hard to make accurate estimations with scarce data. As the problem and estimation methods become more complex, it becomes harder to learn effort function with small datasets. Therefore, it is important to improve the performance of the predictor for effort estimation. Many researchers have used neural networks as a single element to be a robust algorithm in software effort estimation research. In this research, we focused on improving the prediction performance of the algorithm and therefore, we used ensemble of neural networks rather than a single neural network. Furthermore, we combined associative memory with the ensemble to provide the final model. We also analyzed the effect of feature subset selection on effort estimation performance. For this purpose, the features that contain most of the important information are discovered. Thereafter, only these features are used for effort estimation on the proposed model.

The proposed model provides accurate estimations. Therefore, software companies may use this model to estimate software effort and effectively manage their workforce. On the other hand, the results of our experiments showed that using fewer features may provide an improvement on the prediction performance.

ÖZET

SİNİR AĞI TOPLULUĞU İLE ÇAĞRIŞIMLI BELLEK KULLANARAK YAZILIM EFOR TAHMİNİ

Yazılım endüstrisinde bütçenin büyük bir bölümü proje uygulaması için kullanılmaktadır. Bu nedenle, her yazılım şirketi işgücünü etkin bir şekilde yönetmek zorundadır. Yazılım eforunu doğru şekilde tahmin etmek işgücü yönetimi için temel zorunluluktur.

Araştırmacılar yazılım efor tahmininin öneminin farkına 1960'larda varmışlar ve şimdiye kadar aralarında öğrenme tabanlı olanların da bulunduğu birçok model önermişlerdir. Şirketler genellikle az sayıda tamamlanmış projeye ve bundan dolayı yeni projelerin eforunu tahmin etmek için kısıtlı miktarda efor bilgisine sahiptirler. Az miktarda bilgi kullanarak doğru tahminler yapmak zordur. Problem ve tahmin metodları karmaşıklaştıkça küçük veri kümeleriyle efor fonksiyonunu öğrenmek zorlaşır. Bu nedenle, efor tahmini için kullanılan tahmin edicinin performansını arttırmak önemlidir. Birçok araştırmacı güvenilir bir algoritma olarak sinir ağlarını yazılım efor tahmini araştırmalarında tek bir eleman olarak kullanmışlardır. Bu araştırmada, algoritmanın tahmin performansını geliştimeye odaklandık ve bu nedenle tek bir sinir ağı yerine sinir ağı topluluğu kullandık. Bunun yanında çağrışımlı belleği sinir ağı topluluğu ile bir araya getirerek son modeli oluşturduk. Ayrıca, özelliklerin bir kısmını seçmenin efor tahmin performansına etkisini analiz ettik. Bu amaçla, önemli bilginin büyük kısmını taşıyan özellikler bulunur. Ondan sonra ise önerilen modelde efor tahmini yapmak için sadece bu özellikler kullanılır.

Önerilen model doğru tahminler sağlamaktadır. Bu nedenle yazılım firmaları bu modeli yazılım efor tahmininleri yapmak ve işgücünü etkili bir biçimde yönetmek için kullanabilir. Öte yandan, deneylerimizin sonuçları daha az özellik kullanmanın tahmin performansını arttırabileceğini gösterdi.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

ρ	Vigilance
ACAP	Analyst Capabilities
AEXP	Applications Experience
ART	Adaptive Resonance Theory
CMM	Capability Maturity Model
COCOMO	Constructive Cost Model
COTS	Commercial-Off-The-Shelf
CPLX	Product Complexity
DATA	Database Size
DOCU	Documentation Match to Life-Cycle Needs
DSI	Delivered Source Instructions
EM	Effort Multiplier
ENN	Ensemble of Neural Networks
ENNA	Ensemble of Neural Networks with Associative Memory
FCIL	Facilities
FLEX	Development Flexibility
FPA	Function Points Analysis
GA	Genetic Algorithm
GUI	Graphical User Interface
LEXP	Programming Language Experience
LTEX	Language and Tool Experience
MBI	Manpower Buildup Index
MdMRE	Median Magnitude of Relative Error
MLP	Multilayer Perceptron
MMRE	Mean Magnitude of Relative Error
MODP	Modern Programming Practices
MRE	Magnitude of Relative Error
NN	Neural Network

OP	Object Point
PCAP	Programmer Capabilities
PCON	Personnel Continuity
PDIF	Platform Difficulty
PERS	Personnel Capability
PEXP	Platform Experience
PF	Productivity Factor
PM	Person Months
PMAT	Process Maturity
PREC	Precedentedness
PRED	Prediction at a Certain Level
PREX	Personnel Experience
PROD	Productivity Rate
PVOL	Platform Volatility
RCPX	Product Reliability and Complexity
RELY	Required Software Reliability
RESL	Architecture / Risk Resolution
RUSE	Required Reusability
SCED	Required Development Schedule
SDR	Softlab Data Repository
SF	Scale Factor
SITE	Multisite Development
SLIM	Software Life Cycle Model
SLOC	Source Lines of Code
STOR	Main Storage Constraint
TEAM	Team Cohesion
TIME	Execution Time Constraint
TOOL	Use of Software Tools
TURN	Computer Turnaround Time
VEXP	Virtual Machine Experience
VIRT	Virtual Machine Volatility

1. INTRODUCTION

In the last two decades, software development industry has been growing exponentially in terms of revenues [1]. However, ubiquitous and around–the–clock software development made the industry highly competitive forcing companies to work under tight margins. Therefore, software development companies needed to find creative ways to keep their costs under control and make more profit. The major asset (cost) for a software company is its workforce. In a given project, if the estimated effort is much more than the actual effort, then resources are wasted. On the other hand, if the estimated effort is much less than the actual effort, then neither the deadline for the project is met nor the product achieves the desired quality level. Therefore, accurate estimation of effort is essential for a software development company.

Companies usually have small number of completed projects and consequently limited amount of effort data for estimating the effort of new projects [2]. Data scarcity introduces a negative impact on the accuracy of estimations as the estimation methods become more complex.

The interest on software effort estimation began in 1960's [3] and so far several models have been proposed [4, 5]. Most of these models are parametric such as Software Life Cycle Model (SLIM), Function Points Analysis (FPA), ESTIMACS, Constructive Cost Model (COCOMO) and COCOMO II [6, 7, 8, 9, 10, 11, 12, 13, 14, 15]. The main point of concern about parametric models is that they can only be calibrated manually in order to preserve their estimation accuracy [16]. In 1990's machine learning approaches appeared as an alternative to parametric approaches for software effort estimation. Neural networks [17, 18, 19, 20, 21, 22, 23], regression trees [18, 19, 24, 25], case-based reasoning [24, 25, 26, 27, 28, 29] and genetic algorithms [30, 31, 32] are commonly used approaches in this field.

The aim of this research is to improve the prediction performance of an estimator. Firstly, we propose and evaluate a novel effort estimation model. Secondly, we select

essential features in the dataset and estimate the effort by using the subset of essential features. In the proposed model, ensemble of neural networks is used rather than using a single neural network. Moreover, the associative memory is combined with the ensemble to form the final model. We have inspired by the neurophysiologic fact [33, 34] that brain performs a specific calculation by combining the results of different neural networks and remembering the past experience about similar tasks. The contribution of this research is to make accurate effort estimations with limited amount of effort data.

1.1. Motivation

The motivation of this research is proposing a model that provides accurate software development effort estimation. The effect of feature subset selection on the estimation performance is another point of focus.

Effort estimation data is essential for evaluating the performance of the model. In this research, COCOMO, COCOMO II and Function Points based data from public datasets are used.

1.2. Outline

In the next chapter, the history of researches in software effort estimation domain is represented and the underlying theories of models which are related to this research are explained. In the third chapter, we discuss the research questions. Fourth chapter focuses on the proposed model. The key ideas and algorithms to realize those ideas are discussed in detail. Fifth chapter contains information about evaluation criteria, datasets, experiment details and evaluation results of our experiments based on the proposed model. The last chapter discusses conclusion and future work.

2. BACKGROUND

Budgeting is the primary use of effort estimation [4]. Software companies are able to plan short and long term budgets by analyzing estimated efforts. In addition to budgeting, effort estimation guides the company management for making tradeoff and risk analysis [4]. In other words, the cost drivers of software development can be analyzed and some of them can be considered as more important than the remaining cost drivers. As an example, management decides to hire experienced programmers instead of purchasing a new software tool. Project planning and control is another use of effort estimation [4]. Cost and schedule breakdowns can be detected in terms of component, project stage or activity.

Since 1960's researchers have proposed several effort estimation techniques which can be grouped in six major categories as model-based, expertise-based, learning-oriented, dynamics-based, regression-based and composite [4].

Model-based techniques use mathematical equations which are based on historical data and theory [4]. Expertise-based techniques capture the knowledge and experience of practitioners in a domain [4]. Learning-oriented techniques focus on automated models that learn from previous experience [4]. Dynamics-based techniques acknowledge that effort factors change during software development [4]. Regression-based techniques refer to statistical approaches of regression and used in conjunction with model-based techniques [4]. Lastly, composite techniques combine two or more techniques to provide an estimation function [4].

In this research, we propose a novel learning-oriented technique. We have conducted experiments on various datasets to validate our proposed model. These datasets were collected for model-based techniques. The following section provides information about model-based and learning-oriented techniques.

2.1. Model-based Techniques

Several researchers proposed proprietary models for effort estimation [4]. These models define effort as a function of variables such as lines of code, function points, personnel capabilities and process maturity. Some of the most popular models are Software Life Cycle Model (SLIM) [35], Function Points Analysis (FPA) [36], ESTIMACS [37], Constructive Cost Model (COCOMO) [38, 39] and COCOMO II [40, 41].

Larry Putnam proposed Software Life Cycle Model (SLIM) in the late 1970's [35]. This model is based on Rayleigh distribution of project personnel level versus time [35]. The shape of the Rayleigh curve is modified by the user through two key parameters. These parameters are Manpower Buildup Index (MBI) and Productivity Factor (PF) and they are determined either by analyzing previously completed projects or answering a set of questions [35].

The Function Points Analysis (FPA) method was developed by Allan Albrecht at IBM and first published in 1979 [36]. Counting function points consists of two steps. These are counting the user functions and adjusting complexity by using 14 processing complexity characteristics. Once the number of function points is computed, it is used to compare the current project with past projects to estimate effort [36]. This method appeared as an alternative to estimating source lines of code (SLOC) providing some advantages. Firstly, function points can be estimated in the early phase of the development. Secondly, function points can be estimated by non-technical personnel. Lastly, effects of language differences are eliminated [36].

ESTIMACS is a business-oriented method, which was developed by Howard Rubin [37]. ESTIMACS uses business terms in effort estimation and does tradeoff and sensitivity analysis as early as possible. This model is based on six estimation dimensions and their relationships. These dimensions are effort hours, staff size and deployment, cost, hardware resource requirements, risk and portfolio impact [37].

Constructive Cost Model (COCOMO) predicts the effort of a project based on its size and a number of cost drivers [38, 39]. Later, it was tuned to meet the requirements of

modern software lifecycles and COCOMO II was proposed [40, 41]. So far it has been a widely used parametric model in the industry [42]. Therefore, detailed information about these models will be given in the following sections.

In parametric models, the effort is defined as a function of variables. This fact introduces certain advantages. Firstly, the underlying theories of parametric models are clear. Secondly, the formulas can be easily refined and customized. Lastly, input data can be easily modified [43]. In addition to these advantages, parametric models introduce some disadvantages. Firstly, some experience and factors can not be quantified. Secondly, some of these models are developed within companies for internal use and reflect specific companies' performance characteristics. Such models may not provide accurate estimations outside the company. Lastly, parametric models can be calibrated only manually [43].

2.1.1. COCOMO

In 1981, Barry Boehm introduced Constructive Cost Model (COCOMO) in his classic book Software Engineering Economics [38]. This model is based on the analysis of 63 software development projects ranging in size from 2,000 to over 1,000,000 lines of code in several programming languages.

The difficulty of the project and the familiarity are essential factors of the effort. Therefore, COCOMO operates on three different modes, which are organic, semi-detached and embedded [39]. These modes correspond to the difficulty level and familiarity of the project.

Organic mode is used for simple projects. These projects do not have rigid constraints upon development. In addition to that, previously a number of similar projects have been implemented. Therefore, the software development team members are familiar with the characteristics of the project [39].

Semi-detached mode is used for projects where constraints upon development are more rigid than organic mode. However, there still remains some flexibility on the

constraints. Additionally, there are only a few similar past project. These projects are bigger in size and complexity than projects for which organic mode is used [39].

Embedded mode is used for projects with tight hardware, software and operational constraints. Moreover, these projects have unique characteristics. Therefore, such projects do no provide familiarity [39].

COCOMO gives the effort result in units of person-month (PM), which is the number of months needed for one person to develop the project. One month corresponds to 152 working hours [39].

COCOMO is a hierarchical model and it consists of three increasingly detailed levels, which are basic, intermediate and detailed. Basic COCOMO defines software effort as a function of size [39]. Size corresponds to program instructions created by project personnel that are delivered as part of the final product. Delivered Source Instructions (DSI) is the term used for size. Basic COCOMO equation is given in Equation 2.1.

$$PM_{no\min al} = a \times (KDSI)^b \tag{2.1}$$

where *PM* is person-months, *a* and *b* are constant values that are dependent on the operation mode and *KDSI* is thousands of delivered source instructions. Table 2.1 shows basic COCOMO equations with the corresponding constant values replaced.

Table 2.1. Basic COCOMO effort estimation

Mode	**Effort Estimation**
Organic	$PM_{nominal} = 3.2\ (KDSI)^{1.05}$
Semi-detached	$PM_{nominal} = 3.0\ (KDSI)^{1.12}$
Embedded	$PM_{nominal} = 2.0\ (KDSI)^{1.20}$

Basic COCOMO defines effort as a function of operation mode and project size. However, these factors are not the only contributing elements of software effort.

Intermediate COCOMO provides additional factors which are grouped in four categories as product attributes, computer attributes, personnel attributes and project attributes [39].

Product attributes refer to the constraints and requirements of the project. These are required software reliability (RELY), database size (DATA) and product complexity (CPLX).

Computer attributes refer to the limitations placed upon development effort by the platform on which the project will run. Platform corresponds to the hardware and the operating system. These limitations are execution time constraint (TIME), main storage constraint (STOR), virtual machine volatility (VIRT) and computer turnaround time (TURN).

Personnel attributes refer to the skill level of the development team. Skill stands for professionalism, programming ability, experience with the development environment and familiarity with the project's domain. Personnel attributes are listed as analyst capabilities (ACAP), applications experience (AEXP), programmer capabilities (PCAP), virtual machine experience (VEXP) and programming language experience (LEXP).

Project attributes are related with constraints and conditions under which project is developed. Modern programming practices (MODP), use of software tools (TOOL) and required development schedule (SCED) are the project attributes.

Each of these 15 cost drivers is rated on a six-point scale as very low, low, nominal, high, very high and extra high. Each cost driver has corresponding real numbers for the defined range of ratings. The corresponding real numbers for each of these 15 cost drivers are multiplicated together and incorporated to the effort estimation formula of Basic COCOMO to form Intermediate COCOMO equation, which is given in Equation 2.2.

$$PM_{final} = PM_{nominal} \times \prod_{i=1}^{15} EM_i \tag{2.2}$$

where *PM* is person-months and EM_i is the effort multiplier associated with the i^{th} cost driver.

It is obvious in Equation 2.2 that a rating less than one decreases the effort whereas a rating more than one increases it. Moreover, the corresponding real number for all nominal rated attributes is one, which implies that it has no effect on the effort. The cost driver attributes are given in Table 2.2. As the rating of product and computer attributes increase, effort increases. On the other hand, the increase in the ratings of personnel and project attributes decreases the effort.

Table 2.2. COCOMO cost drivers

Cost Drivers		Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes							
Required software reliability	RELY	0.75	0.88	1.00	1.15	1.40	
Database size	DATA		0.94	1.00	1.08	1.16	
Product complexity	CPLX	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes							
Execution time constraint	TIME			1.00	1.11	1.30	1.66
Main storage constraint	STOR			1.00	1.06	1.21	1.56
Virtual machine volatility	VIRT		0.87	1.00	1.15	1.30	
Computer turnaround time	TURN		0.87	1.00	1.07	1.15	
Personnel Attributes							
Analyst capabilities	ACAP	1.46	1.19	1.00	0.86	0.71	
Applications experience	AEXP	1.29	1.13	1.00	0.91	0.82	
Programmer capabilities	PCAP	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	VEXP	1.21	1.10	1.00	0.90		
Programming language experience	LEXP	1.14	1.07	1.00	0.95		
Project Attributes							
Modern programming practices	MODP	1.24	1.10	1.00	0.91	0.82	
Use of software tools	TOOL	1.24	1.10	1.00	0.91	0.83	
Required development schedule	SCED	1.23	1.08	1.00	1.04	1.10	

Intermediate COCOMO treats the project as a whole. In other words, same cost drivers are evaluated and applied for the whole project. On the other hand, Detailed COCOMO divides the project into four phases, which are Product Design, Detailed Design, Coding/Unit Test and Integration/Test [39]. Cost drivers are evaluated and applied

separately for each of these phases. The only difference between intermediate and detailed COCOMO is the fact of treating the project as a whole or dividing into parts.

In this research, the affect of project phases on software development effort is out of the consideration. Therefore, intermediate COCOMO based datasets are used.

2.1.2. COCOMO II

In 1990's a new generation of software processes and products appeared and changed the way of software development. Although some of the existing software effort estimation models address some aspects of these new issues, they were not totally compatible. For example, COCOMO encountered problems in estimating the effort for object-oriented software, software created by using spiral and evolutionary development models and commercial-off-the-shelf (COTS) applications [40]. Because of inaccurate effort estimation concerns, Boehm tuned COCOMO to meet the requirements of the modern software lifecycles and proposed COCOMO II. The first implementation of COCOMO II was released in 1997 and the model was documented in his book Software Cost Estimation with COCOMO II [41].

COCOMO II provides three major process models which are mixed with respect to process strategy. These models are Application Composition, Early Design and Post-Architecture models [40].

Application Composition model focuses on prototyping efforts, which generally appear in the earliest phases or spiral cycles of software development. The applications addressed by this model are composed from interoperable components. Some examples of component-based applications are graphical user interface (GUI) builders, database or object managers and domain-specific components such as financial, medical and industrial process control packages [40].

Application Composition model uses object-point estimation, which is well suited to the practices in developing component-based applications. Firstly, the number of objects such as screens and reports are estimated and each object is classified as simple, medium

or difficult. Secondly, each object instance is weighted according to type and complexity and these weights are summed to form object-point count. Then, percentage of reuse is estimated and subtracted from object-point count as in Equation 2.3 to get final object-point count [40].

$$OP_{final} = \frac{OP \times (100\text{-}RP)}{100} \tag{2.3}$$

where OP_{final} is final object-point count, OP is existing object-point count and RP is reusability percentage. Having calculated the final object-point count, productivity rate is determined according to the developer capability. Productivity rate is the object-points count per person-month. Finally, effort is estimated as in Equation 2.4 [40].

$$PM = \frac{OP_{final}}{PROD} \tag{2.4}$$

where PM is person-months, OP_{final} is the final object-point count and $PROD$ is productivity rate.

Early Design model involves exploration of software and system architectures and incremental development strategies. This model is used in the early phases and spiral cycles of software development where very little is known about the size of the product, target platform features, characteristic of project personnel and detailed process specifications [40].

Size estimation in Early Design model starts with function points because of the fact that they are based on the information available in the early stages of the project. Thereafter, function points count is converted to equivalent thousands source lines of code (KSLOC) which is used as the size measure [40].

Early Design model cost drivers are formed by combining the Post-architecture model cost drivers. These cost drivers are personnel capability (PERS), product reliability and complexity (RCPX), required reuse (RUSE), platform difficulty (PDIF), personnel experience (PREX), facilities (FCIL) and schedule (SCED). Each of these seven cost

drivers are rated on a seven-point scale as extra low, very low, low, nominal, high, very high and extra high. Each cost driver has corresponding real numbers for the defined range of ratings. Early Design model effort equation is given in Equation 2.5 [40].

$$PM_{final} = PM_{nominal} \times \prod_{i=1}^{7} EM_i \tag{2.5}$$

where PM is person-months and EM_i is the effort multiplier associated with the i^{th} cost driver.

Post-architecture model focuses on the actual development and maintenance of a software product. In Post-architecture model, either function points count is converted to equivalent thousands source lines of code (KSLOC) or source lines of code count is directly used as the size measure. Therefore, some components can be sized using function points whereas the remaining components can be sized in source lines of code [40].

There are 17 cost drivers in Post-architecture model, which are grouped into four categories as product factors, platform factors, personnel factors and project factors [40].

Product factors are related to the constraints and requirements of the project. Required software reliability (RELY), database size (DATA) and product complexity (CPLX) remain the same as in COCOMO. Compared with the original COCOMO, there are two new cost drivers. These are required reusability (RUSE) and documentation match to life-cycle needs (DOCU) [39, 40].

Platform factors refer to constraints about the hardware and infrastructure software. Execution time constraint (TIME) and main storage constraint (STOR) remain the same as in COCOMO. Platform volatility (PVOL) substituted virtual machine volatility (VIRT) and computer turnaround time (TURN) of original COCOMO [39, 40].

Personnel factors are about the characteristics of the project team. Analyst capability (ACAP), programmer capability (PCAP) and applications experience (AEXP) are the same as in COCOMO. Platform experience (PEXP) substituted virtual machine experience

(VEXP) and language and tool experience (LTEX) substituted language experience (LEXP) of COCOMO. Moreover, personnel continuity (PCON) joined personnel factors of COCOMO II [39, 40].

Project factors focuses on constraints and conditions under which project is developed. Use of software tools (TOOL) and required development schedule (SCED) are the same as in COCOMO. In COCOMO II, multisite development (SITE) substituted modern programming practices (MODP) of original COCOMO [39, 40].

Each of these 17 cost drivers is rated on a six-point scale as very low, low, nominal, high, very high and extra high. Each cost driver has corresponding real numbers for the defined range of ratings as seen in Table 2.3 [40].

Table 2.3. COCOMO II cost drivers

Cost Drivers		Very Low	Low	Nominal	High	Very High	Extra High
Product Factors							
Required software reliability	RELY	0.75	0.88	1.00	1.15	1.39	
Database size	DATA	0.93	1.00	1.09	1.19		
Product complexity	CPLX	0.75	0.88	1.00	1.15	1.30	1.66
Required reusability	RUSE	0.91	1.00	1.14	1.29	1.49	
Documentation match to life-cycle needs	DOCU	0.89	0.95	1.00	1.06	1.13	
Platform Factors							
Execution time constraint	TIME	1.00	1.11	1.31	1.67		
Main storage constraint	STOR	1.00	1.06	1.21	1.57		
Platform volatility	PVOL	0.87	1.00	1.15	1.30		
Personnel Factors							
Analyst capability	ACAP	1.50	1.22	1.00	0.83	0.67	
Programmer capability	PCAP	1.37	1.16	1.00	0.87	0.74	
Applications experience	AEXP	1.22	1.10	1.00	0.89	0.81	
Platform experience	PEXP	1.25	1.12	1.00	0.88	0.81	
Language and tool experience	LTEX	1.22	1.10	1.00	0.91	0.84	
Personnel continuity	PCON	1.24	1.10	1.00	0.92	0.84	
Project Factors							
Use of software tools	TOOL	1.24	1.12	1.00	0.86	0.72	
Required development schedule	SCED	1.29	1.10	1.00	1.00	1.00	
Multisite development	SITE	1.25	1.10	1.00	0.92	0.84	0.78

In Post-architecture model, five factors determine the project's scaling exponent. These factors replace the modes in original COCOMO, which are organic, semi-detached and embedded. These scale drivers are precedentedness (PREC), development flexibility (FLEX), architecture/risk resolution (RESL), team cohesion (TEAM) and process maturity (PMAT). Precedentedness (PREC) includes factors such as organizational understanding of the product objectives and experience of working with related software systems. Architecture/risk resolution (RESL) focuses on facts like risk management plan, budget, software architects and available tool support. Team cohesion (TEAM) consists of facts such as stakeholder objectives and experience of stakeholders operating as a team. Each of these for scale drivers are rated ranging from very low to extra high like cost drivers. Process maturity (PMAT) is the Capability Maturity Model (CMM) level of the company, which ranges from one to five. Each scale driver has corresponding real numbers for the defined range of ratings as seen in Table 2.4 [40].

Table 2.4. COCOMO II scale drivers

Scale Drivers		Very Low	Low	Nominal	High	Very High	Extra High
Precedentedness	PREC	4.05	3.24	2.43	1.62	0.81	0.00
Development Flexibility	FLEX	6.07	4.86	3.64	2.43	1.21	0.00
Architecture / Risk Resolution	RESL	4.22	3.38	2.53	1.69	0.84	0.00
Team Cohesion	TEAM	4.94	3.95	2.97	1.98	0.99	0.00
Process Maturity	PMAT	4.54	3.64	2.73	1.82	0.91	0.00

Effort multipliers for 17 cost drivers and scale factors for five scale drivers are incorporated into the Basic COCOMO effort equation to form COCOMO Post-Architecture model effort equation is given in Equation 2.6 [40].

$$PM_{final} = a \times (KDSI)^b \times \prod_{i=1}^{17} EM_i$$
$$a = 2.5 \tag{2.6}$$
$$b = 1.01 + 0.01 \times \sum_{j=1}^{5} SF_j$$

where PM is person-months, EM_i is the effort multiplier associated with the i^{th} cost driver and SF_j is the scale factor associated with the j^{th} scale driver.

2.2. Learning-oriented Techniques

Learning-oriented techniques appeared as an alternative to parametric approaches in 1990's [42]. These techniques focus on learning from past experience to provide a solution to future problems. So far, researchers have used case-based reasoning, regression trees and artificial neural networks and genetic programming for software effort estimation [17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32].

Case-based reasoning is defined as the process of solving new problems based on the solutions of similar past problems [44]. In the context of software effort estimation, case-based reasoning is the process of finding the effort of current project by using the actual efforts of similar past projects. Estimation by analogy is a form of case-based reasoning. The steps of this method are identification of the project as a new case, retrieval of similar projects from the repository and effort estimation by using knowledge of similar projects. Similarity of projects can be assessed via various approaches including nearest neighbor algorithms, manual human expert guidance and goal directed preference [24, 25, 26, 27, 28, 29].

Regression tree is a hierarchical model where a local region is identified in a sequence of recursive splits in a smaller number of steps [45, 46]. Software project attributes are used to split projects into smaller groups and this process is recursively repeated to form a regression tree [18, 19, 24, 25].

Artificial neural networks are proven to be capable of approximating non-linear functions [47]. Since software effort can be defined as a function of several variables, artificial neural networks have the ability of learning this function and estimating effort of new software projects [17, 18, 19, 20, 21, 22, 23].

Computer scientists have been inspired by evolution theory which states that genetic operations between chromosomes lead to fitter individuals who are more likely to survive

[48]. Genetic algorithms (GA), therefore, have been used in computer science. In the context of software effort estimation, these algorithms behave effort equation as a chromosome. Instantly, a number of random effort equations are generated. Thereafter, a new population of effort equations is created from previous effort equations by applying genetic operations to the most accurate effort equations. The main genetic operations are reproduction and crossover. Reproduction is the copying of an equation from one generation to the next generation unchanged. In crossover, the parts of two equations are exchanged to provide two new effort equations. Genetic operations continue from one generation to the next until a good enough effort equation is derived [30, 31, 32].

3. PROBLEM STATEMENT

Effort estimation in software project management has become an important task for effective resource management. If the estimated effort is much less than the actual effort, the estimated schedule and budget will not be enough to complete the project within the quality goal. If the related project deadline is specified with a contract, the company will have to compensate for missing the deadline. Additionally, company reputation will be damaged. On the other hand, if the estimated effort is much more than the actual effort, the company will allocate more resources than actually needed. Wasting resources, therefore, results in higher costs [4].

Amount of effort data is usually limited because most of the software companies have a limited number of completed projects [2]. However, it is hard to make accurate estimations with scarce data because as the complexity of the problem and the estimation method increase, it becomes harder to make inference about the effort function. Facing such an important drawback, researchers focused on finding an answer to the following question: “How can we improve software effort estimation accuracy?” There are two main paths which have been followed to provide an answer. First of them is applying new algorithms which may outperform the existing ones. The second path is making decisions on project effort data to improve accuracy of the existing algorithms. Dimensionality of the effort data can be reduced by extracting new features which are a combination of the existing features [49, 50] or selecting the most important features and ignoring the rest [51, 52]. Effort data collected in other companies can also be used to eliminate data scarcity and consequently provide accuracy improvement in software effort estimation [53, 54, 55].

In this research, we followed the path of searching for a new algorithm to outperform the existing ones. In addition to that, we analyzed the effect of selecting the most important features with a view to improve the accuracy of the proposed model.

4. PROPOSED MODEL

Artificial neural networks have been used by many researchers for software effort estimation [17, 18, 19, 20, 21, 22, 23]. The common point for them is that they have used single artificial neural network and this fact introduces several problems. First of all, if the effort function is complex, it may not be learned perfectly for all the regions of the input space. In other words, the model may have high bias on particular regions of the input space. Secondly, variance also contributes to the poor effort estimation performance [56]. Thirdly, some types of neural networks such as multilayer perceptrons (MLP) may be overtrained unless a validation set is used. However, project effort data sets are very small and choosing some instances for validation makes it even smaller, which affects learning performance negatively.

In this research, our aim is to provide solutions to problems stated above. Ensemble of neural networks is used rather than a single neural network. Therefore, neural network variance decreases. Moreover, the results of the overtrained neural networks in the ensemble are eliminated such that they do not affect the accuracy of the result. Although using an ensemble and ignoring the results of overtrained neural networks provide accuracy, there may still be considerable bias of this model. For decreasing bias, associative memory is used. An associative memory is a system which stores mappings of specific input representations to specific output representations. The word “associative” comes from the fact that this system “associates” two patterns such that when one is encountered subsequently, the other can be reliably recalled [57]. In our case, the effort database of past projects corresponds to the associative memory. The projects that are similar to the current project are retrieved from the project database. These projects are given to the ensemble and estimated efforts are compared with the actual efforts. The difference between the estimated and actual values corresponds to the biases of the model for similar projects. The average of biases is the potential bias for the current project. This value is added to the result of the ensemble to compensate for potential bias and to provide a more accurate result.

Previously, the idea of ensemble of neural networks has been used in several fields such as biochemistry [58], fault diagnosis [59] and genetics [60]. Moreover, associative memory is used in the field of biochemistry [58]. In this research, a model is provided for software effort estimation domain. The main question that this research answers is: “Does ensemble of neural networks with associative memory improve software effort estimation accuracy?”

Our model will be explained in a step by step manner to enhance understandability. Firstly, theory of neural networks, which are the basic building blocks of the proposed model, will be discussed. Then, a number of neural networks will come together to form ensemble of neural networks (ENN). Thirdly, associative memory will be combined with ENN to form the final model, namely ensemble of neural networks with associative memory (ENNA).

4.1. Neural Network (NN)

Neurons are physically excitable cells in the nervous system which process and transmit information [61]. A neuron has a number of dendrites, which are the input connections, and one axon, which is the output connection. A neuron interacts with the surrounding neurons by using dendrites and the axon [61]. A neural network describes a population of physically interconnected neurons whose inputs and outputs define a recognizable circuit [61].

Artificial neural network (ANN) is a computational model which is inspired by biological neural networks. It is a population of interconnected artificial neurons and performs a function [62]. An ANN consists of one input layer, one output layer and a number of hidden layers between them. For each input there is a corresponding unit in the input layer. Likewise, number of units in the output layer is equal to the number of outputs. Hidden layers do not have connection with the environment as the word “hidden” implies. An ANN, which has at least one hidden layer, has the capability of approximating a nonlinear function [46].

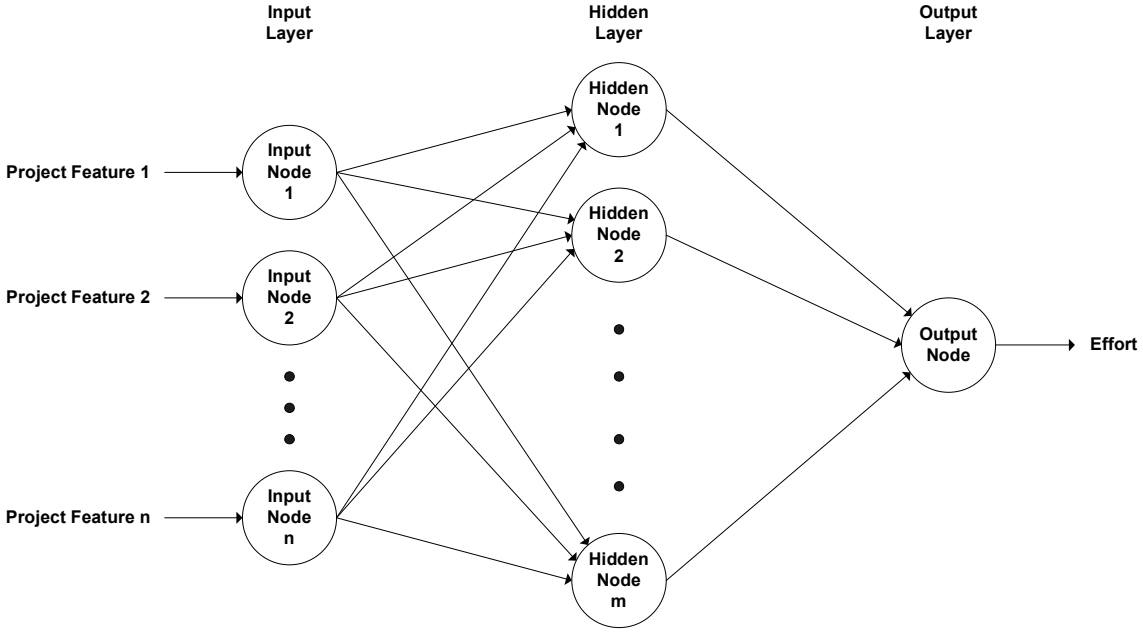

Figure 4.1. Artificial neural network for software effort estimation (adopted from [46])

In software effort estimation domain, inputs are features of the given project and the output is the estimated effort for the given project. ANN's with at least one hidden layer must be used because software effort function is nonlinear [38, 39]. A sample ANN for software effort estimation is shown in Figure 4.1. Project features are fed into the input layer. Input nodes make no computation and directly pass the values to the hidden nodes. Hidden nodes make computations and feed the results to the output node. The result of the computation in the output node gives the result, which is the estimated effort. Since there is no computation in the input layer, it is not counted and the ANN in Figure 4.1 is called a two-layered network [46].

Multilayer Perceptron (MLP) is an ANN structure which is a population of interconnected neuron-like processing elements called perceptrons [46]. A perceptron has inputs that may come from the environment or may be the outputs of other perceptrons. In addition to these inputs, there is a bias unit input that always has the value one [46]. Each input has a corresponding connection weight and the output is a function of inputs and weights. A perceptron is shown in Figure 4.2 [46]. In this figure x_j, j = 1,…,d are the input units. x_0 is the bias unit that always has the value one. y is the output unit. w_j is the weight for input x_j and w_0 is the weight for the bias unit x_0.

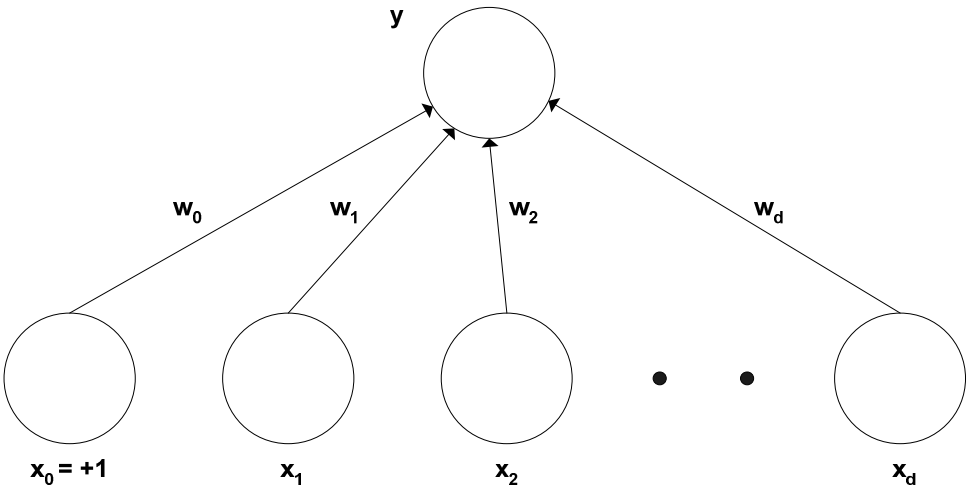

Figure 4.2. Perceptron [46]

In this research, hyperbolic tangent sigmoid function is applied to the weighted sum of the inputs as shown in Equation 4.1. The reason for choosing hyperbolic tangent transfer function is that it is faster than alternative transfer functions whilst it provides the same performance [63].

$$y = \tan sig\left(\sum_{j=1}^{d} w_j x_j + w_0\right) \tag{4.1}$$

where x_j is the input unit, w_j is the corresponding connection weight of x_j and w_0 is the corresponding weight of the bias unit.

A perceptron has a single layer of weights and can approximate only linear functions [46]. However, software effort estimation requires nonlinear regression. Therefore, an artificial neural network with at least one hidden layer between the input layer and the output layer must be used [46]. Such an artificial neural network (ANN) is called a multilayer perceptron (MLP). The structure of a two-layered multilayer perceptron (MLP) with one output is shown in Figure 4.3. In this figure, x_j, j = 1,…,d are the input units, z_h, h = 1,…,H are the hidden units. x_0 and z_0 are the biases of the input layer and the hidden layer respectively. y is the output unit. w_{hj} and v_h are the weights of the input layer and the hidden layer respectively and w_{h0} and v_0 are the weights for the bias units x_0 and z_0.

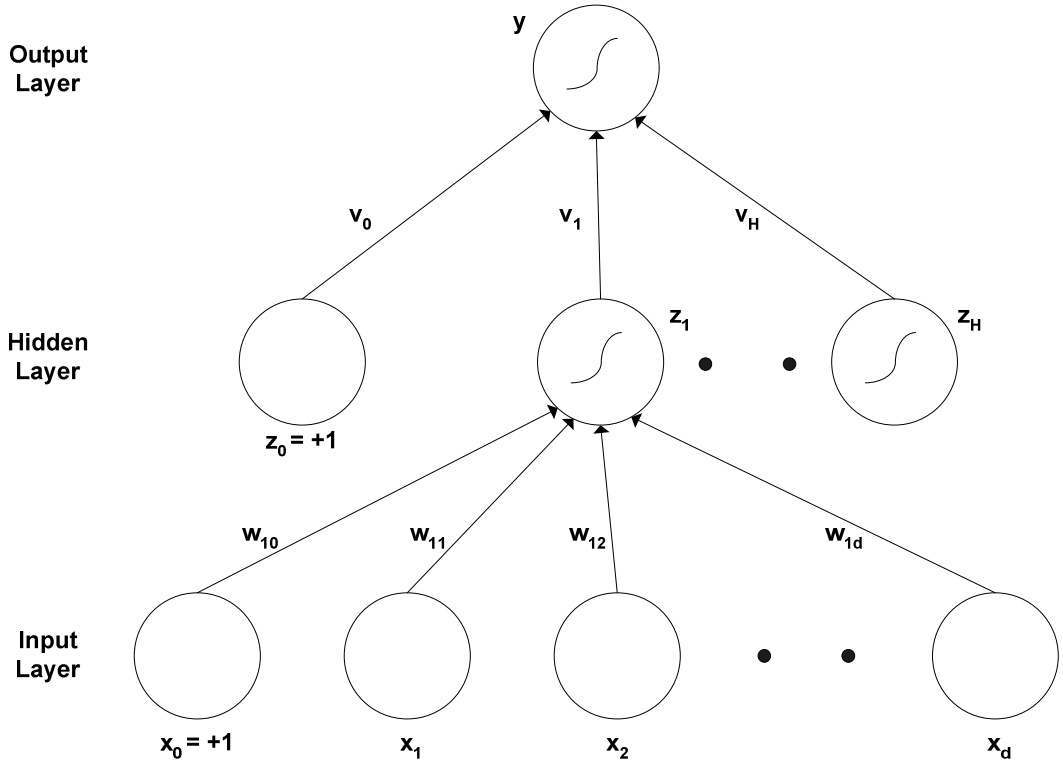

Figure 4.3. Multilayer perceptron [46]

Backpropagation is a supervised learning method which is used for training artificial neural networks (ANN) [46]. This method comprises a forward pass and a backward pass through the network. In forward pass, the training instance inputs are fed to the network to get the network's response. The actual output and the evaluated output are used to calculate the error. In backward pass, the error is propagated back through the network and weights are updated. The whole process of forward and backward passes continues until the error is minimized. The pseudo code of the algorithm is shown in Figure 4.4 [46].

```
initialize weights
until convergence
        for all training samples
                feed training instance inputs to the network
                calculate error
        update weights
```

Figure 4.4. Backpropagation algorithm [46]

In the backpropagation algorithm shown in Figure 4.4, changes to the weights for all training instances are accumulated and made once after a complete pass over the whole training set. This approach is called batch learning [46].

There are several backpropagation training algorithms. In this research, MLP is trained with Levenberg-Marquardt algorithm [64, 65], which is very efficient for training moderate sized neural networks, which have up to several hundred weights like the neural network for software effort estimation as shown in Figure 4.1 [66].

4.2. Ensemble of Neural Networks (ENN)

Neuroscientists have discovered that individual neural networks are very noisy and are unable to perform a certain task alone [33, 34]. The neural networks in the visual cortex, which provide the ability of sight, can be given as an example. Examining the activity of a single neural network is not enough to reconstruct the visual scene that the owner of the brain is looking at. Therefore, a population of visual neural networks provides the visual scene [67].

There is a similar issue for MLP's. If the function is complex, an MLP may not learn the function perfectly in all regions of the input space [68]. Therefore, significant bias may appear for regions of the input space in which the function is not perfectly learned. In addition to this fact, small changes in the training set of an MLP causes large difference in the trained MLP. In other words, learning algorithm has high variance, which contributes to the learning performance negatively [56]. Yet another problem is overtraining. Initially all weights in MLP are close to zero and consequently have little effect. As the training continues, the most important weights start moving away from zero and become utilized. However, if the training continues further, all weight moves away from zero and are utilized. This fact increases complexity and leads to poor generalization [46]. Using a validation set solves the problem of overtraining. When the error on the validation set starts to increase beyond a certain point, the training is stopped [46]. However, this method is not applicable for software effort estimation domain in which data sets are very small [22, 53]. All instances are essential for training and they should be used for training.

Using an ensemble of MLP's and combining the results of individual MLP's by using a robust method provide a solution to the algorithmic problems which are stated above. The main idea of the ensemble is generating complementary base learners which are similar but slightly different. Slight difference between MLP's in the ensemble is provided by training each MLP with a slightly different training set. The individual training set is generated by choosing training instances randomly from the original training set. After each instance is chosen it is replaced back to the original set. Therefore, an instance may appear more than once in the generated training set. This method is called bootstrapping and it is considered as the best way for domains with very small datasets such as software effort estimation [46].

In bootstrapping, the size of the individual training set is equal to the size of the original training set [46]. If there are N instances in the training set, the probability of drawing an instance is 1/N and the probability of not drawing it is 1-1/N. The probability that an instance is not selected after N draws is given in Equation 4.2 below.

$$\left(1-\frac{1}{N}\right)^{N} \approx e^{-1} = 0.368 \tag{4.2}$$

In Equation 4.2, about 63 per cent of the instances are in the individual training set. However, there are a number of MLP's in the ensemble each of which is trained with about 63 per cent of the instances. This fact implies that the original training set is most probably totally covered.

In this research, the ensemble consists of 20 MLP's. For determining the number of MLP's, several experiments with different numbers such as 15, 20, 30, 40 and 50 are conducted and no significant accuracy difference is observed. Therefore, the number of MLP's is decided to be 20. The training process is shown in Figure 4.5. Individual training sets are generated randomly and each MLP is trained with the corresponding training set.

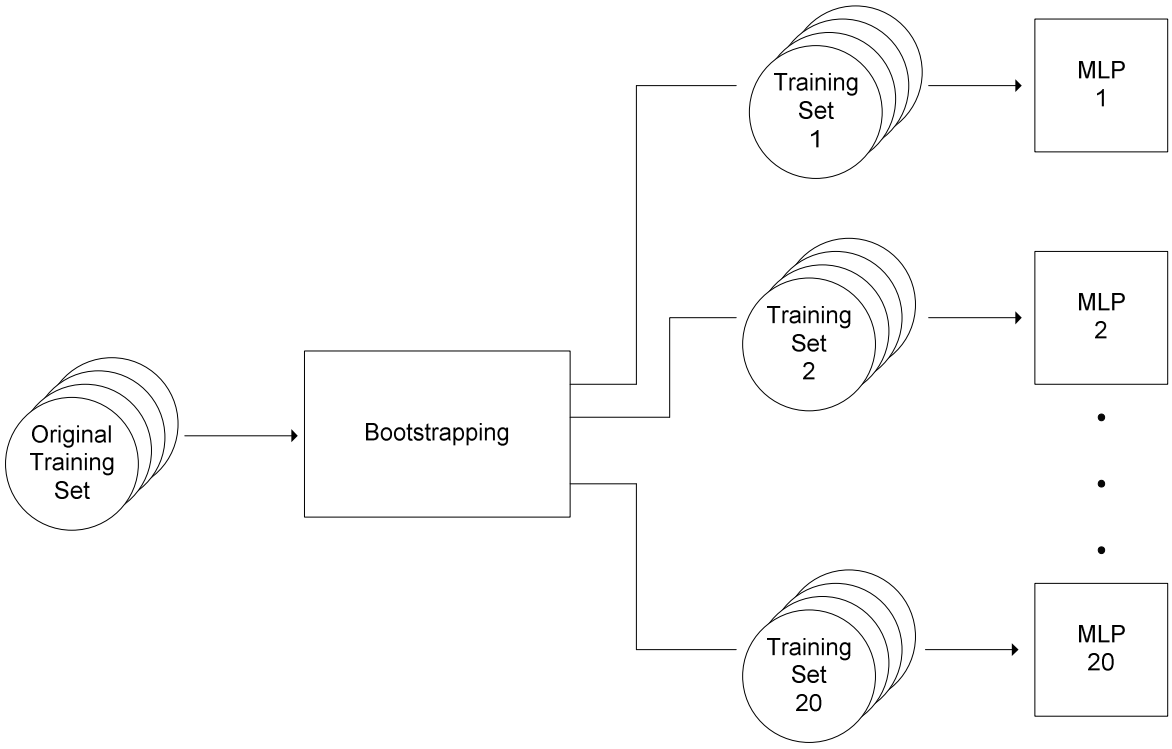

Figure 4.5. ENN training

When a project instance is given to the ensemble for effort estimation, each MLP in the ensemble evaluates the instance to produce an estimated effort value as in Equation 4.3.

$$\mathbf{PM} = \begin{bmatrix} PM_1 \\ PM_2 \\ . \\ . \\ PM_{20} \end{bmatrix} \tag{4.3}$$

where PM_i is the effort value estimated by multilayer perceptron i. Since effort is in person-months, abbreviation PM is used.

The estimations made by some MLP's may deviate much from the largest group of nearby estimations. Because some MLP's would have fallen into local minimum and some others would have been inaccurately trained. Therefore, a simple average of all estimations would not be an accurate result. Instead of this, it would be nice to detect the largest group of nearby results and return the average of this group. In other words, the results which are

not a member of this group should be ignored. We have been inspired by the plurality voting in democracy where the group having the maximum number of votes is the winner [46]. For applying such an idea, clustering is used. At that moment, the number of result groups is not known. Therefore, the clustering algorithm should start with one single cluster, add new clusters as they are needed and delete the empty clusters. Adaptive Resonance Theory (ART) algorithm, which follows such an incremental approach, is used in this research [69].

In software effort estimation, the results are scalar values. Therefore, ART algorithm operates in a one-dimensional space. ART algorithm starts with one center which is equal to the average of all results. For each result, the nearest center in terms of Euclidean distance is calculated. If the Euclidean distance between the result and the nearest center is smaller than the threshold value, nothing is done. Otherwise, a new center is created whose value is equal to the result. Lastly, all centers are updated. The process of calculating the nearest centers, creating new centers and updating center positions continues until the convergence point. The threshold value, namely vigilance, determines the minimum distance for new center creation. The pseudocode of ART algorithm is given in Figure 4.6 [69].

```
m_1 = average ( ∀ PM)          // Start with one center
k = 1                          // The largest center index
until convergence
        for all PM_i in random order
                i = arg min_j ‖PM_i - m_j‖      // Find the nearest center
                if ‖PM_i - m_i‖ > ρ             // If distance > vigilance
                        m_{k+1} = PM_i          // Create new center
        for all m_n
                m_n = average ( ∀ PM ∈ n)       // Update center positions
```

Figure 4.6. ART algorithm (adopted from [69])

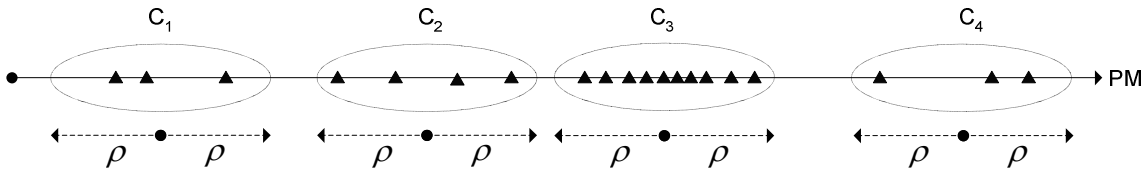

Figure 4.7. Clustering with ART algorithm

To enhance clarity, ART algorithm is simulated in Figure 4.7. Each MLP gives a result which is a scalar effort value in person-months. ART algorithm divides these results into four clusters. The largest cluster is C_3. Therefore, average of the results in C3 is taken to provide the result of the ensemble. ρ is vigilance which indicates the distance between the center and the boundary of the cluster.

By using ART algorithm the largest group of results is detected. The average of the results in the largest group is taken as in Equation 4.4 to provide the effort estimated by the ensemble.

$$PM_{ENN} = \frac{\sum PM_i}{N_C} \tag{4.4}$$

where C is the largest cluster, N_C is the number of results in C and $\forall\ PM_i \in C$.

The overall process of ensemble effort estimation is shown in Figure 4.8. A production instance is fed into each MLP in the ensemble to get estimated effort values from each of them. These values are clustered by using ART algorithm and the average of the largest cluster is calculated to provide the estimated value by the ensemble.

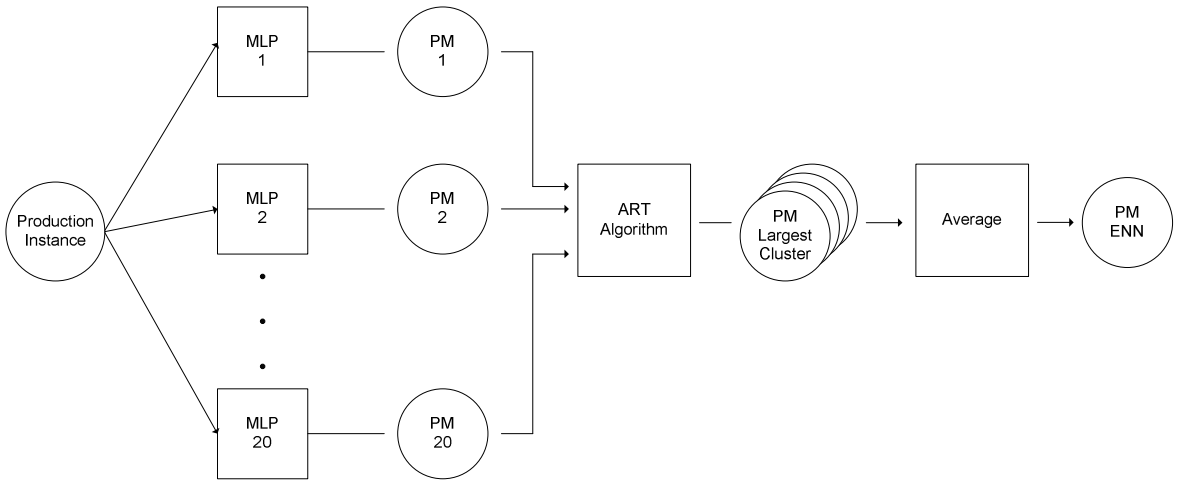

Figure 4.8. ENN simulation

4.3. Ensemble of Neural Networks with Associative Memory (ENNA)

Using an ensemble rather than a single MLP and combining the results of MLP's by using a robust method provide a model for accurate software effort estimation. However, there may still be considerable bias of this model. The concept of associative memory is used for estimating and correcting bias [58].

An associative memory is a system which stores mappings of specific input representations to specific output representations. The word "associative" comes from the fact that this system "associates" two patterns such that when one is encountered subsequently, the other can be reliably recalled [57]. In software effort estimation domain, the effort database of past projects corresponds to the associative memory.

When a new project is given to the ensemble, the estimated effort is returned as in Equation 4.4. In order to correct bias, the bias of the model for the new project must be estimated. The key assumption here is that the model bias is similar for similar projects. Therefore, similar past projects in the effort database are used for estimating bias of the model for the new project. These projects are retrieved from the database using K nearest neighbors algorithm as in Figure 4.9 [46]. Before retrieval process, project effort database and new project are normalized so that the minimum value is -1 and the maximum value is

```
normalize (all Pi in project effort database)   // Normalize project effort database
normalize(P)                                    // Normalize new project
for all Pi in project effort database
        distance(i, 1) = ‖Pi - P‖        // Calculate the distance to the new project P
        distance(i, 2) = i               // Put the index of project to the distance array
sort(distance, 1)        // Sort distances such that the smallest is on top
for i=1:k
        nearest_neighbors(i) = distance(i, 2)  // Nearest neighbors are returned
return nearest_neighbors
```

Figure 4.9. K nearest neighbors algorithm (adopted from [46])

1 for each project feature. Therefore, features with larger values do not dominate the process of finding similarity. The nearest neighbors are given to the ensemble to get estimated effort values as in Equation 4.4. The difference between the actual and the estimated values gives the bias of the ensemble for those projects. The average of biases for nearest neighbors is taken to get the estimated bias for the new project as in Equation 4.5.

$$Bias_{Estimated} = \frac{1}{k} \sum_{i \in N_k} \left(PM_{Actual}(i) - PM_{ENN}(i) \right) \tag{4.5}$$

where N_k is the collection of k nearest neighbors, $PM_{Actual(i)}$ is the actual person month value for project i and $PM_{ENN(i)}$ is the estimated person month value for project i. Lastly, estimated bias value is added to the estimated effort of the ensemble to get the final estimated effort as in Equation 4.6.

$$PM_{ENNA} = PM_{ENN} + Bias_{Estimated} \tag{4.6}$$

The steps of ENNA effort estimation can be seen in Figure 4.10. In step 1, new project is given to the ensemble to get the estimated value, PM_{ENN}. In step 2, projects which are similar to the new project are retrieved from the effort database and given to the

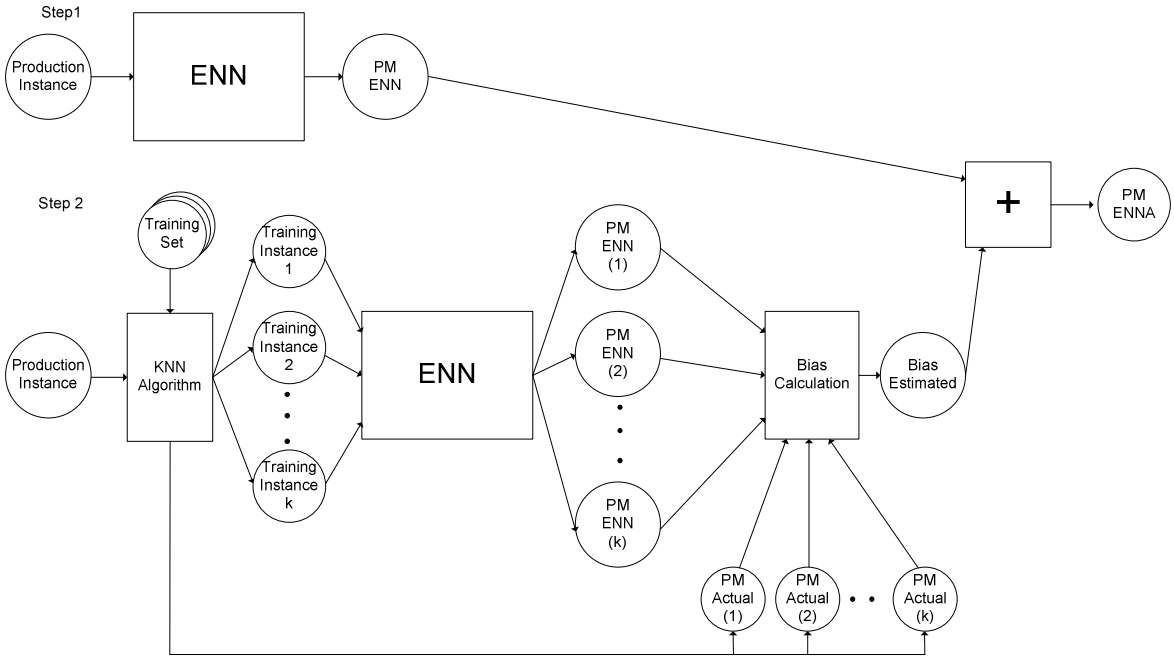

Figure 4.10. ENNA simulation

ensemble to get estimated values, $PM_{ENN}(i)$. Actual efforts for the projects, $PM_{Actual}(i)$, are already known since they are past projects. Therefore, Equation 4.5 gives the estimated bias for the new project, $Bias_{Estimated}$. Finally, $Bias_{Estimated}$ is added to PM_{ENN} to provide PM_{ENNA} as in Equation 4.6.

5. EXPERIMENTS

5.1. Evaluation Criteria

Evaluating the effort estimation performance of a model is a process, which is naturally based on comparing the actual effort with the estimated effort. There are common performance evaluation criteria such as mean magnitude of relative error, median magnitude of relative error and the prediction at a certain level [70]. These criteria are used in this research.

Magnitude of relative error (MRE) defines the ratio of the absolute error to the actual effort as shown in Equation 5.1 [70].

$$MRE(i) = \frac{\left|PM_{Estimated}(i) - PM_{Actual}(i)\right|}{PM_{Actual}(i)} \tag{5.1}$$

where $PM_{Estimated}(i)$ is the estimated person-month value and $PM_{Actual}(i)$ is the actual person-month value for project i.

Calculating MRE for each project in the test set and taking the average gives mean magnitude of relative error (MMRE) as shown in Equation 5.2 [70].

$$MMRE = \frac{1}{N}\sum_{i=1}^{N}\frac{\left|PM_{Estimated}(i) - PM_{Actual}(i)\right|}{PM_{Actual}(i)} \tag{5.2}$$

where N is the number of projects in the test set for which the model is evaluated. The model that obtains the lowest MMRE is preferred.

Since, MMRE is simple average of all MRE's, even one extreme MRE value results in a large deviation of MMRE. Because of this fact, an aggregate measure which is less sensitive to extreme values should be used together along with MMRE. Median of MRE

(MdMRE) is such an evaluation criterion [70]. Like MMRE, the model that obtains the lowest MdMRE is preferred.

Prediction at a certain level (PRED(L)) is another common criterion, which gives the percentage of estimations that are made within the desired MRE range, L [70]. The corresponding equation is shown in Equation 5.3.

$$PRED(L) = \frac{k}{N} \tag{5.3}$$

where k is the number of estimations made with MRE less than or equal to L and N is the number of estimations. Bigger PRED(L) value implies more accuracy. It is suggested that an acceptable value of MRE is 25 per cent or less [70]. Therefore, PRED(25) is used in experiments.

5.2. Datasets

In software effort estimation domain, datasets are usually based on COCOMO, COCOMO II and Function Points. Therefore, the experiments are carried out by using such datasets. These datasets are National Aeronautics and Space Administration (NASA) dataset, NASA 93 dataset, University of South California (USC) dataset, SoftLab Data Repository (SDR) dataset and Desharnais dataset. All of these datasets are publicly available [71, 72].

NASA dataset is a COCOMO based dataset which contains 60 NASA projects from the 1980's and 1990's [71]. These projects are sequencing, avionics and mission planning projects which were implemented in three different NASA software development centers. This dataset is inhomogeneous in terms of size and effort. Project size varies from 2.2 KLOC to 423 KLOC and the actual effort varies from 8.4 PM to 3,240 PM. Secondly, execution time constraint (TIME) and main storage constraint (STOR) values are at least nominal. This fact is not surprising since most NASA applications are hard real time and run on embedded devices, which have scarce main storage. Thirdly, analyst capabilities (ACAP) and programmer capabilities (PCAP) are at least nominal indicating that NASA

analysts and programmers who worked in these projects were over standard. Furthermore, applications experience (AEXP) is at least nominal. This shows project teams are experienced in the project domain. Lastly, required development schedule (SCED) is at least nominal.

NASA 93 dataset is a COCOMO based and it consists of 93 NASA projects from the 1970's and 1980's [71]. These projects are categorized in a variety of application domains such as avionics monitoring, batch data processing, communications, data capture, launch processing, mission planning, operating system and real data processing. These projects were implemented in six different NASA software development centers. In NASA 93 dataset, the size of projects varies from 900 LOC to 980 KLOC and actual effort varies from about 8 PM to 8211 PM.

In 1981, Barry Boehm introduced COCOMO, which is based on the analysis of 63 software development projects in TRW Aerospace [38]. Since Boehm is the member of Center for Systems and Software Engineering at University of South California (USC) this dataset is referred as USC dataset [71]. In USC dataset, project size varies from 2 KLOC to 1,000 KLOC and actual effort varies from 6 PM to 11,400 PM. Likewise, other project attributes are not homogenously distributed.

Boğaziçi University Software Engineering Research Laboratory (SoftLab) members collected 24 projects from different software development organizations in Turkey to form SDR, which is the abbreviation of SoftLab Data Repository [72]. SDR includes automation and banking applications. In SDR, the size of projects varies from 353 LOC to 1250 KLOC and actual effort varies from 2 PM to 342 PM.

Desharnais dataset was collected by J. M. Desharnais for his thesis work [73]. This dataset consists of 81 projects. In this set of experiments, 77 projects are used because 4 projects are incomplete. Unlike the previous datasets, project size is measured in function points (FP) and actual effort is measured in person hours (PH). Team experience, manager experience, year when project ended and programming language are the other attributes of the dataset. The effort of projects varies from 546 PH to 23,940 PH.

The reliability of datasets is essential for correct evaluation of the proposed models. NASA has conducted many researches in software engineering domain and has proposed robust guidelines for collecting project data [74]. Therefore, NASA and NASA 93 datasets are reliable. USC dataset is also reliable because COCOMO is based on this dataset. SDR dataset is collected by Softlab members and this fact makes it reliable. Desharnais dataset can also be assumed to be reliable, because it is used in used in an important software cost estimation research [26].

5.3. Experimental Setup

As discussed in Chapter 4, the proposed model, namely ENNA, is achieved by implementing certain key ideas over the approach of using a single NN for effort estimation.

NN is the basic learning element used in this research. It has one hidden layer with eight hidden nodes. Several hidden node numbers such as two, four, six, eight, 10 and 16 have been tried and the best number is found to be eight for all datasets. The number of hidden nodes was also decided to be eight in a previous effort estimation study [50]. Hyperbolic tangent sigmoid function is used as the transfer function in hidden layer and output layer. NN is trained with Levenberg-Marquardt algorithm, which is very efficient for training moderate sized neural networks, which have up to several hundred weights [66]. The common training parameters for Levenberg-Marquardt algorithm are maximum number of epochs to train, performance goal, initial μ, μ increase factor and μ decrease factor. Maximum number of epochs is decided to be 200. Performance goal is zero. Initial μ, μ increase factor and μ decrease factor are 0.001, 0.1 and 10 respectively.

20 NN's are combined to form ENN. For this purpose, each NN is trained slightly differently by bootstrapping method and their estimations are combined by ART algorithm.

Finally, the concept of associative memory is combined with ENN to form ENNA. K nearest neighbors algorithm seeks for the projects that are most similar to the new project. However, effort estimation datasets are small by nature and consequently there exist a few

similar projects [22, 53]. Therefore, k parameter is kept small, because large k would decrease the accuracy. For a dataset of size N, K nearest algorithm is run with k = N / 10.

In experiments, NN, ENN and ENNA are evaluated so as to observe whether the proposed ideas provide improvement in software effort estimation accuracy. In addition to these models, regression tree (RT) is evaluated to provide the reader an objective comparison. RT is chosen for its popularity [42] and accuracy [50] in software effort estimation.

In any machine learning application, a dataset is divided into two parts as training set and test set [46]. As the names imply, training set is used for training the model and the test set is used for evaluating the performance of the trained model. Collecting project effort data is a laborious process. Therefore, software effort datasets are smaller than datasets in many other domains [22, 53]. In such data scarcity, we preferred keeping the training set as big as possible to provide better estimation accuracy. For a dataset of size N, about N / 6 of the projects are used for testing whereas the remaining projects are used for training. Parameter values are listed in Table 5.1.

Table 5.1. Experiment parameters

Dataset	Training	Test	Total	k
NASA	50	10	60	6
NASA 93	78	15	93	9
USC	53	10	63	6
SDR	20	4	24	2
Desharnais	64	13	77	8

For evaluating the performance of models, holdout method with random sub sampling is used [75]. A predetermined number of test instances are chosen randomly from the original set and the remaining instances form the training set. The model is trained and tested with mutually exclusive training set and test set respectively. This process is repeated 25 times and best case, worst case, average case and standard deviation of the evaluation criteria are reported. Therefore, it is possible to observe the average estimation

performance as well as the boundary points. Additionally, t-test is conducted to compare the models [76].

In this research, Matlab 6.5 Release 13, Matlab Neural Network Toolbox and Matlab Statistics Toolbox are used for implementation. Matlab Neural Network Toolbox is used for training and simulating MLP's whereas Matlab Statistics Toolbox is used for fitting and simulating RT. The remaining parts are self-developed. The reasons for choosing Matlab are its simplicity and its power in machine learning domain.

A software project is defined by a number of features a few of which are programmer capability, product complexity and use of software tools. The managers may wonder about the answer of this question: "Which features are more essential for the accuracy of effort estimation?" The answer can guide them to set more value upon these features. As an example, if programmer capability is discovered to affect the effort much more than the other features then the company would prefer hiring senior programmers even by giving them "above the nominal" salaries. As an additional focus point of this research, the subset of more important features is formed and used to estimate effort by using the proposed models. This experiment answers this question: "Does feature subset selection improve the accuracy of the proposed models?" If so, the cost of extracting less important features is saved and more accuracy is obtained.

In machine learning literature, reducing the number of dimensions has been a point of focus because of several reasons. Firstly, the number of input dimensions directly affects a learning algorithm's complexity, which increases memory and computation requirements. Secondly, if an input is unnecessary, the extraction cost is saved. Thirdly, models, which have fewer inputs, are more robust on small datasets because they have variance. Lastly, analyzing the structure of data is easier when they are represented in fewer dimensions [46].

There are two methods for reducing dimensionality, which are feature extraction and feature selection. In feature extraction, the original features are combined to provide a new set of features. In feature selection, more important features are selected and the others are ignored. Dimensionality reduction has been widely used in software effort estimation

domain. In a research, the effect of feature extraction on the performance of a learning-oriented model, support vector regression, was evaluated [49]. In another research, it was proved that feature subset selection improves accuracy of COCOMO, a parametric method [51, 52].

In this research, Wrapper algorithm [77] is used for feature subset selection and it is introduced in the following subsection.

5.3.1. Wrapper

Wrapper is a supervised feature subset selection algorithm [77] and it is previously discovered that it may improve the accuracy of COCOMO [51, 52]. Wrapper starts with no features and adds them one by one until no further accuracy improvement is obtained. At each step, the target model is trained and tested by using feature subset and the feature whose addition results in the smallest error is selected. If addition of a new feature with smallest error does not provide accuracy improvement, it is still added and named as a stale feature. If the number of stale features reaches a predetermined number, the algorithm stops and slack features are removed from the feature set. In this research, maximum slack number is decided to be five as in previous effort estimation studies with Wrapper [51, 52].

Wrapper algorithm is shown in Figure 5.3. Since ENNA is the final model, it is used as the target model. In addition to that, error measure is selected to be MMRE.

```
F = Ø                                   // Start with no features
S = Ø                                   // Start with no stale features
min_MMRE = MAX
max_stale = 5
stale = 0
for i=1:number_of_features
        for j=1:number_of_features      // Initialize MMRE matrix at each step
                MMRE(j,1) = MAX
                MMRE(j,2) = 0
        for all f_k ∉ F                 // Calculate MMRE for all possible cases
```

```
        MMRE(k, 1) = train and test ENNA(F ∪ f_k)
    sort(MMRE, 1)                       // Sort calculated MMRE values
    if(MMRE(1,1) < min_MMRE)            // If accuracy improvement is obtained
        min_MMRE = MMRE(1,1)
        F = F ∪ f_k
    else                                // If no accuracy improvement is not obtained
        if(stale < max_stale)           // Stale addition
            stale = stale + 1
            F = F ∪ f_k
            S = S ∪ f_k
        else                            // Remove stale features and stop
            F = F – S
            break
```

Figure 5.1. Wrapper algorithm (adopted from [77])

5.4. Experimental Results

5.4.1. Experimental Results without Feature Subset Selection

In this set of experiments, RT, NN, ENN and ENNA are trained and tested by using all features. Table 5.1 shows the best case, the worst case, the average estimation performance as well as standard deviation.

For NASA dataset, ENN has average MMRE = 55.71 per cent, average MdMRE = 41.82 per cent, average PRED(25) = 32.00 per cent whereas NN has average MMRE = 184.46 per cent, average MdMRE = 97.97 per cent and average PRED(25) = 5.00 per cent. These values show that ENN performs better than NN. When the limit points are taken into consideration, the worst case estimation performance values of ENN, which are MMRE = 66.13 per cent, MdMRE = 47.39 per cent, PRED(25) = 30.00 per cent is still better than the best case estimation performance values of NN, which are MMRE = 122.14 per cent, MdMRE = 89.63 per cent and PRED(25) = 20.00 per cent. This fact supports the fact that ENN outperforms NN. ENN is also better than RT, which has average MMRE = 66.02 per

Table 5.2. Experimental results without feature subset selection

		MMRE (%)			MdMRE (%)			PRED(25) (%)		
		Best	Worst	Average (Std Dev)	Best	Worst	Average (Std Dev)	Best	Worst	Average (Std Dev)
NASA	RT	48.03	82.47	66.02 (11.64)	37.76	64.34	46.14 (7.99)	30.00	10.00	24.00 (8.43)
	NN	122.14	385.29	184.46 (82.22)	89.63	118.75	97.97 (8.00)	20.00	0.00	5.00 (8.50)
	ENN	49.23	66.13	55.71 (5.11)	32.50	47.39	41.82 (4.40)	40.00	30.00	32.00 (4.22)
	ENNA	36.04	55.66	47.66 (6.62)	26.07	41.10	33.17 (4.34)	50.00	30.00	38.00 (6.33)
NASA 93	RT	334.97	532.68	394.34 (65.11)	51.71	67.10	58.94 (5.83)	26.67	6.67	20.67 (6.63)
	NN	232.13	1,925.40	527.69 (502.77)	94.19	197.69	107.76 (31.86)	13.33	0.00	6.00 (4.92)
	ENN	50.49	72.76	62.28 (7.56)	39.10	59.64	49.54 (8.31)	40.00	26.67	30.67 (4.66)
	ENNA	46.43	64.86	54.35 (7.03)	27.05	41.44	36.91 (4.19)	40.00	26.67	32.67 (3.78)
USC	RT	176.02	918.37	323.23 (217.49)	76.02	311.88	135.72 (75.20)	10.00	0.00	1.00 (3.16)
	NN	1,934.80	14,829.00	5,584.90 (4,848.40)	96.71	2,717.20	657.39 (914.76)	0.00	0.00	0.00 (0.00)
	ENN	73.42	127.16	105.52 (20.46)	54.10	78.71	67.83 (8.80)	20.00	10.00	18.00 (4.22)
	ENNA	56.99	129.06	85.44 (25.83)	49.25	71.61	61.42 (7.12)	40.00	20.00	24.00 (6.99)
SDR	RT	120.07	882.45	434.77 (241.59)	56.61	796.51	205.64 (266.72)	25.00	0.00	10.00 (12.91)
	NN	315.17	12,774.00	2,274.10 (3,791.30)	87.67	17,000.00	1,892.00 (5,309.60)	25.00	0.00	15.00 (12.91)
	ENN	36.25	83.40	49.85 (13.47)	30.41	49.82	38.60 (6.27)	50.00	25.00	47.50 (7.91)
	ENNA	22.23	53.45	39.88 (10.95)	10.90	44.42	29.10 (11.92)	75.00	25.00	55.00 (15.81)
DESHARNAIS	RT	68.07	251.80	119.36 (58.74)	39.74	62.73	47.67 (7.64)	30.77	15.39	23.85 (5.68)
	NN	83.28	291.93	154.20 (69.00)	66.01	140.20	84.91 (23.57)	23.08	0.00	14.62 (8.47)
	ENN	52.52	64.36	57.85 (3.42)	47.40	53.64	50.06 (2.21)	30.77	15.39	23.08 (5.13)
	ENNA	42.05	55.76	49.82 (5.05)	35.59	49.03	44.13 (4.57)	46.15	23.08	33.08 (7.30)

cent, average MdMRE = 46.14 per cent and average PRED(25) = 24.00 per cent. ENNA provides further improvement over ENN and achieves the best estimation performance values which are average MMRE = 47.66 per cent, average MdMRE = 33.17 per cent and average PRED(25) = 38.00 per cent.

For NASA 93 dataset, ENN achieves average MMRE = 62.28 per cent, average MdMRE = 49.54 per cent and average PRED(25) = 30.67 per cent. These values indicate that ENN is better than NN, which has average MMRE = 527.69 per cent, average MdMRE = 107.76 per cent and average PRED(25) = 6.00 per cent. The worst case estimation performance values of ENN, which are MMRE = 72.76 per cent, MdMRE = 59.64 per cent and PRED(25) = 26.67 per cent, are better than even the best case estimation values of NN, which are MMRE = 232.13 per cent, MdMRE = 94.19 per cent, and PRED(25) = 13.33 per cent. Limit points of estimation performance values support the fact that ENN performs better than NN. ENN is also better than RT, which has average MMRE = 394.34 per cent, average MdMRE = 58.94 per cent and average PRED(25) = 20.67 per cent. ENNA provides further improvement over ENN and achieves the best estimation performance values which are average MMRE = 54.35, average MdMRE = 36.91 per cent and average PRED(25) = 32.67 per cent.

For the experiments conducted by using USC dataset, ENN has average MMRE = 105.52 per cent, average MdMRE = 67.83 per cent and average PRED(25) = 18.00 per cent. NN has terrific evaluation values, which are average MMRE = 5,584.90 per cent, average MdMRE = 657.39 per cent and average PRED(25) = 0.00 per cent. NN's estimation performance values are explicitly worse than the values for ENN. When limit points are examined, it is observed that the worst case estimation performance values of ENN, which are MMRE =127.16 per cent, MdMRE = 78.71 per cent, PRED(25) = 10.00 per cent, are even better than the best estimation values of NN, which are MMRE = 1934.80 per cent, MdMRE = 96.71 per cent and PRED(25) = 0.00 per cent. When the performance of ENN is compared with the performance of RT, the values show that ENN outperforms RT, which has average MMRE = 323.23 per cent, average MdMRE = 135.72 per cent and average PRED(25) = 1.00 per cent. ENNA achieves the best estimation performance values as average MMRE = 85.44 per cent, average MdMRE = 61.42 per cent and average PRED(25) = 24.00 per cent.

ENN has average MMRE = 49.85 per cent, average MdMRE = 38.60 per cent average and average PRED(25) = 47.50 per cent for SDR dataset. On the other hand, NN can achieve average MMRE = 2,274.10 per cent, average MdMRE = 1,892.00 per cent and average PRED(25) = 15.00 per cent. These values show that, ENN is much better than NN. The worst case estimation performance values of ENN, which are MMRE = 83.40 per cent, MdMRE = 49.82 per cent and PRED(25) = 25.00 per cent, are either better than or equal to the best case estimation performance values of NN, which are MMRE = 315.17 per cent, MdMRE = 87.67 per cent, and PRED(25) = 25.00 per cent. Limit points of the performance values support the fact that ENN performs better than NN. ENN is also better than RT, which has average MMRE = 437.77 per cent, average MdMRE = 205.64 per cent and average PRED(25) = 10.00 per cent. ENNA provides further improvement over ENN and achieves the best estimation performance values which are average MMRE = 39.88 per cent, average MdMRE = 29.10 per cent and average PRED(25) = 55.00 per cent.

For the experiments conducted by using Desharnais dataset, ENN has average MMRE = 57.85 per cent, average MdMRE = 50.06 per cent and average PRED(25) = 23.08 per cent whereas NN has average MMRE = 154.20 per cent, average MdMRE = 84.91 per cent and average PRED(25) = 14.62 per cent. These values show that ENN performs better than NN. When the limit points are taken into account, the worst case estimation performance values of ENN, which are MMRE = 64.36 per cent, MdMRE = 53.64 per cent, PRED(25) = 15.39 per cent is better than even the best estimation values of NN, which are MMRE = 83.28 per cent, MdMRE = 66.01 per cent and PRED(25) = 23.08 per cent. ENN is also better than RT, which has average MMRE = 119.36 per cent, MdMRE = 47.67 per cent and PRED(25) = 23.85 per cent. ENNA achieves the best estimation performance values as average MMRE = 49.82 per cent, average MdMRE = 44.13 per cent and average PRED(25) = 33.08 per cent.

According to the evaluation results, the ideas we proposed provide accuracy improvement. Furthermore, ENN and ENNA have smaller standard deviation values which imply that they are stable algorithms [46].

T-test results, which are listed in Table 5.3, support the evaluation results, which are listed in Table 5.2. For NASA, NASA 93 and USC datasets, ENN is statistically better

than RT and NN in terms of all evaluation criteria. Furthermore, ENNA is better than RT, NN and ENN for all datasets and all evaluation criteria.

Table 5.3. T-test results without feature subset selection

is better than		MMRE			MdMRE			PRED(25)		
		RT	NN	ENN	RT	NN	ENN	RT	NN	ENN
NASA	ENN	TRUE	TRUE		TRUE	TRUE		TRUE	TRUE	
	ENNA	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
NASA 93	ENN	TRUE	TRUE		TRUE	TRUE		TRUE	TRUE	
	ENNA	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
USC	ENN	TRUE	TRUE		TRUE	TRUE		TRUE	TRUE	
	ENNA	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
SDR	ENN	TRUE	TRUE		TRUE	FALSE		TRUE	TRUE	
	ENNA	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE
DESHARNAIS	ENN	TRUE	TRUE		FALSE	TRUE		FALSE	TRUE	
	ENNA	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE

5.4.2. Experimental Results with Feature Subset Selection

Wrapper algorithm is applied to five datasets and the selected features are used for training and testing ENNA model whereas other features are simply ignored [77].

Table 5.4 shows the selected features for NASA dataset. Database size (DATA), execution time constraint (TIME), required development schedule (SCED) and lines of code (LOC) are selected by Wrapper.

Table 5.4. Selected features for NASA dataset

COCOMO Attributes		Selected
Cost Drivers		
Product Attributes		
Required software reliability	RELY	✗
Database size	DATA	✓
Product complexity	CPLX	✗
Computer Attributes		
Execution time constraint	TIME	✓
Main storage constraint	STOR	✗
Virtual machine volatility	VIRT	✗
Computer turnaround time	TURN	✗
Personnel Attributes		
Analyst capabilities	ACAP	✗
Applications experience	AEXP	✗
Programmer capabilities	PCAP	✗
Virtual machine experience	VEXP	✗
Programming language experience	LEXP	✗
Project Attributes		
Modern programming practices	MODP	✗
Use of software tools	TOOL	✗
Required development schedule	SCED	✓
Lines of Code	LOC	✓

Wrapper selected five features for NASA 93 dataset as shown in Table 5.5. Execution time constraint (TIME), virtual machine experience (VEXP), programming language experience (LEXP), required development schedule (SCED) and lines of code (LOC) are the selected features. TIME, SCED and LOC have also been selected for NASA dataset. This fact proves that NASA and NASA 93 datasets share some common characteristics, because they have been developed in the same organization.

Table 5.5. Selected features for NASA 93 dataset

COCOMO Attributes		Selected
Product Attributes		
Required software reliability	RELY	✗
Database size	DATA	✗
Product complexity	CPLX	✗
Computer Attributes		
Execution time constraint	TIME	✓
Main storage constraint	STOR	✗
Virtual machine volatility	VIRT	✗
Computer turnaround time	TURN	✗
Personnel Attributes		
Analyst capabilities	ACAP	✗
Applications experience	AEXP	✗
Programmer capabilities	PCAP	✗
Virtual machine experience	VEXP	✓
Programming language experience	LEXP	✓
Project Attributes		
Modern programming practices	MODP	✗
Use of software tools	TOOL	✗
Required development schedule	SCED	✓
Lines of Code	LOC	✓

Table 5.6 shows the selected features for USC dataset. Main storage constraint (STOR), required development schedule (SCED) and lines of code (LOC) are selected by Wrapper. SCED and LOC have also been selected for NASA and NASA 93 datasets.

Table 5.6. Selected features for USC dataset

COCOMO Attributes		Selected
Product Attributes		
Required software reliability	RELY	✗
Database size	DATA	✗
Product complexity	CPLX	✗
Computer Attributes		
Execution time constraint	TIME	✗
Main storage constraint	STOR	✓
Virtual machine volatility	VIRT	✗
Computer turnaround time	TURN	✗
Personnel Attributes		
Analyst capabilities	ACAP	✗
Applications experience	AEXP	✗
Programmer capabilities	PCAP	✗
Virtual machine experience	VEXP	✗
Programming language experience	LEXP	✗
Project Attributes		
Modern programming practices	MODP	✗
Use of software tools	TOOL	✗
Required development schedule	SCED	✓
Lines of Code	LOC	✓

Wrapper selected only two features for SDR dataset as shown in Table 5.7. These features are team cohesion (TEAM) and lines of code (LOC). LOC have also been selected for NASA, NASA 93 and USC datasets. This is not surprising though because size of the project has a direct effect on the effort.

Table 5.7. Selected features for SDR dataset

COCOMO II Attributes		Selected
Cost Drivers		
Product Factors		
Required software reliability	RELY	✕
Database size	DATA	✕
Product complexity	CPLX	✕
Required reusability	RUSE	✕
Documentation match to life-cycle needs	DOCU	✕
Platform Factors		
Execution time constraint	TIME	✕
Main storage constraint	STOR	✕
Platform volatility	PVOL	✕
Personnel Factors		
Analyst capability	ACAP	✕
Programmer capability	PCAP	✕
Applications experience	AEXP	✕
Platform experience	PEXP	✕
Language and tool experience	LTEX	✕
Personnel continuity	PCON	✕
Project Factors		
Use of software tools	TOOL	✕
Required development schedule	SCED	✕
Multisite development	SITE	✕
Scale Drivers		
Precedentedness	PREC	✕
Development Flexibility	FLEX	✕
Architecture / Risk Resolution	RESL	✕
Team Cohesion	TEAM	✔
Process Maturity	PMAT	✕
Lines of Code	LOC	✔

Table 5.8 shows the features which are selected for Desharnais dataset. The selected features are number of entities, nonadjusted function points and programming language.

Table 5.8. Selected features for Desharnais dataset

Desharnais Dataset Attributes	Selected
Team experience	✗
Manager experience	✗
Year project ended	✗
Transactions	✗
Entities	✓
Nonadjusted function points	✓
Function point complexity adjustment factor	✗
Adjusted function points	✗
Programming language	✓

Table 5.9 shows the best case, the worst case and the average estimation performance values of ENNA as well as standard deviation values for datasets whose dimensions are reduced by using Wrapper.

Wrapper algorithm selected database size (DATA), execution time constraint (TIME), required development schedule (SCED) and lines of code (LOC) from NASA dataset. ENNA is trained and tested with these features to give estimation performance results as average MMRE = 35.51 per cent, average MdMRE = 21.63 per cent and average PRED(25) = 60.00 per cent. As shown in Table 5.2, training and testing ENNA by using all features give average MMRE = 47.66 per cent, average MdMRE = 33.17 per cent and average PRED(25) = 38.00 per cent. These results imply that feature subset selection improves the performance of ENNA on NASA dataset.

Wrapper selected five features for NASA 93 dataset which are execution time constraint (TIME), virtual machine experience (VEXP), programming language experience (LEXP), required development schedule (SCED) and lines of code (LOC). ENNA is trained and tested by using these features to give estimation performance results as average MMRE = 55.03 per cent, average MdMRE = 32.03 and average PRED(25) = 43.33. As shown in Table 5.2, training and testing ENNA by using all features give average MMRE = 54.35, average MdMRE = 36.91 per cent and average PRED(25) = 32.67 per cent. Concerning MMRE and MdMRE criteria, same accuracy is obtained by using fewer

Table 5.9. Experimental results with feature subset selection

ENNA	MMRE (%)			MdMRE (%)			PRED(25) (%)		
	Best	Worst	Average (Std Dev)	Best	Worst	Average (Std Dev)	Best	Worst	Average (Std Dev)
NASA	31.85	38.30	35.51 (2.25)	17.82	27.52	21.63 (3.20)	70.00	50.00	60.00 (8.17)
NASA 93	33.14	70.95	55.03 (11.79)	22.72	39.34	32.03 (6.83)	53.33	33.33	43.33 (6.48)
USC	49.52	106.61	69.42 (17.38)	23.78	58.84	45.83 (11.63)	60.00	20.00	34.00 (13.50)
SDR	16.68	55.04	37.92 (12.02)	16.15	45.17	30.40 (9.71)	50.00	50.00	50.00 (0.00)
DESHARNAIS	40.59	59.77	50.43 (6.16)	20.71	41.84	33.07 (7.61)	61.54	30.77	40.77 (10.91)

features. In terms of PRED(25) criterion, feature subset selection improves estimation performance. Therefore, feature subset selection provides gain on NASA 93 dataset.

For USC dataset, main storage constraint (STOR), required development schedule (SCED) and lines of code (LOC) are selected by Wrapper algorithm. ENNA is trained and tested with these features to give estimation performance results as average MMRE = 69.42 per cent, average MdMRE = 45.83 per cent and average PRED(25) = 34.00 per cent. As shown in Table 5.2, training and testing ENNA by using all features give average MMRE = 85.44 per cent, average MdMRE = 61.42 per cent and average PRED(25) = 24.00 per cent. These results imply that feature subset selection improves the performance of ENNA on USC dataset.

Wrapper algorithm selected only two features for SDR dataset which are team cohesion (TEAM) and lines of code (LOC). ENNA is trained and tested by using these features to give estimation performance results as average MMRE = 37.92 per cent, average MdMRE = 30.40 and average PRED(25) = 50.00. As shown in Table 5.2, training and testing ENNA by using all feature give average MMRE = 39.88, average MdMRE = 29.10 per cent and average PRED(25) = 55.00 per cent. These results imply that same accuracy is obtained by using fewer features.

For Desharnais dataset, the selected features are number of entities, nonadjusted function points and programming language. ENNA is trained and tested with these features

to give estimation performance results as average MMRE = 50.43 per cent, average MdMRE = 33.07 per cent and average PRED(25) = 40.77 per cent. As shown in Table 5.2, training and testing ENNA by using all features give average MMRE = 49.82 per cent, average MdMRE = 44.13 per cent and average PRED(25) = 33.08 per cent. Concerning MMRE criterion, same accuracy is obtained with fewer features. In terms of MdMRE and PRED(25), feature subset selection improves the performance of ENNA.

The effect of feature subset selection by using Wrapper is summarized in Table 5.10. The accuracy of ENNA either stays the same or increases when the dimensionality of the datasets are reduced with Wrapper algorithm. Moreover, the running time of ENNA is reduced and the cost of extracting less important features is saved [46].

Table 5.10. Effect of wrapper on estimation accuracy

ENNA	MMRE (%)	MdMRE (%)	PRED(25) (%)
NASA	BETTER	BETTER	BETTER
NASA 93	SIMILAR	SIMILAR	BETTER
USC	BETTER	BETTER	BETTER
SDR	SIMILAR	SIMILAR	SIMILAR
DESHARNAIS	SIMILAR	BETTER	BETTER

6. CONCLUSION

We have been motivated to accurately estimate the effort of a software development project as one of the major challenges in the project management. There are parametric models such as COCOMO which need company specific calibration. Therefore learning based models can better be generalized and their calibration give far better accurate estimations.

In learning based models estimation accuracy depends on either the strength of the prediction algorithm or better understanding of the software engineering content of the data. In our research, we focused primarily on improving the prediction algorithm and then to better understand the features that compose the effort related information in software engineering projects.

We have developed a novel algorithm called ENNA which is based on neural networks. Neural networks have been a widely used learning algorithm in software effort estimation. We used ensemble method and associative memory to improve the performance of NN algorithm for better effort estimation. We validated the robustness of algorithm by comparing it to NN and Regression Tree algorithms. In our experiments we used a wide selection of public and private datasets (NASA, NASA 93, USC, SDR and Desharnais) to validate our proposed algorithm. Our results showed that ENNA is the best performing algorithm with MMRE = 39.88 per cent, MdMRE = 29.10 per cent and PRED(25) = 55.00 per cent.

In order to further improve the prediction accuracy of ENNA we claimed that we needed to better understand software engineering knowledge of our data, i.e. deciding on which features to select and which features are more important than others?. We have then applied a well accepted feature subset selection algorithm called Wrapper to use the subset of features. Our experiment results showed that the more we understand the software engineering content of our data the better we can predict effort. Our results improved to

MMRE = 35.51 per cent, MdMRE = 21.63 per cent and PRED(25) = 60.00 per cent with feature subset selection.

A possible future direction would be to test our proposed algorithm with company and/ or domain specific data (within company and/ or within domain). Another future direction could be to estimate effort intervals by defining the problem as a classification problem. Additionally, we will combine the concept of associative memory with other learning–oriented models such as regression trees. Therefore, we will observe whether the estimation accuracy is improved for these models as is the case with neural networks. Being hopeful with the accuracy improvement that feature subset selection provided, we will also concentrate on using other feature subset selection algorithms such as correlation-based feature selection.

REFERENCES

1. Desmond, J. P., "Applications Go Worldwide", *Software Magazine*, October 2007.

2. Boehm, B. and K. Sullivan, "Software economics: status and prospects", *Information and Software Technology*, Vol. 41, No. 14, pp. 937-946, November 1999.

3. Nelson, E. A., *Management Handbook for the Estimation of Computer Programming Costs*, Systems Development Corporation, California, March 1967.

4. Boehm, B., C. Abts and S. Chulani, "Software Development Cost Estimation Approaches – A Survey", *Annals of Software Engineering*, Vol. 10, No. 1-4, pp. 177-205, November 2000.

5. Heemstra, F. J., "Software Cost Estimation Models", *'Next Decade in Information Technology', Proceedings of the Fifth Jerusalem Conference on Information Technology*, pp. 286-297, October 1990.

6. Ahn, Y., J. Suh, S. Kim and H. Kim, "The Software Maintenance Project Effort Estimation Model Based on Function Points", *Journal of Software Maintenance and Evolution: Research and Practice*, Vol. 15, No. 2, pp. 71-85, April 2003.

7. Adrangi, B. and W. Harrison, "Effort Estimation in a System Development Project", *Journal of Systems Management*, Vol. 36, No. 8, pp. 21-23, 1987.

8. Banker, R. D., H. Chang and C. F. Kemerer, "Evidence on Economies of Scale in Software Development", *Information and Software Technology*, Vol. 36, No. 5, pp. 275-282, 1994.

9. Benediktsson, O., D. Dalcher and K. Reed, “COCOMO-Based Effort Estimation for Iterative and Incremental Software Development”, *Software Quality Journal*, Vol. 11, No. 4, pp. 265-281, 2003.

10. Harrison, W. and B. Adrangi, “The Role of Programming Language in Estimating Software Development Costs”, *Journal of Management Information Systems*, Vol. 3, No. 3, pp. 101-110, 1987.

11. Bielak, J., “Improving Size Estimates Using Historical Data”, *IEEE Software*, Vol. 17, No. 6, pp. 27-35, November-December 2000.

12. Kaplan, H. T., “The Ada COCOMO Cost Estimating Model and VASTT Development Estimates vs. Actuals”, *Vitro Technical Journal*, Vol. 9, No. 1, pp. 48-60, 1991.

13. Briand, L. C., K. El Emam and I. Wieczorek, “Explaining the Cost of European Space and Military Projects”, *Proceedings of the 21st International Conference on Software Engineering*, IEEE Computer Society Press, pp. 303-312, May 1999.

14. Menzies, T., D. Port, Z. Chen and J. Hihn, “Simple Software Cost Analysis: Safe or Unsafe?”, *Proceedings of the 2005 Workshop on Predictor Models in Software Engineering*, ACM, pp. 1-6, May 2005.

15. Menzies, T., Z. Chen, J. Hihn and K. Lum, “Selecting Best Practices for Effort Estimation”, *IEEE Transactions on Software Engineering*, Vol. 32, No. 11, pp. 883-895, November 2006.

16. Kemerer, C. F., “An Empirical Validation of Software Cost Estimation Models”, *Communications of the ACM*, Vol. 30, No. 5, pp. 416-429, May 1987.

17. Venkatachalam, A. R., “Software Cost Estimation Using Artificial Neural Networks”, *Proceedings of 1993 International Joint Conference on Neural Networks*, Vol. 1, pp. 987-990, October 1993.

18. Srinivasan, K. and D. Fisher, ”Machine Learning Approaches to Estimating Software Development Effort”, *IEEE Transactions on Software Engineering*, Vol. 21, No. 2, pp. 126-137, February 1995.

19. Kim, Y. and K. Lee, “A Comparison of Techniques for Software Development Effort Estimating”, *System Integration*, 2005.

20. Finnie, G. R. and G. E. Wittig,, “AI Tools for Software Development Effort Estimation”, *Proceedings of International Conference on Software Engineering: Education and Practice*, pp. 346-353, 1996.

21. Tadayon, N., “Neural Network Approach for Software Cost Estimation”, *International Conference on Information Technology: Coding and Computing*, Vol. 2, pp. 815-818, April 2005.

22. Boetticher, G. D., “Using Machine Learning to Predict Project Effort: Empirical Case Studies in Data-Starved Domains”, *Model Based Requirements Workshop*, 2001.

23. Idri, A., T. M. Khoshgoftaar and A. Abran, “Can Neural Networks be easily Interpreted in Software Cost Estimation?”, *Proceedings of the 2002 IEEE International Conference on Fuzzy Systems*, Vol. 2, pp. 1162-1167, May 2002.

24. Briand, L. C., K. El Emam, D. Surmann and I. Wieczorek, “An Assessment and Comparison of Common Software Cost Estimation Techniques”, *Proceedings of the 21st International Conference on Software Engineering*, IEEE Computer Society Press, pp. 313-322, May 1999.

25. Briand, L. C., T. Langley and I. Wieczorek, “A Replicated Assessment and Comparison of Common Software Cost Estimation Techniques”, *Proceedings of the 22nd International Conference on Software Engineering*, pp. 377-386, June 2000.

26. Shepperd, M. and C. Schofield, “Estimating Software Project Effort Using Analogies” *IEEE Transactions on Software Engineering*, Vol. 23, No. 12, pp. 736-743, November 1997.

27. Idri, A., A. Abran and T. M. Khoshgoftaar, “Fuzzy Analogy: A New Approach for Software Cost Estimation”, *11st International Workshop on Software Measurement*, pp. 93-101, August 2001.

28. Idri, A., A.Abran and T. M. Khoshgoftaar, “Estimating Software Project Effort by Analogy Based on Linguistic Values”, *Proceedings of the Eighth IEEE Symposium on Software Metrics*, pp. 21-30, 2002.

29. Khoshgoftaar, T. M., A. Idri and A. Abran, “Investigating Soft Computing in Case-Based Reasoning for Software Cost Estimation”, *International Journal of Engineering Intelligent Systems for Electrical Engineering and Communications*, Vol. 10, No. 3, pp. 147-158, 2002.

30. Burgess, C. J. and M. Lefley, “Can genetic programming improve software effort estimation? A comparative evaluation”, *Information and Software Technology*, Vol. 43, No. 14, pp. 863-873, December 2001.

31. Shan, Y., R. I. McKay, C. J. Lokan and D. L. Essam, “Software Project Effort Estimation Using Genetic Programming”, *International Conference on Communications, Circuits and Systems and West Sino Expositions*, Vol. 2, pp. 1108-1112, June-July 2002.

32. Lefley, M. and M. J. Shepperd, “Using Genetic Programming to Improve Software Effort Estimation Based on General Datasets”, *Proceedings of Genetic and Evolutionary Computation Conference (GECCO) 2003*, Springer, pp. 2477-2487, July 2003.

33. Abeles, M., *Corticotronics: Neural circuits of the cerebral cortex*, Cambridge University Press, New York, 1991.

34. Villa, A. E. P., I. V. Tetko, B. Hyland and A. Najem, "Spatiotemporal Activity Patterns of Rat Cortical Neurons Predict Responses in a Conditioned Task", *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 96, No. 3, pp. 1106-1111, February 1999.

35. Putnam, L. H., "A General Empirical Solution to the Macro Software Sizing and Estimating Problem", *IEEE Transactions on Software Engineering*, Vol. SE 4, No. 4, pp. 345-361, July 1978.

36. Albrecht, A. J., "Measuring Application Development Productivity", *Proceedings of SHARE/GUIDE IBM Applications Development Symposium*, pp. 83-92, October 1979.

37. Rubin, H. A., "Macroestimation of Software Development Parameters: The ESTIMACS System", *SOFTFAIR Conference on Software Development Tools, Techniques and Alternatives*, IEEE Press, pp. 109-118, July 1983.

38. Boehm, B. W., *Software Engineering Economics*, Prentice Hall, New Jersey, 1981.

39. University of Southern California, *USC COCOMO Reference Manual*, 1994.

40. University of Southern California, *COCOMO II Model Definition Manual*, 1999.

41. Boehm, B. W., B. Steece and R. Madachy, *Software Cost Estimation with COCOMO II*, Prentice Hall, New Jersey, 2000.

42. Jorgensen, M. and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies", *IEEE Transactions on Software Engineering*, Vol. 33., No. 1, pp. 33-53, January 2007.

43. NASA, *Parametric Cost Estimating Handbook*, 2002

44. Aamodt, A. and E. Plaza, "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches", *Artificial Intelligence Communications*, Vol. 7, No. 1, pp. 39-59, March 1994.

45. Breiman, L., J. H. Friedman, R. A. Olshen and C. J. Stone, *Classification and regression trees*, Wadsworth, 1984.

46. Alpaydın, E., *Introduction to Machine Learning*, MIT Press, 2004.

47. Hornik, K., M. Stinchcombe and H. White, "Multilayer Feedforward Networks are Universal Approximators", *Neural Networks*, Vol. 2, No. 5, pp. 359-366, 1989.

48. Charles, D., *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*, 1859.

49. Turhan, B., O. Kutlubay and A. Bener, "Evaluation of Feature Extraction Methods on Software Cost Estimation", *First International Symposium on Empirical Software Engineering and Measurement*, p. 497, 2007.

50. Baskeles, B., B. Turhan and A. Bener, "Software Effort Estimation Using Machine Learning Methods", ISCIS 2007, Ankara, Turkey, November 7-9 2007.

51. Chen, Z., T. Menzies, D. Port and B. Boehm, "Feature Subset Selection Can Improve Software Estimation Accuracy", *ACM SIGSOFT Software Engineering Notes*, Vol. 30, No. 4, pp. 1-6, July 2005.

52. Chen, Z., T. Menzies, D. Port and B. Boehm, "Finding the Right Data for Software Cost Modeling", *IEEE Software*, Vol. 22, No. 6, pp. 38-46, November-December 2005.

53. Kitchenham, B., E. Mendes and G. H. Travassos, "A Systematic Review of Cross- vs. Within- Company Cost Estimation Studies", *Proceedings of the Tenth International Conference on Evaluation and Assessment in Software Engineering (EASE)*, Keele University, Staffordshire, UK, 10-12 April 2006, pp. 79-88, April 2006.

54. Jeffery, R., M. Ruhe and I. Wieczorek, "A comparative study of two software development cost modeling techniques using multi-organizational and company-specific data", *Information and Software Technology*, Vol. 42, No. 14, pp. 1009-1016, November 2000.

55. Mendes, E., S. Di Martino, F. Ferucci and C. Gravino, "Effort estimation: how valuable is it for a web company to use a cross-company data set, compared to using its own single-company data set?", *Proceedings of the Sixteenth International Conference on World Wide Web*, Banff, Alberta, Canada, pp. 963-972, May 2007.

56. German, S., E. Bienenstock and R. Doursat, "Neural Networks and the Bias/Variance Dilemma", *Neural Computation*, Vol. 4, No. 1, pp. 1-58, 1992.

57. Kohonen, T., "Self-Organization and Associative Memory", *Springer Series in Information Sciences*, Vol. 8, 1984.

58. Tetko, I. V., "Associative Neural Network", *Neural Processing Letters*, Vol. 16, No. 2, pp. 187-199, October 2002.

59. Sharkey, A. J. C. and N. E. Sharkey, "Combining diverse neural nets", *The Knowledge Engineering Review*, Vol. 12, No. 3, pp. 231-247, September 1997.

60. Liu, B., Q. Cui, T. Jiang and S. Ma, "A combinational feature selection and ensemble neural network method for classification of gene expression data", *BMC Bioinformatics*, September 2004.

61. Lopez-Munoz, F., J. Boya and C. Alamo, "Neuron theory, the cornerstone of neuroscience, on the centenary of the Nobel Prize award to Santiago Ramon y Cajal", *Brain Research Bulletin*, Vol. 70, No. 4-6, pp. 391-405, August 2006.

62. Funahashi, K., "On the approximate realization of continuous mappings by neural networks", *Neural Networks*, Vol. 2, No. 3, pp. 183-192, 1989.

63. Vogl, T. P., J. K. Mangis, A. K. Rigler, W. T. Zink and D. L. Alkon, "Accelerating the convergence of the backpropagation method", *Biological Cybernetics*, Vol. 59, pp. 257-263, 1988.

64. Levenberg, K., "A Method for the Solution of Certain Non-Linear Problems in Least Squares", *The Quarterly of Applied Mathematics 2*, pp. 164-168, 1944.

65. Marquardt, D. W., "An Algorithm for Least-Squares Estimation of Nonlinear Parameters", *Journal of the Society for Industrial and Applied Mathematics*, Vol. 11, No. 2, pp. 431-444, June 1963.

66. Hagan, M. T. and M. B. Menhaj, "Training Feedforward Networks with the Marquardt Algorithm", *IEEE Transactions on Neural Networks*, Vol. 5, No. 6, pp. 989-993, November 1994.

67. Sherrington, C. S., *The Integrative Action of the Nervous System,* Charles Scribner's Sons, 1906.

68. Lawrence, S., A. C. Tsoi and A. D. Back, "Function approximation with neural networks and local methods; bias, variance and smoothness", *Australian Conference on Neural Networks*, Australian National University, pp. 16-21, 1996.

69. Carpenter, G. A., and S. Grossberg, "The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network", *IEEE Computer*, Vol. 21, No. 3, pp. 77-88, March 1988.

70. Conte, S. D., H. E. Dunsmore and V. Y. Shen, *Software engineering metrics and models*, Benjamin-Cummings Publishing, California, 1986.

71. Boetticher, G., T. Menzies and T. Ostrand, PROMISE Repository of empirical software engineering data http://promisedata.org/ repository, West Virginia University, Department of Computer Science, 2007.

72. SDR Dataset for Cost Estimation, Software Research Laboratory, Bogazici University, http://softlab.boun.edu.tr/?q=resources

73. Desharnais, J. M., *Analyse statistique de la productivitie des projets informatique a partie de la technique des point des function*, M.S. Thesis, University of Montreal, 1989.

74. Basili, V. R., F. E. McGarry, R. Pajersky and M. V. Zelkowitz, "Lessons learned from 25 years of process improvement: The Rise and Fall of the NASA Software Engineering Laboratory", *Proceedings of the 24th International Conference on Software Engineering*, pp. 69-79, 2002.

75. Kohavi, R., "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection", *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 1137-1145, 1995.

76. Gosset, W. S., "The probable error of a mean", *Biometrika*, 1908.

77. Kohavi, R. and G. H. John, "Wrappers for feature subset selection", *Artificial Intelligence*, Vol. 97, No. 1-2, pp. 273-324, December 1997.