

SOFTWARE EFFORT ESTIMATION USING
MACHINE LEARNING METHODS

by

Bilge Başkeleş

B.S., Computer Engineering, Istanbul Technical University, 2004

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

2006

ACKNOWLEDGEMENTS

I'd like to thank a bunch of people for helping me in this research. Firstly, I would like to thank my supervisor, Dr. Ayşe Başar Bener. She has been helpful on many circumstances and provided ideas in our discussions. This thesis would not have been finished, without her encouragement, knowledge and help.

In making this thesis, other people have also contributed such as Onur Kutlubay, who helped in clarifying ideas and gathering data. I also thank Evren Ceylan, a former masters student, who encouraged me to complete this thesis and provided me with her recommendations and help.

I'm also grateful to many software companies in Turkey. They have been eager to help, at once. I also thank the students of a cmpe master's course of Dr. Ayşe Başar Bener, they helped in gathering data.

My sincere appreciation goes out to all of my committee members: Prof. Ethem Alpaydın and Prof. Emin Anarım.

I would also like to thank NASA and USC and all the people working in collecting metrics in those institutions. They provided the datasets which constituted a great part of this research.

Finally, I'd like to thank to my family. Without their support, encouragement and love this thesis would not have been possible.

ABSTRACT

SOFTWARE EFFORT ESTIMATION USING MACHINE LEARNING METHODS

In software engineering, the main aim is to develop projects that produce the desired results within limited schedule and budget. There are many factors that affect the budget of a project. The most important factor that is affecting the budget of a project is effort which includes the developers, managers, and architects working on a project. Estimating effort is crucial since hiring more people than actually needed leads to loss of income and likewise hiring less people than actually needed leads to delay in product delivery. To balance schedule and budget, the effort needs to be correctly predetermined. In software engineering this problem is called as effort estimation problem. Software companies usually use expert judgment to estimate the required effort, however, they are far from getting satisfactory results.

The main objective of this research is making an analysis of software effort estimation to overcome problems related to budget and schedule overruns. Our proposed solution not only brings another point of view into software engineering cost and effort estimation but it also tries to improve the software effort estimation process. If practitioners have more accurate estimations then they would be able to manage risks and prevent any losses that may have occurred due to these risks.

We have proposed a model that uses machine learning methods such as Multilayer Peceptrons, Radial Basis Functions, Decision Trees, Support Vector Machines and Principal Components Analysis for a better cost and effort estimation in software development projects. We have obtained the metric data that is used as an input to these methods from third parties such as National Aeronautics and Space Administration (NASA) and University of South California (USC) and various projects from software development organizations in Turkey.

ÖZET

YAZILIM EFORUNUN MAKİNA ÖĞRENME YÖNTEMLERİ KULLANARAK TAHMİNİ

Yazılım mühendisliğinde, asıl amaç limitli zaman planı ve bütçe ile istenilen sonuçları veren projeler üretebilmektir. Bir projenin bütçesini etkileyen bir çok faktör vardır. Bunlardan en önemlisi bir proje üzerinde harcanan efordur ve bu efor proje üzerinde çalışan geliştirici, yönetici ve mimarları da kapsar. Eforu tahmin etmek ise önemlidir çünkü bir projeye fazladan insan atamak gelir kaybına yol açacağı gibi gereğinden az insanı atamak ürünün bitirilmesinde gecikmeye yol açacaktır. Zaman planı ve bütçeyi dengelemek için, efor değerinin önceden belirlenmesi gerekir. Yazılım firmaları genelde gerekli eforu tahmin etmek için uzman yargısını kullanırlar, ancak tatmin edici sonuçlar almaktan uzaktadırlar.

Bu araştırmanın asıl amacı bütçe ve zaman planı aşımalarına bağlı problemlerin üzerinden gelmek için yazılım efor tahmininin bir analizini gerçekleştirmektir. Bizim sunduğumuz çözüm hem yazılım mühendisliğinde efor ve fiyat tahminine farklı bir bakış açısı getirmekte hem de yazılım efor tahmini sürecini geliştirmeye çalışmaktadır. Eğer uygulayıcılar daha doğru efor tahminlerine sahip olurlarsa ilgili riskleri yönetebilirler ve bu risklere bağlı ortaya çıkabilecek zararları önceden önleyebilirler.

Biz, yazılım projelerinde daha iyi fiyat ve efor tahminleri için çok katmanlı perseptron, radyal taban fonksiyonları, karar ağaçları ve destek vektör makinaları gibi makina öğrenme yöntemlerini kullanarak bir model önerdik. Bu modele girdi olarak kullanılacak metric veriyi NASA, USC gibi üçüncü partilerden ve Türkiye’deki farklı yazılım firmalarından topladık.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
ÖZET	v
LIST OF FIGURES	vii
LIST OF TABLES	ix
LIST OF ABBREVIATIONS	xi
1. INTRODUCTION	1
1.1. Motivation	2
1.2. Outline	3
2. BACKGROUND	4
2.1. Parametric Models	4
2.1.1. COCOMO	5
2.1.2. COCOMO II	9
2.2. Learning-oriented Techniques	14
3. METHODOLOGY	16
3.1. Data Collection	17
3.2. Methods Used in the Implementation	18
3.2.1. Artificial Neural Networks	19
3.2.2. Decision Trees	29
3.2.3. Support Vector Machines	31
3.2.4. Principal Components Analysis	34
4. PROBLEM STATEMENT	35
5. OUR PROPOSED SOLUTION	36
5.1. Learning System	37
5.2. Evaluation of Experimental Results	40
6. EXPERIMENTAL RESULTS	42
6.1. Experimental Results without Dimensionality Reduction	43
6.1.1. NASA Dataset	43
6.1.2. USC Dataset	47
6.1.3. SDR Dataset	50

6.2. Experimental Results with Dimensionality Reduction using PCA	54
6.2.1. NASA Dataset.....	54
6.2.2. USC Dataset.....	58
6.2.3. SDR Dataset.....	63
6.3. Method Comparisons.....	67
7. CONCLUSIONS	72
REFERENCES	74

LIST OF FIGURES

Figure 3.1. A standard neural network with three layers	20
Figure 3.2. The structure of a typical three-layer multilayer perceptron	23
Figure 3.3. The backpropagation algorithm.....	24
Figure 3.4. The RBF network	25
Figure 3.5. The algorithm for radial basis functions.....	27
Figure 3.6. k-means algorithm	28
Figure 5.1. The learning system and its components.....	37
Figure 6.1. Scree graph for NASA dataset	55
Figure 6.2. Scree graph for USC dataset	59
Figure 6.3. Scree graph for SDR dataset	63

LIST OF TABLES

Table 2.1. Equations for Effort and Schedule Estimation for COCOMO	6
Table 2.2. COCOMO 81 effort multiplier values	8
Table 2.3. Cost drivers for COCOMO II	11
Table 2.4. COCOMO II Scale factor values	12
Table 6.1. Results for NASA dataset with MLP (%).....	44
Table 6.2. Results for NASA dataset with RBF (%)	45
Table 6.3. Results for NASA dataset with DT (%)	46
Table 6.4. Results for NASA dataset with SVM (%)	46
Table 6.5. Results for USC dataset with MLP (%).....	48
Table 6.6. Results for USC dataset with RBF (%)	49
Table 6.7. Results for USC dataset with DT (%).....	49
Table 6.8. Results for USC dataset with SVM (%)	50
Table 6.9. Results for SDR dataset with MLP (%).....	51
Table 6.10. Results for SDR dataset with RBF (%)	52
Table 6.11. Results for SDR dataset with DT (%).....	53

Table 6.12. Results for SDR dataset with SVM (%)	54
Table 6.13. Results for NASA dataset with MLP (%).....	56
Table 6.14. Results for NASA dataset with RBF (%)	57
Table 6.15. Results for NASA dataset with DT (%)	57
Table 6.16. Results for NASA dataset with SVM (%)	58
Table 6.17. Results for USC dataset with MLP (%).....	60
Table 6.18. Results for USC dataset with RBF (%)	61
Table 6.19. Results for USC dataset with DT (%).....	62
Table 6.20. Results for USC dataset with SVM (%)	62
Table 6.21. Results for SDR dataset with MLP (%).....	64
Table 6.22. Results for SDR dataset with RBF (%)	65
Table 6.23. Results for SDR dataset with DT (%).....	66
Table 6.24. Results for SDR dataset with SVM (%)	67
Table 6.25. Summary of experimental results for NASA dataset (%)	68
Table 6.26. Summary of experimental results for USC dataset (%).....	69
Table 6.27. Summary of experimental results for SDR dataset (%).....	71

LIST OF ABBREVIATIONS

ANN	Artificial Neural Networks
ART	Adaptive Resonance Theory
CMM	Capability Maturity Model
COCOMO	Constructive Cost Model
DSI	Delivered Source Instructions
DT	Decision Tree
EAF	Effort Adjustment Factor
EM	Effort Multiplier
kNN	k-Nearest Neighborhood
LOC	Lines of Code
mdMRE	Median of Magnitude of Relative Error
MLP	Multi Layer Perceptron
MMRE	Mean Magnitude of Relative Error
MRE	Magnitude of Relative Error
MoE	Mixture of Experts
MSE	Mean Squared Error
NASA	National Aeronautics and Space Administration
NOP	Number of Object Points
PCA	Principal Component Analysis
PM	Person Month
PRED	Percentage of relative error within a predefined value
RBF	Radial Basis Functions
SDR	Software Data Repository
SLOC	Source Lines of Code
SMO	Sequential Minimal Optimization
std. dev.	Standard Deviation
SVM	Support Vector Machines
SVR	Support Vector Regression

1. INTRODUCTION

The estimation of software development effort is important and it is carried out for a number of reasons such as budgeting, risk analysis, project planning and investment analysis. Many methods have been developed and applied to explain software development effort as a function of a large number of potentially relevant cost factors [1]. These methods are used on small or medium-sized databases and they come from various fields such as statistics, machine learning and knowledge acquisition.

Software cost estimation which implicitly includes software effort estimation has been the center of attention since the late 1970s [1]. Many software cost estimation models have been proposed such as SLIM, Checkpoint, Price-S, Seer and COCOMO [1]. The developers of these models have all faced the same problem: as software grew in size and importance it also grew in complexity, making it very difficult to accurately predict the cost of software development [1]. Another problem is that the evolving nature of software development has made it very difficult to develop parametric models that can accurately estimate the software development cost or development effort.

Besides the usage of software cost estimation models to estimate software development effort, there are also other approaches such as regression analysis, expertise-based analysis and so forth [2]. However, these methods alone do not lead to very accurate results considering the two problems mentioned above together with the correctness of historical data that is used. There is also another major concern for such methods which questions the accuracy of using company specific data or multi-organization data [2, 3]. The dataset may also need modification according to the size of the companies [4].

In order to use all of the methods mentioned above we need to collect software development related data. How we collect data is important in software projects, since inaccurate measurements may lead to incorrect estimation of project time, effort and cost. Therefore, the collection of data requires expert skills involving the software development process especially in parametric models such as COCOMO and SLIM. In other methods

such as machine learning, the data is also critical because the process is completely based on learning from the historical data at hand.

In this research, we planned to collect software development related data including COCOMO related metrics in companies ranged from small to medium size. The collection of the data is carried out using questionnaires and metric-collection programs. Then, we applied machine learning methods such as back-propagation neural networks, decision trees, radial basis functions and support vector machine methods and we compared the results. These methods are used with historical data as input to learn from previous experiences and then they could be used for estimation.

The major difficulty of estimation with machine learning in this context is the scarcity of data. The companies either do not store the data related to software development or they are not willing to share such data. For this reason, many companies were contacted. The data has been multi-organizational rather than company specific because of the scarcity of data. This also brings the possibility of constructing a general model for effort estimation.

The major contribution of this research is the development of a hybrid effort estimation model that uses model-related data with machine learning methods. We then compared the different machine learning methods as to how they perform when our proposed model is used.

1.1. Motivation

The motivation of this research is estimating the software development effort by using data available or obtained from companies relating different projects and come up with a general model that can explain the software development effort of different projects at different companies.

For estimating effort, we have obtained COCOMO related data for every project in each company. These data are combined to form a single dataset so that learning, in which degree of performance is proportional to the size of data, is more accurate.

Different machine learning methods are used for estimating effort. The analysis of cost dataset containing different projects' data in different companies is carried out by using these methods.

1.2. Outline

Chapter 1 is the introductory part of this thesis. The motivation of the research and the outline is given.

In Chapter 2, the literature for software effort estimation is reviewed with an explanation of the form of the required data for this research.

In Chapter 3, the methodology and the machine learning methods used in the experiments are explained and available comments are stated and discussed.

In Chapter 4, the problem is stated and in Chapter 5, an approach including a learning system is given to how the problem is handled. In Chapter 6, the results of the experiments are stated and compared. Finally, conclusions and future work are given in Chapter 7.

2. BACKGROUND

Reliable and accurate estimates of software development costs are needed throughout the software development cycle to determine feasibility of software projects and to provide for appropriate allocation of resources. In the literature, many methods have been proposed for predicting the cost of software [5]. One of the most popular software estimation methods is estimation by expert knowledge [5]. In this method, the reliability of estimates based on expert judgement depends on the degree in which a new project concurs with the experience and the ability of the expert to remember facts of historical projects [2, 5]. Mostly the estimates are qualitative and not objective. An important problem in using this method is that it is difficult for someone else to reproduce and use the knowledge and experience of an expert [2, 5].

There are many other research done in software cost and effort estimation and the approaches developed up to now can be divided up into six different categories which are parametric models, expertise-based techniques, learning-oriented techniques, dynamics-based models, regression-based models and composite Bayesian techniques for integrating expertise-based and regression-based models [1].

In this section, only the software cost and effort estimation models and techniques that are related to our research is going to be explained which include some parametric models (COCOMO, COCOMO II), learning-oriented techniques (neural networks). Even though some calibrations of COCOMO II is considered as composite techniques, here it is regarded as a parametric technique.

2.1. Parametric Models

In parametric models the estimation of development time and effort is a function of a number of variables [5]. These variables represent the most important cost drivers. The nucleus of an estimation model is a number of algorithms and parameters. The values of the parameters and the selection of algorithms are based on mostly the content of a database of completed projects and previous research. These parameters and algorithms use inputs

such as Source Lines of Code (SLOC), number of functions to perform, and other cost drivers such as language, design methodology, skill-levels, risk assessments, etc.

There are a great deal of parametric models developed up to now. However, only few has been favoured in terms of accuracy, coverage of life-cycle activities and resources [1]. COCOMO, which has been originally published by Barry Boehm in 1981 [6, 7], has been very popular in 1980s. Another very popular parametric model is ESTIMACS which is based on function points analysis and developed in the late 1970s [8]. COCOMO II, which is based on COCOMO, has been the center of attention lately in software estimation because of its accuracy [2].

The parametric models are favored both for their accuracy and simplicity [9]. They are able to generate repeatable estimations. They are easy to modify input data, refine and customize formulas. They are efficient and able to support a family of estimations or a sensitivity analysis. They are objectively calibrated to previous experience. However, there are disadvantages as well as advantages. They are unable to deal with exceptional conditions, such as exceptional personnel in any software cost estimating exercises, exceptional teamwork, and an exceptional match between skill-levels and tasks. Their poor sizing inputs and inaccurate ratings of the cost drivers may result in inaccurate estimation. Additionally, some experience and factors can not be easily quantified [9].

2.1.1. COCOMO

Constructive Cost Model (COCOMO) is originally a cost and schedule estimation model published by Barry Boehm in his book in 1981 [7, 10]. There are three different levels of this model. Basic COCOMO model computes software development effort and cost as a function of program size expressed in estimated Delivered Source Instructions (DSI). The related equation is given in equation 2.1.

$$PM_{nominal} = a * (\text{Thousands of Delivered Source Instructions})^b \quad (2.1)$$

where a and b are two parameters dependent on the application and development environment.

There are three different modes that this basic COCOMO operates on, which correspond to the level of difficulty of the project: organic, semi-detached and embedded [10]. Familiar environment and small team of experienced programmers are the main characteristics of organic mode projects. They usually require less than 50000 DSIs.

Semi-detached mode projects are intermediate projects between organic mode and embedded mode [10]. They involve teams of people with mixed levels of experience. The project has characteristics of both projects for Organic Mode and projects for Embedded Mode and requires no more than 300,000 DSIs.

In the embedded mode, development projects must fit into a rigid set of requirements because the software must work on a system consisting of hardware, software, regulations and operating procedures. Projects have tight constraints, more than one processor and various types of peripherals exist. Uncertainties in the problem definition are another characteristic of such projects [11, 12].

Dependent on these modes, Equation 2.1 changes as stated in Table 2.1 where $PM_{nominal}$ stands for number of months one person would need to develop, $TIME_{dev}$ is the actual number of months needed for development by a properly staffed full-time development team and KDSI is the number of thousands of delivered source instructions i.e. number of source line code given by KLOC [10].

Table 2.1. Equations for Effort and Schedule Estimation for COCOMO

MODE	EFFORT ESTIMATION	SCHEDULE ESTIMATION
Organic Mode	$PM_{nominal} = 3.2 (KDSI)^{1.05}$	$TIME_{dev} = 2.5 (PM)^{0.38}$
Semi-detached Mode	$PM_{nominal} = 3.0 (KDSI)^{1.12}$	$TIME_{dev} = 2.5 (PM)^{0.35}$
Embedded Mode	$PM_{nominal} = 2.8 (KDSI)^{1.20}$	$TIME_{dev} = 2.5 (PM)^{0.32}$

The different modes alone were found not to be the only contributing factor to a project's delivery time and effort. So, intermediate COCOMO is developed as an extension to the basic model which introduces 15 different development productivity factors that fall

under four different categories: product attributes, computer attributes, personnel attributes, project attributes [10].

Product attributes refers to the constraints and requirements placed upon the project to be developed. These included required software reliability (RELY), database size (DATA) and product complexity (CPLX).

Computer attributes refer to the limitations placed upon development effort by the hardware and operating system being used to run the project. The multipliers implying the limitations are execution time constraint (TIME), main storage constraint (STOR), virtual machine volatility (VIRT) and computer turnaround time (TURN).

Personnel attributes refer to the level of skill that is possessed by the personnel. The skills in question are general professional ability, programming ability, experience with the development environment and familiarity with the project's domain. These skills are characterized as analyst capabilities (ACAP), applications experience (AEXP), programmer capabilities (PCAP), virtual machine experience (VEXP) and programming language experience (LEXP).

Project attributes refer to the constraints and conditions under which project development takes place. The issues that affect development are modern programming practices (MODP), use of software tools (TOOL) and required development schedule (SCED).

Each of the factors has associated with it up to five ratings [10]. These ratings are *very low*, *low*, *nominal*, *high*, *very high* and *extra high*. Each rating has a corresponding real number based upon the factor and the degree to which the factor can influence productivity. A rating less than 1 denotes a factor that can decrease the schedule and effort. A rating greater than 1 denotes a factor that extends the schedule or effort. The ratings for every factor is given in Table 2.2.

Table 2.2. COCOMO 81 effort multiplier values [51]

Cost Driver	Description	Very Low	Low	Nominal	High	Very High	Extra High
<i>Product</i>							
RELY	Required software reliability	0.75	0.88	1.00	1.15	1.40	-
DATA	Database size	-	0.94	1.00	1.08	1.16	-
CPLX	Product complexity	0.70	0.85	1.00	1.15	1.30	1.65
<i>Computer</i>							
TIME	Execution time constraint	-	-	1.00	1.11	1.30	1.66
STOR	Main storage constraint	-	-	1.00	1.06	1.21	1.56
VIRT	Virtual machine volatility	-	0.87	1.00	1.15	1.30	-
TURN	Computer turnaround time	-	0.87	1.00	1.07	1.15	-
<i>Personnel</i>							
ACAP	Analyst capability	1.46	1.19	1.00	0.86	0.71	-
AEXP	Applications experience	1.29	1.13	1.00	0.91	0.82	-
PCAP	Programmer capability	1.42	1.17	1.00	0.86	0.70	-
VEXP	Virtual machine experience	1.21	1.10	1.00	0.90	-	-
LEXP	Language experience	1.14	1.07	1.00	0.95	-	-
<i>Project</i>							
MODP	Modern programming practices	1.24	1.10	1.00	0.91	0.82	-
TOOL	Software Tools	1.24	1.10	1.00	0.91	0.83	-
SCED	Development Schedule	1.23	1.08	1.00	1.04	1.10	-

These multipliers are incorporated into the schedule and effort estimation formulas by multiplying them together as in the equation below.

$$PM_{total} = PM_{nominal} * \prod_{i=1}^{15} EM_i \quad (2.2)$$

Incorporating all the characteristics of intermediate COCOMO, advanced COCOMO model includes an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process [9]. However, the most widely used level is the intermediate level, because of its simplicity in defining cost [9]. In this research, intermediate cocomo model is used, too.

2.1.2. COCOMO II

COCOMO II is developed taking into consideration the new approaches in software such as new life-cycle processes and capabilities, usage of fourth generation languages, reuse-driven software approaches and software process maturity initiatives [16]. Its main objectives are as follows: improved accuracy in cost and schedule estimation for the new software practices; customization for different types of projects which is limited to three in the COCOMO model; ease of use for both developers and managers; increased usefulness with tools and tool support capabilities; resource management through an analytic framework; ability to modify that enables evaluating effects of technology improvements.

There are 17 multiplicative cost drivers and 5 scale factors for determining the project's scaling exponent in the equation for estimation. The cost drivers are categorized under four different titles as in COCOMO model. The five scaling factors actually replace the development modes (organic, semi-detached and embedded) in COCOMO model. The 10 of the cost drivers are taken from COCOMO model with different rating scales. The remaining 7 cost drivers are defined from scratch. These cost drivers that are defined under four different categories defined in COCOMO II are given below.

Product attributes refers to the constraints and requirements placed upon the project to be developed. Five cost drivers are included. RELY, DATA and CPLX are already defined in Section 2.1.1. The other two are required reusability (RUSE) which accounts for

the additional effort needed to construct components intended for reuse on the current or future projects and documentation match to life-cycle needs (DOCU) which corresponds to the level of required documentation that is suitable for the project's life cycle needs.

Computer attributes refer to the limitations placed upon development effort by the hardware and operating system being used to run the project. 3 cost drivers are included here. TIME and STOR are already defined in Section 2.1.1. The last one is platform volatility (PVOL) which is the rate of change for the platform of the project (software, hardware etc.). It ranges from low where there is a major change every 12 months, to very high where there is a major change every two weeks.

Personnel attributes refer to the level of skill that is possessed by the personnel. 6 cost drivers are included here. ACAP, AEXP and PCAP are already defined in Section 2.1.1. The remaining skills are platform experience (PEXP) which includes the experience on different and powerful platforms with complex network, database and middleware capabilities, language and tool experience (LTEX) which is the level of experience of the project team on the software tool or language used and it ranges from very low (experience < 2 months) to very high (experience > 6 years). The last multiplier is personnel continuity (PCON) which is the rating scale in terms of project's annual personnel turnover and it ranges from very high (3%) to very low (48%).

Project attributes refer to the constraints and conditions under which project development takes place. Three cost drivers are included here. TOOL and SCED are already defined in Section 2.1.1. The last one is multisite development (SITE) which involves two factors; site collocation (from fully collocated to international distribution) and communication support factors (from surface mail to full interactive multimedia).

The cost drivers are rated from very low to extra high as the COCOMO effort multipliers do. Each rating has a corresponding real number based upon the factor and the degree to which the factor can influence effort. These values for the cost drivers are given in Table 2.3.

Table 2.3. Cost drivers for COCOMO II [51]

Cost Driver	Description	Very Low	Low	Nominal	High	Very High	Extra High
<i>Product</i>							
RELY	Required software reliability	0.82	0.92	1.00	0.85	0.71	-
DATA	Database size	-	0.90	1.00	1.14	1.28	-
CPLX	Product complexity	0.73	0.87	1.00	1.17	1.34	1.74
RUSE	Required reusability	-	0.95	1.00	1.07	1.15	1.24
DOCU	Documentation	0.81	0.91	1.00	1.11	1.23	-
<i>Computer</i>							
TIME	Execution time constraint	-	-	1.00	1.11	1.29	1.63
STOR	Main storage constraint	-	-	1.00	1.05	1.17	1.46
PVOL	Platform volatility	-	0.87	1.00	1.15	1.30	-
<i>Personnel</i>							
ACAP	Analyst capability	1.42	1.19	1.00	0.85	0.71	-
PCAP	Programmer capability	1.34	1.15	1.00	0.88	0.76	-
PCON	Personnel continuity	1.29	1.12	1.00	0.90	0.81	-
AEXP	Applications experience	1.22	1.10	1.00	0.88	0.81	-
PEXP	Platform experience	1.19	1.09	1.00	0.91	0.85	-
LTEX	Language and tool experience	1.20	1.09	1.00	0.91	0.84	-
<i>Project</i>							
TOOL	Software Tools	1.17	1.09	1.00	0.90	0.78	-
SITE	Multisite dev.	1.22	1.09	1.00	0.93	0.86	0.80
SCED	Development Schedule	1.43	1.14	1.00	1.00	1.00	-

The scale factors in COCOMO II help determine the type of the project. There are five scale factors namely precedentedness (PREC), development flexibility (FLEX), architecture risk resolution (RESL), team cohesion (TEAM) and process maturity (PMAT). PREC and FLEX largely capture the differences between the organic, semidetached and embedded modes of the original COCOMO model. If the product is similar to several other products that have been developed before then the precedentedness (PREC) value is high. On the other hand, FLEX captures the amount of constraints the product has to meet. The more flexible the requirements, schedules, interfaces, etc., the higher the rating. RESL captures the thoroughness of definition and freedom from risk of the software architecture used for the product. The team cohesion (TEAM) cost driver accounts for the sources of project turbulence and extra effort due to difficulties in synchronizing the project's stakeholders: users, customers, developers, maintainers, interfacers, others. PMAT is determined according to the level of maturity as a software organization according to Software Engineering Institute's Capability Maturity Model (CMM) [42]. The scale factors are also rated from very low to extra high and numerical values are associated with each factor. These values are given in Table 2.4 below.

Table 2.4. COCOMO II Scale factor values [51]

Scale factors	Very low	Low	Nominal	High	Very High	Extra High
Precedentedness	6.20	4.96	3.72	2.48	1.24	0.00
Development flexibility	5.07	4.05	3.04	2.03	1.01	0.00
Architecture / Risk Resolution	7.07	5.65	4.24	2.83	1.41	0.00
Team Cohesion	5.48	4.38	3.29	2.19	1.10	0.00
Process Maturity	7.80	6.24	4.68	3.12	1.56	0.00

There are three main sub-models in COCOMO II (application composition, early design and post-architecture) based on different sizing models, involving object points, function points and source lines of code [16]. These sub-models constitute the software project process model.

The application composition model is based on object points. Object points are counts of screens, reports and third generation language models developed in the application, each weighted at a three level complexity factor (simple-medium-difficult) [17]. The model involves prototyping efforts (during planning) before life-cycle processes. It is used to resolve high-risk issues such as user interfaces, software-system interaction and performance. The formula adopted here to compute effort is [16]

$$PM_{total} = (NOP * (1 - \%reuse/100)) / PROD \quad (2.3)$$

where PM is the effort in person-months, NOP is the number of object points, %reuse is the percentage of components or modules reused from previous applications, and PROD is the productivity. Here productivity is measured by number of object points developed per month.

The early design model is based on function points. This model is used when requirements are available but design has not yet started. At this stage, the use of function points and small number of cost drivers are involved. The formula here is [16]

$$PM_{total} = a * Size^b * EAF \quad (2.4)$$

where PM is the effort in person months, $EAF = PERS * RCPX * RUSE * PDIF * PREX * FCIL * SCED$, $a = 2.94$ in initial calibration, size in KLOC, b varies from 1.1 to 1.24 depending on novelty of the project, development flexibility, risk management approaches and the process maturity.

The post-architecture model is based on source lines of code. This model is used when the system architecture or software life-cycle architecture has been designed and more information about the system is available. At this stage, source lines of code, modifiers for reuse and software breakage and a total of 17 multiplicative cost drivers and 5 factors for determining the project's scaling exponent are used. These five factors actually replace the development modes (organic, semi-detached and embedded) in COCOMO model [16]. In this model, the same formula is used as in the early design model with a difference of EAF being the multiplication of 17 cost drivers.

The Post-Architecture model equation is :

$$PM_{total} = a * Size^b * EAF \quad (2.5)$$

where a is set to 2.55 and b is calculated as:

$$b = 1.01 + 0.01 \times \sum W_i \quad (2.6)$$

where W are the scale factors.

Since the first version of COCOMO II in 1997, there have been many improvements to the model. The first calibration of the COCOMO model proposed in 1998 uses a weighted linear combination of expert prior vs. sample information [18]. The second calibration to the model is done by incorporating Bayesian analysis [19]. Bayesian analysis is simply a mode of inductive reasoning which enables the usage of both sample and prior (expert-judgment) information. This enables organizations to calibrate the model using their own data to increase model accuracy.

2.2. Learning-oriented Techniques

Learning has been the center of attention since the first announcement of learning algorithms and it has been used in many areas as well as software engineering [1]. There have been two main approaches that is referred to as learning-oriented techniques. One of them is learning by analogy and the other one is neural networks [1].

The analogy cost estimation technique involves comparing one or more completed projects in a similar domain as a means of producing new estimates. This can be achieved by analyzing the data collected from these completed projects against the new project, and by assessing similarities. As the development effort is known this is used as a basis for estimating the current project. Estimation by analogy seems straightforward, but there are a number of problems that need to be addressed. Project features are restricted to information that is available at the start of the estimation process, examples of these can include application domain, software process and so forth. Another problem is the number of projects that should be measured against. Small number of projects can lead to maverick projects being used and too many numbers of projects can weaken the effect of the closest analogies [1].

A previous study compares the use of analogy with prediction models based upon stepwise regression analysis and yields higher accuracies for estimation by analogy [13]. Learning by analogy generally includes a five-step process in which the estimation is developed using project base [14]. The first step in the process is to compile a database of suitable completed projects. This compilation involves projects that reflect both the development environment and project features like in a new project. The second step is to assess the similarities and differences of the sample completed projects with the new project. Similarity is determined in terms of project features. The number of features depend upon the data available to characterize the projects. The third step is to assess the quality of analogy by applying measures for the quality of estimation predication, such as absolute error, percentage error, mean percentage error, magnitude of relative error (MRE) and mean magnitude of relative error (MMRE). The fourth step is consideration of special cases which may lead to ignoring some project features in selected projects that might be unnecessary. Some projects may be excluded to achieve better estimation. The last step is effort estimation of the new projects when the model is fine-tuned.

The usage of neural networks is the alternative to mean least squares regression [1]. These regression methods are based on mean least square error. The simple idea behind this method is using historical data to train the network and produce better estimates by adjusting the parameters of the network. It has been used in software engineering from defect prediction to schedule estimation [15]. An estimation accuracy of within 10% is reported in a related work [37], however the model can have statistical problems with the training data since the standard regression technique is used. The structure of the neural networks is going to be explained in detail in Section 3.1.

In our research, we take advantage of both parametric models and learning oriented techniques. We take projects' data collected using the guidelines asserted by parametric models mentioned in the last section. We then apply machine learning techniques to our proposed model in order to come up with better estimations. The guideline that is used to collect the data is retrieved from COCOMO repository [50].

3. METHODOLOGY

Potts claims that after so many years, of software engineering research practitioners find the results inadequate since the research has been pure scientific [49]. The failure of software engineering research to influence industry is caused by treating research and its application by industry as separate, sequential activities [49]. This fact has led software engineering researchers to redefine the research in this field.

Current trends in software engineering research mostly concentrate on being the glue between scientific research and industrial practice. Such trends have led the researchers to the ‘industry-as-laboratory’ approach [49]. According to this approach, the problems are acquired from the real projects in industry which enables industry and researchers work together to find solutions. This approach removes the gap between research and practice and it enables finding real solutions to real-world problems which has been the main aim of software engineering research in the first place. However, the balance of theory and practice must be adequate because too industrially focused research may lack adequate theory and academically focused research may miss the practice.

There are many topics in software engineering research such as process improvement, software defect prediction etc. In this research, we concentrate on the problem of effort estimation which has been a very important topic in software engineering since 1970’s [1, 2]. Many approaches have been proposed for this particular problem some of which have been explained in Section 2. However, as the software engineering industry evolves, research has to evolve as well.

In our research we collected data from publicly available data repositories and from projects in software development organizations in Turkey. We have applied machine learning techniques for estimating effort. In the following sections, the methods for collecting data and machine learning approaches are explained in more detail.

3.1. Data Collection

Data collection is a very important part of this research since it forms the bond between practice and theory. However, there are many problems involved when collecting data. One of them is the form of the data which must be determined to reflect every aspect in a software project that may affect the total effort value. Another problem is finding data. It is hard since software development companies in Turkey are reluctant to share their project related data. They believe it is a reputation issue that such a metric data is highly confidential to share.

The first problem can be overcome by using a well-known estimation model's form of data to characterize a software project. These models are COCOMO and COCOMO II which are proven to be effective in many estimation problems such as size, effort and schedule in software engineering [7, 9]. COCOMO II is an extended version of COCOMO which is regarded to be more comprehensive due to recent changes in software development.

Data collection problem is overcome by using two different ways for collecting data. One of them is using publicly available data that is collected by other software researchers around the world. Two different datasets USC and NASA have been obtained this way. These data are formed according to COCOMO model. The other way of collecting data is by requesting software project data from the software development organizations in Turkey. This included many challenges in our research such as confidentiality agreements, supervising organizations on how to collect data and data reviews to prevent failures in data collection. For confidentiality reasons, the names of the organizations will not be declared in this thesis. We have used a questionnaire to be filled up and we have prepared a detailed tutorial on how to collect data [50]. The tutorial is prepared in Turkish to prevent misunderstandings. All the companies that participated in data sharing regarded effort estimation as a very important problem. They told us that wrong estimations lead to budget and schedule overruns and project failures, so they wanted to make use of any improvement in that process. This shows the feasibility and applicability of our research.

One of the datasets is obtained from University of South California (USC) [58]. This data is collected by software engineering researchers from credited organizations in software industry and is based on COCOMO model. The other one is obtained from NASA which consists of aeronautical software projects in NASA [14, 48]. The main character of these datasets is that they have been used by many researchers before and they are proven to be trusted to use in any research [14, 48]. USC data consists of data from 63 projects and 17 metrics including effort value in person-month. NASA dataset consists of data from 60 projects and 17 metrics which are same as in the USC dataset.

The dataset collected from software organizations in Turkey, SoftLab Data Repository (SDR), consists of 23 projects ranging from automation projects to online-banking applications. The form of this dataset is based on COCOMO II because it is a more up-to-date and advanced model.

These three datasets are analyzed in their own context by using different machine learning methods with individual parameter calibration.

3.2. Methods Used in the Implementation

The machine learning methods are generally used for classification, learning and estimation problems where the usage of historical data is stressed upon. The real world application areas include not only speech and pattern recognition but also financial predictions and analysis [20]. In theory, machine learning is referred to as a field of study used routinely to discover valuable knowledge from large databases whereas the neural network theory is used for feature generation having m inputs, n outputs and h hidden layers [20]. Machine learning is also programming computers to optimize a performance criterion example data or past experience. This capacity to learn from experience, analytical observation, and other means, results in a system that can continuously self-improve and thereby offer increased efficiency and effectiveness [21].

In this research, we use several machine learning methods as tools to estimate effort using past data gathered by experience using COCOMO guidelines based on COCOMO and COCOMO II model. The methods we use for effort estimation are backpropagation

multilayer perceptrons, decision trees, radial basis function (RBF) and support vector machines (SVM). Principal components analysis (PCA) has also been used as a method to reduce dimensionality. These methods are going to be explained in the following subsections.

3.2.1. Artificial Neural Networks

An artificial neural network (ANN) is an interconnected group of artificial neurons that uses a mathematical or computational model for information processing based on a connectionist approach to computation [22]. The original inspiration for the technique was from examination of the central nervous system and the neurons (and their axons, dendrites and synapses) which constitute one of its most significant information processing elements which have their roots from neuroscience [22]. These networks do not approach a full appreciation of neuronal function, however the structure of the networks in which each neuron takes input signals, processes them according to an internal weighting system and produces an output that is propagated to other neurons, resembles the general structure of biological networks [22].

The neural networks consists of input layer, a number of hidden layers and an output layer each of which consists a number of units (neurons) that are connected to one another. Basically, a neuron receives inputs from the related sources, combines these inputs according to the learning algorithms, performs a generally nonlinear operation on that result, and then outputs the final result. Typically, units in neighboring layers can have connections with one another and there is no connection between the units in the same layer. When there are cyclic connections of neurons, the network becomes a recurrent neural network [22]. A typical neural network structure is a three-layer network, in which there are three layers, namely input layer, one hidden layer and an output layer. An example is shown below (Figure 3.1).

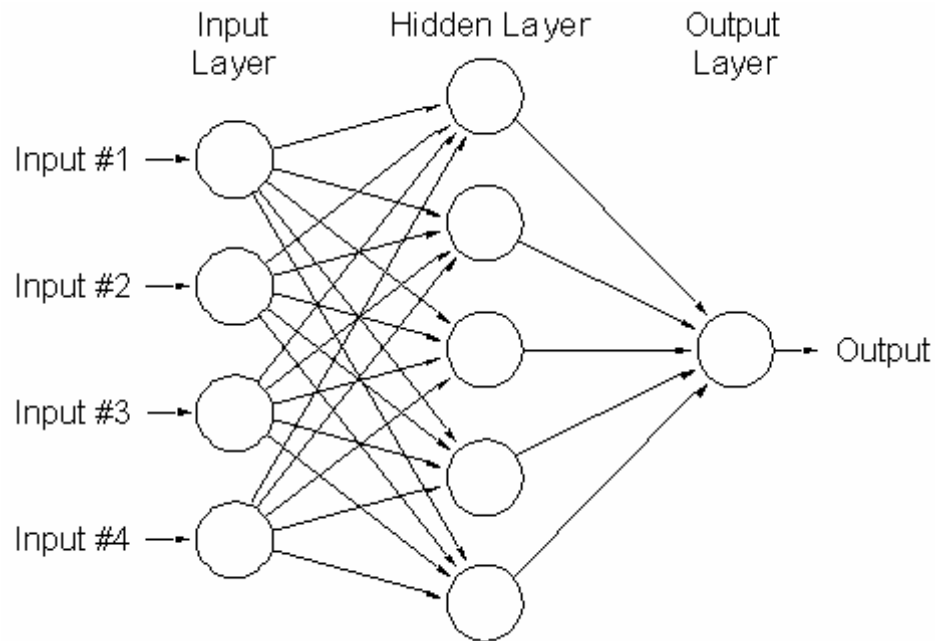


Figure 3.1. A standard neural network with three layers [22]

There are two main constraints to be determined in the usage of neural networks: the model of the network, the algorithm to be used for training and the cost function. The model of the network depends on the data representation and the application. The choice of the model is important in that complex models may lead to problems in learning.

The training algorithms generally will work well with the correct hyper parameters for training on a particular fixed dataset [22]. However selecting and tuning an algorithm for training on unseen data requires a significant amount of experimentation. To overcome this in our work, several artificial neural network methods with various parameters are applied on the collected data that is collected.

Cost functions must be used in neural network for the functions to produce optimal outputs and thus learn the model better. It is a measure of how far away we are from an optimal solution to the problem that we want to solve. Learning algorithms search through the solution space in order to find a function that has the smallest possible cost. Cost function is generally taken as mean-squared error. If the model, cost function and learning algorithm are selected appropriately the resulting ANN can work in an efficient way [22].

The simple mechanism of the neural network is as follows: the input layer takes the input data, the units in the input layer compute the output and propagate to the hidden layers. The hidden layers compute outputs and propagate to the output layer. The learning takes place according to this mechanism [38]. There are three types of learning: supervised learning, unsupervised learning and reinforcement learning [22].

In supervised learning, the aim is to learn a mapping from the input to an output whose correct values are provided by a supervisor. In this case, we are given both the input and the output. The network is processed so that an output is produced for every input and compared with the desired output through a cost function. The result of this function, the error, is fed back to the network and the weights of the network are adjusted according to this error. This process is repeated until there is an acceptable change or no change in the total error. Tasks that fall within the paradigm of supervised learning are pattern recognition (also known as classification) and regression [20, 21].

In unsupervised learning, there is no supervisor as in supervised learning and we only have input data without any outputs. The aim is to find regularities in the input, i.e. a structure in the input data is tried to be found with certain patterns. This is often done by clustering the data that is the data is grouped into clusters. The cost function here is dependent on the task to be carried out, not on the output. Tasks that fall within the paradigm of unsupervised learning are in general estimation problems; the applications include clustering, the estimation of statistical distributions, compression and filtering [20, 21].

In reinforcement learning, neither the input nor the output is given. The input is generated by an agent's interactions with the environment. At each point in time t , the agent performs an action and the environment generates an observation and an instantaneous cost, according to some (usually unknown) dynamics. The aim is to discover a policy for selecting actions that minimizes some measure of a long-term cost, i.e. the expected cumulative cost. The environment's dynamics and the long-term cost for each policy are usually unknown, but can be estimated. Tasks that fall within the paradigm of reinforcement learning are control problems, games and other sequential decision making tasks [22, 23].

There are many types of ANNs such as multilayer perceptrons (MLP), radial basis functions (RBF), mixture of experts (MOE), recurrent networks [22]. In this research we are using MLP and RBF which are supervised networks which is appropriate for estimation.

The implementation of ANNs in fast and parallel hardware architectures and their ability of dealing with complex and large datasets make them the ultimate solution in many tasks. The tasks to which ANNs are applied tend to fall within the following broad categories [22]: Function approximation, or regression analysis, including time series prediction and modeling, classification, including pattern and sequence recognition, novelty detection and sequential decision making and data processing, including filtering, clustering, blind source separation and compression.

3.2.1.1. Multilayer Perceptrons. Multilayer perceptrons (MLPs) are feed-forward neural networks trained with the standard back-propagation algorithm [22]. They are supervised networks so they are typically used in *supervised learning* problems and they require a desired response to be trained. They learn how to transform input data into a desired response, so they are widely used for pattern classification [22]. With one or two hidden layers, they can approximate virtually any input-output map. With no hidden layer, the multilayer perceptron can only perform linear tasks. It has been proved that the MLP with only one hidden layer has the ability to approximate any continuous function [24]. Below is the structure of a multilayer perceptron in Figure 3.2.

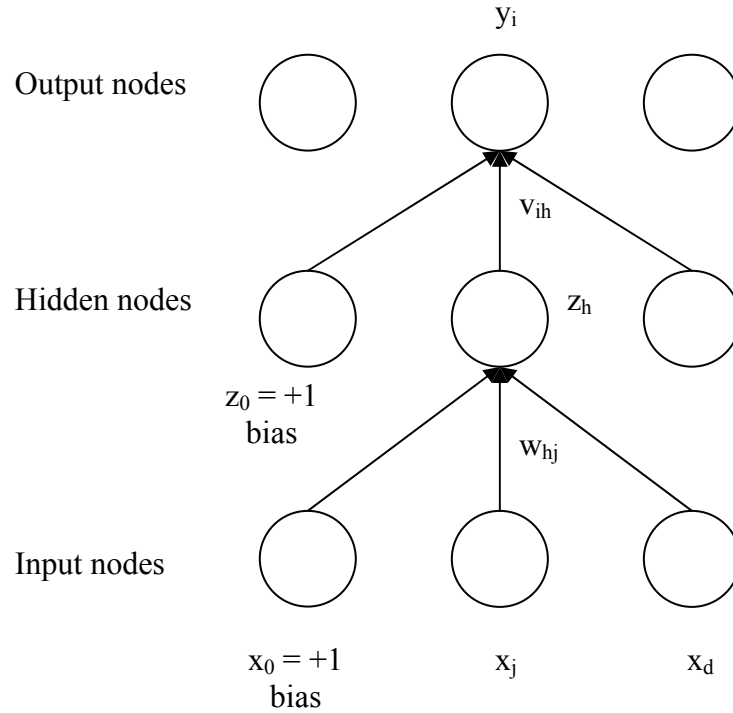


Figure 3.2. The structure of a typical three-layer multilayer perceptron [21]

In Figure 3.2, x_j , $j=0,\dots,d$ are the inputs, z_h , $h=1,\dots,H$ are the hidden units where H is the number of hidden units except bias. z_0 is the bias of the hidden layer and x_0 is the optional bias in the input layer. y_i , $i=1,\dots,K$ are the output units. w_{hj} are the weights in the first layer, and v_{ih} are the weights in the second layer.

The supervised learning problem of the MLP can be solved with the *back-propagation algorithm*. The algorithm consists of two steps. In the *forward pass*, the predicted outputs corresponding to the given inputs are evaluated [25]. In the *backward pass*, partial derivatives of the cost function (error) with respect to the different parameters are propagated back through the network [25]. The network weights can then be adapted using any gradient-based optimization algorithm. The whole process is iterated until the weights w and v have converged and the error is minimized [25, 26]. This whole process is regarded as training where the network is trained so that the network model approaches to producing the desired output.

Each hidden unit in the multilayer perceptron we used in our experiments applies the nonlinear sigmoid function to its weighted input summation [22]. The nonlinear sigmoid functions transform the discriminant value to a posterior probability.

$$\text{Sigmoid}(x) = 1 / (1 + \exp(-x)) \quad (3.1)$$

Mean square error is used as the cost function. The networks weights are adapted using a learning rate (η) at 0.01. The algorithm implemented for our experiments is below:

Initialize weights randomly so that $-0,01 < (W_{hj}, V_{ih}) < 0,01$

Repeat for all (x^t, r^t) in random order

For $h = 1, \dots, H$

$$Z_h = \text{sigmoid}(W_h^T + W_{h0}) = 1 / (1 + \exp(-W_h^T X + W_{h0}))$$

For $i=1, \dots, K$

$$y_i = v_i^T z$$

For $i = 1, \dots, K$

$$\Delta v_i = \eta (r_i^t - y_i^t) z$$

For $h=1, \dots, H$

$$\Delta w_h = \eta (\sum (r_i^t - y_i^t) v_{ih}) z_h (1 - z_h) x^t$$

For $i=1, \dots, K$

$$v_i = v_i + \Delta v_i$$

For $h=1, \dots, H$

$$w_h = w_h + \Delta w_h$$

Find the training error

$$\text{Error} = 0.5 \cdot \sum_1^h (r_i - y_i)^2$$

Until error does not change or changes at an acceptable range

Figure 3.3. The backpropagation algorithm

Multilayer perceptrons are known as the mostly used neural networks trained with backpropagation algorithm [24]. We also used the typical MLP with backpropagation algorithm in our approach, to obtain results that are comparable to parametric models discussed in Section 2.

3.2.1.2. Radial Basis Functions. Radial basis functions (RBFs) provides local representation for given input by activating few units unlike multilayer perceptrons where a distributed representation of the input is given by simultaneous activation of many hidden units [28]. However, the structure of RBFs are similar to that of MLPs. They have input and output nodes like an MLP. They typically have two distinct layers. The bottom layer consists of units which use Gaussian functions as the basis functions. The top layer is a simple linear discriminant that outputs a weighted sum of the basis functions. The Gaussian function responds only to a small region of the input space where the Gaussian is centered. Within basis functions in the network there is a radial symmetry with respect to a center. [28, 29]. A representation of RBF is below in Figure 3.4.

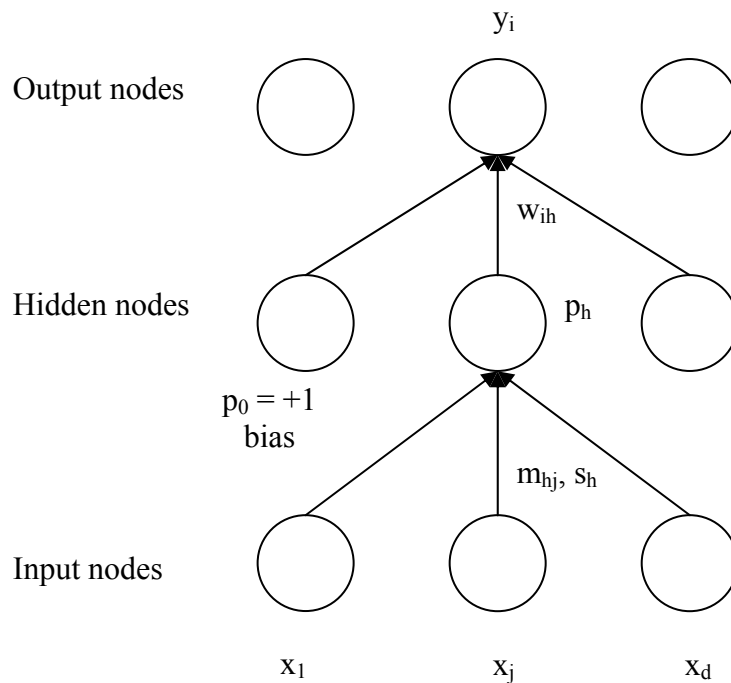


Figure 3.4. The RBF network [21]

In Figure 3.4, p_h are the hidden units using the basis function. m_h (local center of unit h), s_h (spread of the local unit h) are the first layer parameters, and w_i are the second layer weights.

For each unit, there is a local center and the centers are input to the basis functions. The concept of locality requires a distance function to measure the similarity between the given input x and the position of unit h , m_h . This distance is measured frequently as Euclidean distance [21]. The response function is chosen to be the Gaussian, as denoted before. It has a maximum where $x = m_h$ and decreases as they get less similar. The function is in equality 3.2. The idea in using such local basis functions is that in the input data, there are groups or clusters of instances and for each cluster, a basis function p_h^t is defined which becomes nonzero if instance x^t belongs to cluster h .

$$p_h^t = \exp \left[- \frac{\|x^t - m_h\|^2}{2s_h^2} \right] \quad (3.2)$$

The key to a successful implementation of these networks is to find suitable centers m_h for the Gaussian functions [28]. There are many methods to finding centers such as k-means clustering and adaptive resonance theory (ART). K-means clustering is the most generally used method [27]. In the following subsection, k-means is explained in more detail. The RBF algorithm that is used in this research is shown below (Figure 3.5).

Find m_h values using k-means clustering in order to start with better centers,
find s_h values heuristically.

Repeat

For all (x^t, r^t) in random order

For $h = 1, \dots, H$

$$p_h^t = \exp (||x^t - m_h||^2 / 2s_h^2)$$

For $i = 1, \dots, K$

$$y_i^t = \sum_1^H w_{ih} p_h^t + w_{i0}$$

For $i = 1, \dots, K$

$$\Delta w_{ih} = \eta \sum_1^H (r_i^t - y_i^t) p_h^t$$

$$\Delta w_{i0} = \eta \sum_1^H (r_i^t - y_i^t) p_0^t$$

For $j = 1, \dots, K$

$$\Delta m_{hj} = \eta \sum_1^N \left(\sum_1^K (r_i^t - y_i^t) w_{ih} \right) p_h^t ((x^t - m_{hj}) / s_h^2)$$

$$\Delta s_h = \eta \sum_1^N \left(\sum_1^K (r_i^t - y_i^t) w_{ih} \right) p_h^t (||x^t - m_{hj}||) / s_h^2$$

$$w_h = w_h + \Delta w_h$$

$$m_h = m_h + \Delta m_h$$

$$s_h = s_h + \Delta s_h$$

Find the training error (MSE is calculated according to the formula)

$$Error = 0.5 \cdot \sum_1^h (r_i - y_i)^2$$

Until Convergence (error is not changing or changing in a negligible range)

Figure 3.5. The algorithm for radial basis functions

The advantage of the radial basis function network is that it finds the input to output map using local approximators [21]. Usually the supervised segment is simply a linear combination of the approximators which vary smoothly over the target space and change centers and features where applicable. This leads to a more accurate fitting to the target function thus leading to small extrapolation errors. Also these networks train extremely fast and require fewer training samples, since linear combiners have few weights. It is known that RBFs outperform MLPs in function estimations by both performance and speed [23]. This follows from the fact that in MLPs the inputs to the activation functions in the first hidden layer are linear combinations of input space so the outputs are the same and in RBFs the output of each RBF node, is only the same for all points having the same *Euclidean distance* from the respective center (cluster centers); and decreases

exponentially with the distance and the input layer and the output layer of an RBF are trained separately [20]. In our research, besides MLPs we also used RBF with k-means clustering since we have a problem of the scarcity of data.

3.2.1.3. K-means. K-means is an error based clustering method in which data is partitioned into k , a predefined positive integer value, partitions and groups each component in data into a cluster so the minimum average distance between clusters is obtained [27]. The sum of squared Euclidean distances from the mean of each cluster is used as a measure between cluster centers and individual points. The algorithm first takes k points in the dataset randomly and assigns them as cluster centers. Then every component in the data is examined and assigned into a cluster according to the minimum distance with the cluster centers. Cluster centers are recalculated according to the distance between cluster centers and the component, after every component is assigned a cluster. This process is repeated until all cluster centers are assigned a value so that they are mutually farthest apart. The simple k-means algorithm is below in Figure 3.6.

```

Initialize  $m_i, i=1, \dots, k$  to  $k$  random  $x^t$ 
Repeat
  For all  $x^t$  in random order
     $i = \arg \min_j ||x^t - m_j||$ 
     $m_i = m_i + \eta (x^t - m_i)$ 
  Until  $m_i$  converge

```

Figure 3.6. k-means algorithm [21]

An advantage of k-means clustering approach is that it is an error based clustering method, i.e. it incorporates errors associated with data into the clustering algorithm. Standard clustering methods ignore measurement errors, or uncertainty, associated with the data. However, errors play a significant role in improving the clustering decision [30]. Another advantage of the algorithm is its simplicity and its speed which allows it to run on large datasets. The main weakness of the algorithm is that the resulting cluster centers may change because of the random initialization of them at the beginning [39].

This property maximizes the inter-cluster variance and minimizes the intra-cluster variance.

3.2.2. Decision Trees

A decision tree is a hierarchical model for supervised learning which is a popular method used mostly for estimation purposes [32]. A decision tree is composed of internal decision nodes and terminal leaves. The internal nodes are tests and the leaf nodes are the categories. Given an input, at each node, a test is applied and one of the branches is taken depending on the outcome. This test process starts at the root and is repeated recursively until a leaf node is hit, in which case the value at the leaf constitutes the output.

There are two main types of decision trees [32]. One of them is classification tree used for estimating categorical variables, i.e. class labels for each input. Other type is the regression tree which is used to approximate real-valued continuous functions. In our research, we use regression trees to approximate the desired output.

For classification problems, decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. The leaf nodes are categorical values and the inputs are either categorical or numerical. [20]. Classification trees are the trees used for classification and are widely used in many areas such as stock market forecasting, categorical predictions and so on.

For regression problems, the output variable is numerical, whereas the inputs are a mixture of continuous and categorical variables. A decision tree is generated where each decision node in the tree contains a test on some input variable's value. The terminal nodes of the tree contain the predicted output variable values instead of categories. The decision trees used for this task are called regression trees that may be considered as a variant of decision trees, designed to approximate real-valued functions instead of being used for classification tasks [31]. These trees can be used for single-output approximation and prediction problems.

Decision trees are attractive because they show clearly how to reach a decision, and because they are easy to construct automatically from labeled instances. The main advantage of decision trees is that they are robust and they perform well with large data in a short time. Also, in this method there is no need for data preparation such as normalization, filling blank values and so on [20, 32].

Decision trees progressively split the set of training examples into smaller and smaller subsets. It would be ideal if all the samples in each subset had the same category label. In that case, each subset would be pure, and that portion of the tree can be terminated with a leaf. Usually, however, there is a mixture of labels in each subset, and thus for each branch we will have to decide either to stop splitting and accept an imperfect decision, or instead select another property and grow the tree further [33]. This suggests an obvious recursive tree-growing process in a top-down manner: given the data represented at a node, either declare that node to be a leaf (and state what category to assign to it), or find another property to use to split the data into subsets [33]. For these purposes, tree construction algorithms have been proposed such as ID3, CART and C4.5 [40, 41]. In this research, C4.5 is used as the decision tree algorithm. This algorithm improves the computational efficiency, avoids data over-fitting problems and reduces error pruning. It is fast and efficient in inducing trees from training data [34].

The approach to constructing decision trees usually involves using greedy heuristics that overfit the training data and lead to poor accuracy in future predictions [35]. In response to the problem of overfitting nearly all modern decision tree algorithms adopt a pruning strategy of some sort [35]. Many algorithms use a technique known as postpruning or backward pruning. This essentially involves growing the tree from a dataset until all possible leaf nodes have been reached and then removing particular subtrees. Studies have shown that post-pruning will result in smaller and more accurate trees by up to 25% [35]. Another technique is pre-pruning where tree construction is stopped early before it is full. However, pre-pruning is not a preferred approach since precise estimation of when to stop growing the tree usually can not be made [36]. So, the post-pruning is used mostly and it leads to more accurate results. Also, variance in the data is minimized by selecting a simpler tree and the combination of two attributes may lead to better estimations than when they are individually taken as two different attributes at different nodes [36].

3.2.3. Support Vector Machines

Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression. Their common factor is the use of a technique known as the "kernel trick" to apply linear classification techniques to non-linear classification problems [43].

In classification problem, classifying data as part of a machine-learning process is interesting. When the data points in the set are multidimensional, the classification can be carried out with separating the points by using hyperplanes [43]. This form of classification is linear and the classification is required to be neat with maximum distance to the closest data point from both classes where the distance is the margin [43]. If the hyperplane has this margin property, it is called maximum-margin hyperplane, as are the vectors that are closest to this hyperplane, which are called the *support vectors*.

The formalization of the algorithm is as follows. Data points are first considered to be of the form $\{ (x_1, c_1), (x_2, c_2), \dots, (x_n, c_n) \}$ where the c_i is either 1 or -1 and denotes the class to which the point x_i belongs to in a two-class classification problem. This is viewed as training data and the dividing hyperplane takes the form below:

$$w * x - b = 0 \quad (3.3)$$

However, the maximum margin is of interest where the support vectors and the parallel hyperplanes (to the optimal hyperplane) are closest to these support vectors in either class. These parallel hyperplanes can be described by equations below:

$$w * x - b = 1 \quad (3.4)$$

$$w * x - b = -1 \quad (3.5)$$

These hyperplanes must maximize the distance from the dividing hyperplane so that there is no data points between them. By using geometry, the distance between the hyperplanes is $2/|w|$, so $|w|$ must be minimized. To exclude data points, for all i the inequalities below must be assured:

$$w^* x_i - b \geq 1 \quad (3.6)$$

$$w^* x_i - b \leq -1 \quad (3.7)$$

$$c_i(w^* x_i - b) \geq 1, \quad 1 \leq i \leq n. \quad (3.8)$$

The problem reduces to minimizing $|w|$ which is a quadratic programming (QP) optimization problem [43]. After training, test data can be classified by using the rules below :

$$C = \begin{cases} 1, & \text{if } w^* x - b \geq 0 \\ -1, & \text{if } w^* x - b \leq 0 \end{cases} \quad (3.9)$$

The use of the maximum-margin hyperplane is motivated by Vapnik Chervonenkis theory, which provides a probabilistic test error bound that is minimized when the margin is maximized [43]. However the utility of this theoretical analysis is sometimes questioned given the large slack associated with these bounds: the bounds often predict more than 100% error rates [43].

The parameters of the maximum-margin hyperplane are derived by solving the optimization. There exist several specialized algorithms for quickly solving the quadratic programming (QP) problem that arises from SVMs. The most common method for solving the QP problem is Platt's sequential minimal optimization SMO algorithm [44]. Sequential Minimal Optimization (or SMO) is a fast method to train the support vector machines. SMO breaks the large QP problem into a series of smallest possible QP problems. These small QP problems are solved analytically, which avoids using a time-consuming numerical QP optimization as an inner loop. The amount of memory required for SMO is linear in the training set size, which allows SMO to handle very large training sets.

The original optimal hyperplane algorithm proposed by Vladimir Vapnik in 1963 was a linear classifier. However, in 1992, Bernhard Boser, Isabelle Guyon and Vapnik suggested a way to create non-linear classifiers by applying the kernel trick (originally proposed by Aizerman) to maximum-margin hyperplanes [44]. The resulting algorithm is formally similar to the previous, except that every dot product is replaced by a non-linear kernel function. This allows the algorithm to fit the maximum-margin hyperplane in the

transformed feature space. The transformation may be non-linear and the transformed space high dimensional; thus though the classifier is a hyperplane in the high-dimensional feature space it may be non-linear in the original input space. Some possible kernel functions are denoted below:

1. Polynomial homogeneous

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}')^d \quad (3.10)$$

1. Polynomial inhomogeneous

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + 1)^d \quad (3.11)$$

2. Gaussian RBF

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right) \quad (3.12)$$

3. Sigmoid

$$k(\mathbf{x}, \mathbf{x}') = \tanh(\kappa \mathbf{x} \cdot \mathbf{x}' + c) \quad (3.13)$$

A version of a SVM for regression was proposed in 1997 by Vapnik, Steven Golowich, and Alex Smola [45]. This method is called support vector regression (SVR). The model produced by support vector classification (as described above) only depends on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin. Analogously, the model produced by SVR only depends on a subset of the training data, because the cost function for building the model ignores any training data that is close (within a threshold ε) to the model prediction.

Support vector regression is used here as a means to estimate effort. It has been reported that SVMs can be effectively trained even if the number of examples is much lower than the dimensionality of the object space [54]. Linear kernel function is used within because of its simplicity, accuracy and fastness which has been stated by previous researchers [55].

3.2.4. Principal Components Analysis

Principal components analysis (PCA) is an unsupervised technique that can be used to simplify a dataset by reducing the number of dimensions without using the output values associated to every input [47]. It is a linear transformation that chooses a new coordinate system for the data set such that the greatest variance by any projection of the data set comes to lie on the first axis, called the first principal component, the second greatest variance on the second axis, and so on. This procedure is carried out by eigenvector and eigenvalue computations. So, a set of correlated variables are transformed into a set of uncorrelated variables which are ordered by reducing variability. The uncorrelated variables are linear combinations of the original variables, and the last of these variables can be removed with minimum loss of real data [47].

There are several algorithms for calculating the Principal Components [20, 21]. One of them is the covariance method where eigenvectors of the covariance matrix of the data are used to construct the new dataset. When the variances of the original dimension of the data vary considerably, correlation algorithm can be used where correlation matrix is used for the correlations to be effective and not the individual variances [20, 21].

PCA can be used for reducing dimensionality in a dataset while retaining those characteristics of the dataset that contribute most to its variance, by keeping lower-order principal components and ignoring higher-order ones. The idea is that such low-order components often contain the most important aspects of the data [46]. The number of dimensions can also be predicted by using some techniques such as the analysis of scree graph. This graph is the plot of variance explained as a function of the number of eigenvectors kept. If adding an eigenvector does not increase the variance, then it means that keeping the first ones is enough to explain the data [21].

PCA is usually used in problems where the dataset is large so that computations will be less complicated. In our case, PCA is used even though the datasets are not big in size. This is done, because in regression the reduction of the attributes might lead to a better estimation since there might be misleading attributes.

4. PROBLEM STATEMENT

Software quality is driven by many aspects in software development. In software development organizations, the main aim is to start a project and finish it within acceptable schedule and budget. The main drivers affecting the budget are the number of employees and time. These two come together to form a measure called effort [1,2]. Effort is the most important factor in a software project determining the cost. Estimating effort is a very important factor affecting the quality of software. At early stages of software development, effort must be estimated to come up with a planned schedule and budget. This may in turn be used to determine the cost and the price of the project. However, estimations of effort lower than the actual value may lead to budget and schedule overruns if the organization insists on quality. This usually leads to a damage in the reputation of the organization resulting in less number of new projects and lower revenues. Effort estimation, lower than the actual, does not bring any benefits either for the organizations or for the customers. On the other hand, when the estimation of effort is higher than the actual, the software development company would allocate resources (time and people) more than actually it is needed, resulting in higher costs and lower margins.

Since 1970s researchers have proposed many models on software effort estimation. These models have been mostly parametric to estimate effort [1,2]. They largely make use of historical data from projects. Inputs of these models include many aspects of a software project such as personnel, environment, language etc. However, every one of them produced limited accuracy and there have been also many calibrations to the methods [18]. The main problem here is that these models are not updated periodically. Software processes constantly evolve as new and different technologies and applications are developed and used. Especially, in current software industry, where changes are arbitrary, an evolving system is required for the problem of effort estimation.

Estimating effort at the early phases is a main factor affecting the quality of software taking into account the risks such as extended schedules, budget overflows etc. in a software project.

5. OUR PROPOSED SOLUTION

In this research, the main aim is to improve software effort estimation by constructing a learning system that makes use of several machine learning methods on software effort estimation datasets. The usage of machine learning on software engineering has been tried before on problems such as defect prediction [15]. The main reason for using such a learning system for this problem is to keep the estimation process up-to-date by incorporating up-to-date project data. If the effort estimation process evolves with new projects, the estimation model is kept up-to-date which overcomes the main problem in parametric estimation models. The datasets are collected based on the guidelines of two parametric cost estimation models, namely COCOMO and COCOMO II. These guidelines are either prepared by us or retrieved from official site. There are two COCOMO datasets one of which is collected at NASA and the other one is the dataset collected by USC. There are 60 different projects' data in the NASA dataset and 63 different projects' data in USC dataset. SDR dataset is collected from several software organizations in Turkey, is based on COCOMO II and includes 23 projects' data.

Although the datasets seem small considering the rule of learning which states that for every attribute there must be 5-10 instances, the sizes are satisfactory considering the scarcity of data in effort estimation process around the world. For example, a five-year-old organization can have at most 30 medium-sized projects and it can still carry out estimation processes since the number of projects whose effort is to be estimated is very limited in the sense that not more than three or four projects are started at a time. This fact enables us to keep the training set large while keeping the test set small.

The learning system here evolves with new projects to come and create better estimations for effort while depending on the efficiency of metrics formation invented earlier such as COCOMO and COCOMO II. The most important property of the models that our metric datasets are based on is that they are mostly used and dependable to explain different attributes of every individual project. Thus, the datasets are reliable and can be used for research purposes. The reliability is very important since usage of datasets is crucial in our research.

The other important part is the learning system where different methods are used to test the performance of every one of them and come up with a model that gives the best prediction for every dataset. There are also other methods incorporated for calibration of the dataset which are explained in the last section.

5.1. Learning System

In the learning system, there are many steps that are carried out. Firstly, the datasets are individually separated into training and test sets. The selection of the number of data points in the training set and the test set can be varied according to the information given above. The experiments are carried out with training and test sets of different sizes. Secondly, normalization and PCA is applied as a means to reduce dimensionality. This is optional and a dimensionality test is also applied to determine the number of dimensions to reduce the dimensionality. Thirdly, one of the methods explained in section 3 is used to train the system such as MLP, RBF, DT and SVM. It must be noted that k-means algorithm is used to create centers for the RBF algorithm, though it can not be seen in the system in Figure 5.1. Finally the performance of the system is evaluated by using the test set and comparing results.

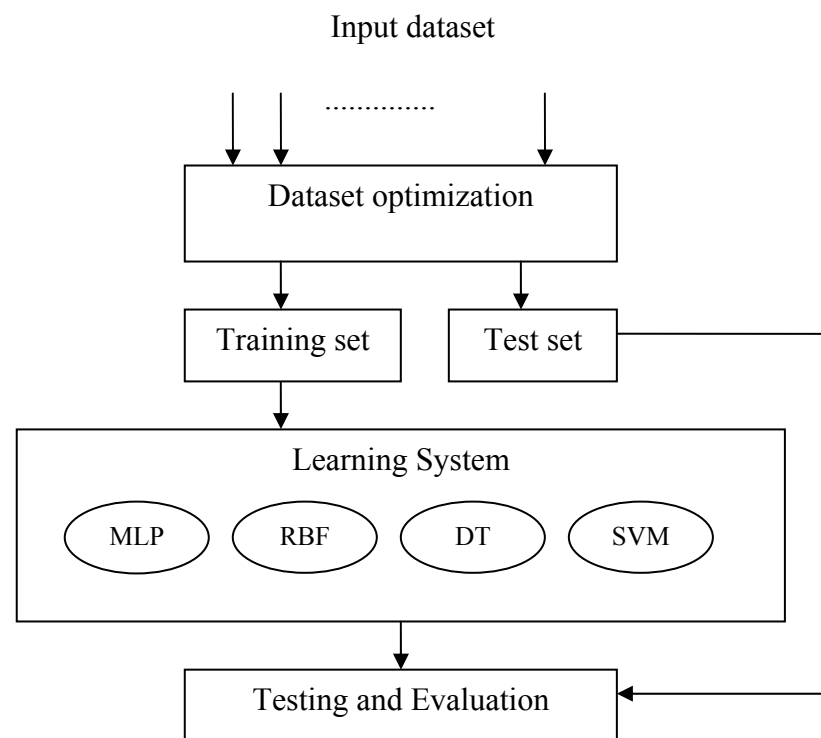


Figure 5.1. The learning system and its components

In the first step of the learning system, the data is given as input. The data consists of the effort estimation data based on two different universally accepted effort estimation models - COCOMO and COCOMO II.

For COCOMO-based datasets, there are sixteen input values and an output value which is the effort value in person-month. The fifteen input values are as primarily described in Section 2.1.1 which are product attributes – RELY, DATA, CPLX, computer attributes – TIME, STOR, VIRT, TURN, personnel attributes – ACAP, AEXP, PCAP, VEXP, LEXP, project attributes – MODP, TOOL, LEXP. The last input value is the size of the project as source lines of code in kloc (kilolines of code).

For COCOMO II-based datasets there are twenty-three input values and an output value which is again the effort value in person-month. The seventeen of the input values are the effort multipliers described in Section 2.1.2 which are product attributes - RELY, DATA, CPLX, RUSE, DOCU, computer attributes - TIME, STOR, PVOL , personnel attributes - ACAP, AEXP, PCAP, PEXP, LTEX, PCON, project attributes – TOOL, SCED, SITE. The five of the input values are scale factors - PREC, FLEX, RESL, TEAM, PMAT. The last input value is the size of the project as source lines of code in kloc (kilolines of code).

In the second step, data optimization, the dataset goes under a number of operations which are normalization, principal components analysis for an efficient computation in the learning system. The dataset is firstly normalized for keeping the dataset in a predefined range. Principal components analysis (PCA) is used optionally to see the effects of dimensionality reduction on the performance of the prediction of the system. Before applying PCA, a test is run to determine the number of dimensions that the dataset will be reduced to. This test includes the analysis of the scree graph which is the analysis of eigenvalues and eigenvectors computed throughout PCA. PCA is used also for eliminating the negative effects of correlations within the dataset. However, it must be noted that PCA is not expected to improve the performance or decrease it, since the effect depends entirely on the dataset and the method used within.

In the third step, the training set is input to the learning system to train it. The test set is kept aside for performance evaluation. It is intuitive that using the same set for training and test set would result in very optimistic outcomes for a problem like effort estimation where the accuracy is important. Thus we use different parts of the same dataset. The percentage of the number of inputs in the training and test sets are determined earlier, before the application of PCA. This percentage is determined according to the size of the datasets. Although the rule of training and test set production for machine learning applications is obeyed in some experiments, there are also cases when the test set is kept small to improve the learning scale of the system. We think that this is an acceptable case because the number of projects to be estimated are usually very limited in the software engineering domain. Because the software projects are usually started as one at a time and estimation process is encountered according to the frequency of starting projects. These two different aspects of training and test sets' size determination will enable us to evaluate the performance of the system by number of projects to be estimated.

In the fifth step, the learning system is trained using the training set determined in the last steps. The methods used in the learning system are multilayer perceptrons (MLP), radial basis functions (RBF), decision trees (DT) and support vector machines (SVM). These methods have been described in detail in Chapter 3. In each experiment, a different method is used and the results are stated for each of the methods. Each of the method is executed using the cross validation technique to compare learners. The training and test sets are kept the same for a healthy comparison of the performance of each method with another.

In the system, the best performing approach (a combination of methods) for every dataset can be found by the evaluation process. By finding such an approach, organizations can come up with their own approach and they can do effort estimation on the dataset of their own projects. Such an approach is handy where a specialization per organization can improve the accuracy of estimation, thus reducing the risks related to a software project within the organization.

5.2. Evaluation of Experimental Results

The performance of effort estimation datasets and models can be evaluated using a number of different approaches such as PRED(L), MMRE (mean magnitude of relative error) and correlation which are measures that are widely known and almost always used in research on effort estimation for comparison purposes [2]. PRED, mdMRE and MMRE are the de facto standard evaluation criteria to assess the accuracy of software project prediction models [56, 57].

The magnitude of the relative error is called MRE and it is the absolute value of relative error. MRE gives a normalization of the difference of the estimated and actual effort, assessing a reliable measure for comparison of different models and techniques. MRE is calculated as follows:

$$MRE = \frac{|PM_{estimated} - PM_{actual}|}{PM_{actual}} \quad (5.1)$$

MMRE is the mean magnitude of the relative error which is the average percentage of the absolute values of the relative errors over a dataset. The equation for MMRE is as follows:

$$MMRE = \frac{1}{N} \sum_i \frac{|PM_{estimated} - PM_{actual}|}{PM_{actual}} \quad (5.2)$$

However, MMRE is largely sensitive to the large deviations in the predictions which are stated by MRE. To obtain a measure about the performance as a whole, a new measure median MRE (mdMRE) is used. This measure is the median of all the MREs.

PRED(L) is calculated from the relative error which states the difference between the actual and estimated effort. The equation for PRED(L) is below:

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

Here, the prediction accuracy is at the level of L where k is the number of observations where MRE is less than or equal to L . L is determined according to the accuracy expectations. Commonly 25 or 30 are used as values of L [2]. In the experiments, L is taken to be 30. It is suggested that a model is acceptable if PRED value is greater than or equal to 75 percent of the predicted values [52].

The implicit rationale for using relative error measures such as MRE, rather than an absolute one, is to have a measure that is independent of the output value. The concept of relative error makes intuitive sense to software researchers and practitioners because it is reasonable to predict effort of small and large systems with the competing relative error rather than the competing absolute error [56, 57]. This need is also supported by the use of PRED where it is reasonable to choose models with high PRED meaning high percentage of accuracy. On the other hand, MRE is also preferred when the variance of the data is not constant which is the case in our datasets. MMRE is used to select between competing models where models with low MMRE is preferred. It is also used to provide a quantitative measure of the uncertainty of a prediction where a low MMRE is taken to mean low uncertainty or inaccuracy.

Besides these relative metrics, since an average of many executions is going to be reported for every metric, a standard deviation (std. dev.) of all values for every metric is also included for every set of executions to see changing range of these values.

6. EXPERIMENTAL RESULTS

The experiments are conducted by using Matlab 6.5 Release 13 tool. The reasons for using Matlab are for simplicity of language of implementation and readily available functions that can be used within its library. The source codes of the functions are either self-developed or belong to Matlab or other third parties. We used SVM toolbox for SVM. This toolbox is developed at University of Southampton and can be found at [53].

It is important in our research that the datasets are reliable. There are two kinds of datasets in this research : based on COCOMO and based on COCOMO II. The two of the datasets are based on COCOMO. These datasets are collected by third parties such as NASA and USC and they are available to researchers. NASA dataset is reliable since the conclusions derived using this data is considered to be relevant to the general software engineering industry [48]. The other dataset is the original dataset collected by the creators, so it is reliable, too. The last dataset is based on COCOMO II parametric cost estimation model and it is collected by us through many software organizations in Turkey. This dataset (SDR) is relevant to the software engineering industry in Turkey.

The datasets are divided into two sets as explained in previous sections, namely training and test sets. Usually, the common approach in machine learning applications is to divide the dataset so that two thirds of the dataset is the training set and the remaining part is the test set. Although, we use this approach in our methodology, we also use another approach in which we keep the test set small. Our argument here is that the data to be estimated is small in size in practice, so it might make more sense to keep the test set small to reflect practical results.

For performance evaluation, cross validation is used where different sets of training and test sets are used and the mean of the results are taken to obtain more objective results. The number of times different sets are executed is taken to be 20-30 which is a general approach.

6.1. Experimental Results without Dimensionality Reduction

The first type of experiments is conducted by using every attribute in the datasets. The datasets are analyzed in their own context.

6.1.1. NASA Dataset

The dataset that have been collected by NASA includes 60 projects. The main characteristic of this dataset is that the types of the projects are similar considering that they have all been developed at NASA. Actually, this is the only dataset with such a characteristic. Four different machine learning methods have been applied to this dataset and the acquired result are given below.

In these experiments, there are two types of experiments according to the size of the training and test sets. The machine learning methods are applied firstly with 50 projects in the training set and 10 projects in the test set and secondly with 40 projects in the training set and 20 projects in the test set.

6.1.1.1. MLP Results. MLP with back-propagation is applied in two different ways as explained above according to the size of the training and test sets. In these experiments, there are common parameters concerning the algorithm.

One of the parameters is the learning rate. It is taken to be 0.005. Other learning rates between 0.001 and 0.03 have been tried. We found out that this learning rate produced more efficient results in terms of performance on error.

The second parameter is the number of hidden units. In both of the experiments, many number of hidden units have been tried such as 4, 8, 10, 12, 16, 20, 32. Though there are not significant improvements especially between close numbers, the best number of hidden units is found out to be 8 in both cases.

Last parameter is the maximum number of epochs which is taken to be 200 in both of the experiments.

The results for this algorithm and dataset are given in Table 6.1.

Table 6.1. Results for NASA dataset with MLP (%)

	MLP (50 + 10)	MLP (40+20)
MMRE (std. dev.)	110.94 (67)	230.62(437.53)
mdMRE (std. dev.)	68.01 (33)	76.89 (33.84)
PRED (std. dev.)	33 (19.49)	23 (18.45)

It can be assessed from the results that MLP does not perform well on improving MMRE. However, this does not show the general behavior of the MLP algorithm. Because, mdMRE value is low considering the MMRE value suggesting that there are exceptional cases that MLP perform very badly in our test set. It can also be seen from the PRED value that the performance is not that bad since %33 of the test set have been estimated with %30 or less MMRE. However, it is not acceptable as a good estimator.

The difference between the performances due to the different sizes of training and test sets shows clearly that the choice of a bigger training set improves the performance even though it is limited in this case. The improvement is around %52 by MMRE and around %43 by PRED.

6.1.1.2. RBF Results. There are two different experiments according to the size of the training and test sets, like in MLP experiments. In these experiments, the parameters are selected according to the performance obtained using the dataset.

One of the parameters is the learning rate. It is taken to be 0.01. Other learning rates between 0.001 and 0.03 have been tried. We found out that this learning rate produced more efficient results in terms of performance on error.

The second parameter is the number of hidden units. In both of the experiments, many number of hidden units have been tried such as 4, 8, 10, 12, 16, 20, 32. Though there are not significant improvements especially between close numbers, the best number of hidden units is found out to be 8 in both cases.

Last parameter is the maximum number of epochs which is taken to be 100 in both of the experiments.

The results for this algorithm and dataset are given in Table 6.2.

Table 6.2. Results for NASA dataset with RBF (%)

	RBF (50 + 10)	RBF (40+20)
MMRE (std. dev.)	22.11 (5.24)	22.39 (4.68)
mdMMRE (std. dev.)	14.44 (7.41)	15.47 (5.90)
PRED (std. dev.)	73.25 (9.63)	73 (14.90)

In this case, MMRE values are pretty good. This shows that unlike in MLP, RBF doesn't produce high errors, thus keeping the MMRE in a limited range. PRED values are also good. However, it is not regarded as a good estimator since it is less than 75%.

The distinction between the results of different training set sizes indicates that choosing a bigger training set increases accuracy and performance. The usage of a bigger training set improved MMRE by around %1 and PRED by around %0.3.

6.1.1.3. DT Results. Decision tree method is applied in two different ways as in MLP and RBF algorithms. There are no controlling parameters in DT except the threshold value determining the acceptable error which is $1e^{-5}$. The results are given in Table 6.3.

In the results, MMRE values are very promising. The analysis of mdMMRE, MMRE and PRED values together suggests that the majority of instances in the test set are estimated with less than %30 accuracy. However, there are also exceptional instances that keep the MMRE high. DT can be regarded as a good estimator here.

Table 6.3. Results for NASA dataset with DT (%)

	DT (50 + 10)	DT (40+20)
MMRE (std. dev.)	12.85 (4.54)	13.10 (7.18)
mdMRE (std. dev.)	5.31 (1.50)	6.29 (5.41)
PRED (std. dev.)	83.75 (7.27)	83.50 (12.68)

The difference between the results according to the size of the training and test sets are as expected. The choice of a bigger training set improves performance around %2 by MMRE and around %0.2 by PRED. Although the increase in MMRE is due to its small magnitude, the usage of a bigger training set is considerable, here.

6.1.1.4. SVM Results. This method is also applied in two different ways according to the different sizes of the training set. The linear kernel function is used because of its accuracy and fastness. The results are given below in Table 6.4.

Table 6.4. Results for NASA dataset with SVM (%)

	SVM (50 + 10)	SVM (40+20)
MMRE (std. dev.)	25.72 (13.83)	47.56 (87.20)
mdMRE (std. dev.)	9.39 (5.48)	14.95 (27.97)
PRED (std. dev.)	75.75 (8.63)	69.50 (14.13)

In the results, MMRE and PRED values are good. The analysis of MMRE, mdMRE and PRED suggests that there are instances that have high MMREs, although the majority of the instances have less than or equal to %30 MMRE. This method can be regarded as a good estimator when the training set is bigger, although the value of PRED is on the limit.

The difference between the results according to the size of the training and test sets are as in the other methods. The choice of a bigger training set improves performance around %45 by MMRE and around %7 by PRED. This suggests that usage of a bigger training set is considerable.

6.1.2. USC Dataset

The dataset that have been collected by University of South California (USC) includes 63 projects. The main characteristic of this dataset is that they are collected from different software organizations. They are not expected to possess similar characteristics such as the type of the project. Four different machine learning methods have been applied to this dataset and the acquired results are given below.

In these experiments, there are two types of experiments according to the size of the training and test sets. Every machine learning method is applied firstly with 53 projects in training set and 10 projects in test set and secondly with 42 projects in training set and 21 projects in test set.

6.1.2.1. MLP Results. There are two different experiments according to the size of the training and test sets. In these experiments, there are common parameters concerning the algorithm.

One of the parameters is the learning rate. It is taken to be 0.005. Other learning rates between 0.001 and 0.03 have been tried. We found out that this learning rate produced more efficient results in terms of performance on error.

The second parameter is the number of hidden units. In both of the experiments, many number of hidden units have been tried such as 4, 8, 10, 12, 16, 20, 32. Though there are not significant improvements especially between close numbers, the best number of hidden units is found out to be 8 in both cases.

Last parameter is the maximum number of epochs which is taken to be 200 in both of the experiments.

The results for this algorithm and dataset are given in Table 6.5.

Table 6.5. Results for USC dataset with MLP (%)

	MLP (53 + 10)	MLP (42+21)
MMRE (std. dev.)	113.13 (85.35)	121.97(145.32)
mdMRE (std. dev.)	81.46 (45.75)	82.55 (42.46)
PRED (std. dev.)	13.49 (11.27)	8.12 (10.10)

It can be assessed from the results that MLP does not perform well on improving MMRE. PRED values are not good, either. The analysis of all the results enables us to state that MLP does not perform well on this dataset, although parameters of the algorithm are chosen carefully to maximize performance. Also, it is not a good estimator considering the PRED value.

The difference between the performances due to the different sizes of training and test sets shows clearly that the choice of a bigger training set improves the performance around %7 by MMRE and around %66 by PRED.

6.1.2.2. RBF Results. There are two different experiments according to the size of the training and test sets, like in MLP experiments. In these experiments, the parameters are selected according to the performance obtained using the dataset.

One of the parameters is the learning rate. It is taken to be 0.01. Other learning rates between 0.001 and 0.03 have been tried. We found out that this learning rate produced more efficient results in terms of performance on error.

The second parameter is the number of hidden units. In both of the experiments, many number of hidden units have been tried such as 4, 8, 10, 12, 16, 20, 32. Though there are not significant improvements especially between close numbers, the best number of hidden units is found out to be 8 in both cases.

Last parameter is the maximum number of epochs which is taken to be 100 in both of the experiments.

The results for this algorithm and dataset are given in Table 6.6.

Table 6.6. Results for USC dataset with RBF (%)

	RBF (53 + 10)	RBF (42 + 21)
MMRE (std. dev.)	13.38 (4.52)	14.56 (3.07)
mdMRE (std. dev.)	8.42 (3.90)	9.61 (3.65)
PRED (std. dev.)	89 (8.52)	87.56 (5.67)

In this case, MMRE values are pretty good and so is mdMRE. This shows that unlike in MLP, RBF doesn't produce high errors, thus keeping the MMRE in a limited range. PRED values are also very good indicating the behavior of a good estimator.

The distinction between the results of different training set sizes indicates that choosing a bigger training set increases accuracy and performance. The usage of a bigger training set improved MMRE by around %8 and PRED by around %1.

6.1.2.3. DT Results. Decision tree method is applied in two different ways as in MLP and RBF algorithms. There are no controlling parameters in DT except the threshold value determining the acceptable error which is $1e^{-5}$. The results are given in Table 6.7.

Table 6.7. Results for USC dataset with DT (%)

	DT (53 + 10)	DT (42 + 21)
MMRE (std. dev.)	13.57 (6.04)	14.25 (9.13)
mdMRE (std. dev.)	3.98 (2.20)	5.18 (3.90)
PRED(30) (std. dev.)	85.24 (6.54)	85 (9.46)

In the results, MMRE values are very promising. The analysis of mdMRE and MMRE values together suggests that the majority of instances in the test set are estimated with less than %30 accuracy. This is also supported by PRED value which states that DT is a good estimator.

The difference between the results according to the size of the training and test sets are as expected. The choice of a bigger training set improves performance around %5 by MMRE and around %0.2 by PRED. The choice of a bigger dataset does not make a big difference by all means in this case.

6.1.2.4. SVM Results. This method is also applied in two different ways according to the different sizes of the training set. The linear kernel function is used because of its accuracy and fastness. The results are given below in Table 6.8.

Table 6.8. Results for USC dataset with SVM (%)

	SVM (53 + 10)	SVM (42+21)
MMRE (std. dev.)	33.28 (30.02)	33.39 (17.95)
mdMMRE (std. dev.)	14.83 (6.12)	16.45 (7.09)
PRED (std. dev.)	75.5 (15.38)	70 (15.46)

In the results, MMRE and PRED values are very good. The analysis of MMRE, mdMMRE and PRED suggests that there are exceptional instances that have high MMRE but in an acceptable range. SVM can be regarded as a good estimator when a bigger training set is used.

The difference between the results according to the size of the training and test sets are as in the other methods. The choice of a bigger training set improves performance around %0.3 by MMRE and around %8. These values suggest that choosing a bigger training set is considerable.

6.1.3. SDR Dataset

The dataset that we have collected includes 23 projects. The main characteristic of this dataset is that they are collected from different software development organizations in Turkey. On the whole, they are medium sized software projects that include automation or banking applications. Four different machine learning methods have been applied to this dataset and the results are given below.

In these experiments, there are two types of experiments according to the size of the training and test sets. Every machine learning method is applied firstly with 20 projects in training set and 3 projects in test set and secondly with 16 projects in training set and 7 projects in test set.

6.1.3.1. MLP Results. There are two different experiments according to the size of the training and test sets. In these experiments, there are common parameters concerning the algorithm.

One of the parameters is the learning rate. It is taken to be 0.005. Other learning rates between 0.001 and 0.03 have been tried. We found out that this learning rate produced more efficient results in terms of performance on error.

The second parameter is the number of hidden units. In both of the experiments, many number of hidden units have been tried such as 4,8,10,12,16,20,32. Though there are not significant improvements especially between close numbers, the best number of hidden units is found out to be 8 in both cases.

Last parameter is the maximum number of epochs which is taken to be 200 in both of the experiments.

The results for this algorithm and dataset are given in Table 6.9.

Table 6.9. Results for SDR dataset with MLP (%)

	MLP (20 + 3)	MLP (16+7)
MMRE (std. dev.)	121.67(134.25)	150.78(110.38)
mdMRE (std. dev.)	63.21 (37.72)	86.27 (33)
PRED (std. dev.)	41.67 (14.81)	25.71 (12.78)

It can be assessed from the results that MLP does not perform well on improving MMRE. PRED values are not good, either. The analysis of all the results enables us to state that MLP does not perform well on this dataset, although parameters of the algorithm

are chosen carefully to maximize performance. It is not a good estimator considering the PRED value.

The difference between the performances due to the different sizes of training and test sets shows clearly that the choice of a bigger training set improves the performance around %19 by MMRE and around %62 by PRED.

6.1.3.2. RBF Results. There are two different experiments according to the size of the training and test sets, like in MLP experiments. In these experiments, the parameters are selected according to the performance obtained using the dataset.

One of the parameters is the learning rate. It is taken to be 0.01. Other learning rates between 0.001 and 0.03 have been tried. We found out that this learning rate produced more efficient results in terms of performance on error.

The second parameter is the number of hidden units. In both of the experiments, many number of hidden units have been tried such as 4, 8, 10, 12, 16, 20, 32. Though there are not significant improvements especially between close numbers, the best number of hidden units is found out to be 8 in both cases.

Last parameter is the maximum number of epochs which is taken to be 100 in both of the experiments, because convergence occurs before that.

The results for this algorithm and dataset are given in Table 6.10.

Table 6.10. Results for SDR dataset with RBF (%)

	RBF (20 + 3)	RBF (16+7)
MMRE (std. dev.)	22.60 (12.12)	23.47 (7.34)
mdMMRE (std. dev.)	18.99 (16.26)	16.12 (8.28)
PRED (std. dev.)	72.14 (11.79)	65 (25.31)

In this case, MMRE values are pretty good. All values show that unlike in MLP, RBF doesn't produce high errors, thus keeping the MMRE in a limited range. PRED values are not good enough to be regarded as a good estimator.

The distinction between the results of different training set sizes indicates that choosing a bigger training set increases accuracy and performance. The usage of a bigger training set improved MMRE by around %4 and PRED by around %11.

6.1.3.3. DT Results. Decision tree method is applied in two different ways as in MLP and RBF algorithms. There are no controlling parameters in DT except the threshold value determining the acceptable error which is $1e^{-5}$. The results are given in Table 6.11.

Table 6.11. Results for SDR dataset with DT (%)

	DT (20 + 3)	DT (16+7)
MMRE (std. dev.)	14.55 (9.42)	20.41 (9.48)
mdMMRE (std. dev.)	11.06 (5.94)	12.78 (8.85)
PRED (std. dev.)	90 (15.67)	74.29 (15.09)

In the results, MMRE values are very promising. The analysis of all values together suggests that the majority of instances in the test set are estimated with less than %30 accuracy. PRED value suggests that DT is a good estimator when the training set is bigger.

The difference between the results according to the size of the training and test sets are as expected. The choice of a bigger training set improves performance around %29 by MMRE and around %21 by PRED. The choice of a bigger training set does not make a big difference by all means in this case.

6.1.3.4. SVM Results. This method is also applied in two different ways according to the different sizes of the training set. The linear kernel function is used because of its accuracy and fastness. The results are given below in Table 6.12.

Table 6.12. Results for SDR dataset with SVM (%)

	SVM (20 + 3)	SVM (16 + 7)
MMRE (std. dev.)	187.26(174.64)	264.30(279.57)
mdMRE (std. dev.)	97.29 (57.07)	168.99 (95.93)
PRED (std. dev.)	22.86 (15.65)	18.33 (20.16)

In the results, MMRE and PRED values are very good. The analysis of MMRE, mdMRE and PRED suggests that there are exceptional instances that have high MMREs but in an acceptable range. PRED value is not enough for SVM to be regarded as a good estimator.

The difference between the results according to the size of the training and test sets are as in the other methods. The choice of a bigger training set improves performance around %29 by MMRE and around %28. These values suggest that choosing a bigger training set is considerable.

6.2. Experimental Results with Dimensionality Reduction using PCA

6.2.1. NASA Dataset

The dataset that have been collected by NASA includes 60 projects. In the experiments that will be explained below, there is an initial application of PCA algorithm which makes dimensionality reduction. For this dataset, the number of dimensions that this dataset will be reduced to is 13. The scree graph explaining this is given in Figure 6.1 below. As can be seen from this graph, most of the data can be explained by 13 attributes. After this operation, four different machine learning methods are applied to the dataset obtained from PCA and the acquired result are given below.

In these experiments, there are two types of experiments according to the size of the training and test sets. Every machine learning method is applied firstly to the dataset with 50 projects in training set and 10 projects in test set and secondly to the dataset with 40 projects in training set and 20 projects in test set.

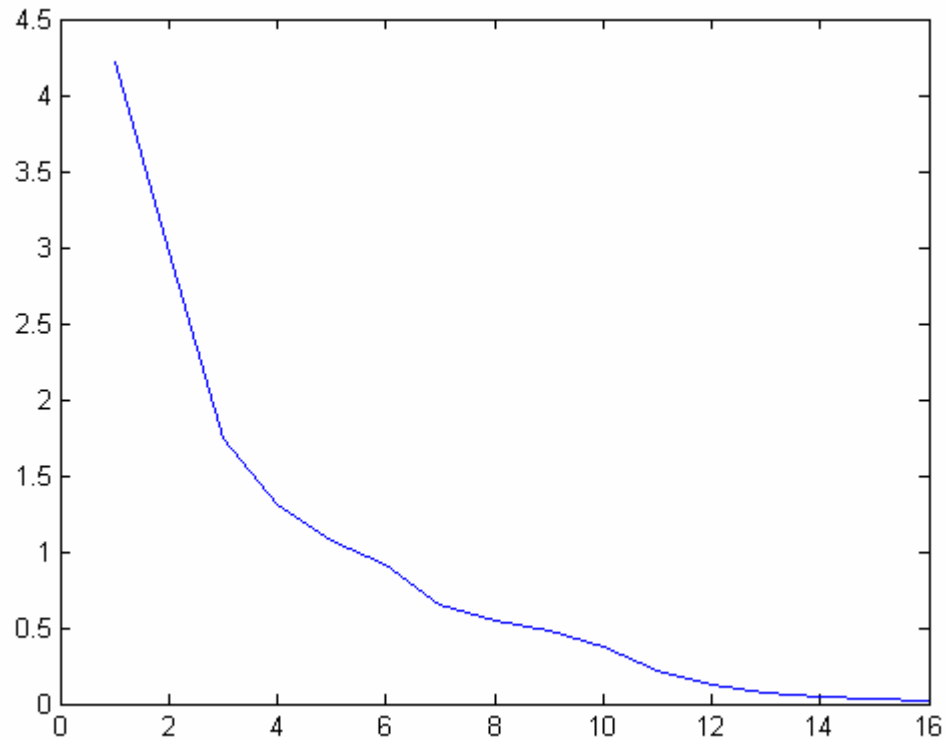


Figure 6.1. Scree graph for NASA dataset

6.2.1.1. MLP Results. MLP with back-propagation is applied in two different ways as explained above according to the size of the training and test sets. In these experiments, there are common parameters concerning the algorithm.

One of the parameters is the learning rate. It is taken to be 0.005. Other learning rates between 0.001 and 0.03 have been tried. We found out that this learning rate produced more efficient results in terms of performance on error.

The second parameter is the number of hidden units. In both of the experiments, many number of hidden units have been tried such as 4, 8, 10, 12, 16, 20, 32. Though there are not significant improvements especially between close numbers, the best number of hidden units is found out to be 8 in both cases.

Last parameter is the maximum number of epochs which is taken to be 200 in both of the experiments, because convergence occurs before that.

The results for this algorithm and dataset are given in Table 6.13.

Table 6.13. Results for NASA dataset with MLP (%)

	MLP (50 + 10)	MLP (40+20)
MMRE (std. dev.)	109.96 (71.28)	166.57(306.01)
mdMRE (std. dev.)	65.75(35.37)	74.58 (27.04)
PRED (std. dev.)	34 (20.88)	24.50 (16.38)

It can be assessed from the results that MLP does not perform well on improving MMRE. However, this does not show the general behavior of the MLP algorithm. Because, mdMRE value is very low considering the MMRE value suggesting that there are exceptional cases that MLP perform very badly in our test set. It can also be seen from the PRED value that the performance is not that bad since almost %40 of the test set have been estimated with %30 MRE. However, MLP can not be regarded as a good estimator.

The difference between the performances due to the different sizes of training and test sets shows clearly that the choice of a bigger training set improves the performance around %34 by MMRE and around %39 by PRED.

6.2.1.2. RBF Results. There are two different experiments according to the size of the training and test sets, like in MLP experiments. In these experiments, the parameters are selected according to the performance obtained using the dataset.

One of the parameters is the learning rate. It is taken to be 0.01. Other learning rates between 0.001 and 0.03 have been tried. We found out that this learning rate produced more efficient results in terms of performance on error.

The second parameter is the number of hidden units. In both of the experiments, many number of hidden units have been tried such as 4, 8, 10, 12, 16, 20, 32. Though there are not significant improvements especially between close numbers, the best number of hidden units is found out to be 8 in both cases.

Last parameter is the maximum number of epochs which is taken to be 100 in both of the experiments, because convergence occurs before that.

The results for this algorithm and dataset are given in Table 6.14.

Table 6.14. Results for NASA dataset with RBF (%)

	RBF (50 + 10)	RBF (40+20)
MMRE (std. dev.)	20.19 (4.92)	20.27 (4.87)
mdMRE (std. dev.)	11.78 (6.49)	11.99 (6.74)
PRED (std. dev.)	75.25 (9.03)	75 (11)

In this case, MMRE values are pretty good and all values show that unlike in MLP, RBF doesn't produce high errors, thus keeping the MMRE in a limited range. PRED values are also very good indicating a good estimator's performance.

The distinction between the results of different training set sizes indicates that choosing a bigger training set increases accuracy and performance. The usage of a bigger training set improved MMRE by around %0.3 and PRED by around %0.3.

6.2.1.3. DT Results. Decision tree method is applied in two different ways as in MLP and RBF algorithms. There are no controlling parameters in DT except the threshold value determining the acceptable error which is $1e^{-5}$. The results are given in Table 6.15.

Table 6.15. Results for NASA dataset with DT (%)

	DT (50 + 10)	DT (40+20)
MMRE (std. dev.)	22.51 (7.74)	23.22 (15.82)
mdMRE (std. dev.)	11.03 (6.24)	12.58 (6.36)
PRED (std. dev.)	72 (13.99)	67.75 (10.70)

In the results, MMRE values are very promising. The analysis of mdMRE and MMRE values together suggests that the majority of instances in the test set are estimated

with less than or equal to %30 accuracy. The value of PRED shows that there are also instances with MMRE greater than %30 that keep the MMRE high. Here, DT cannot be regarded as a good estimator considering this value.

The difference between the results according to the size of the training and test sets are as expected. The choice of a bigger training set improves the performance around %3 by MMRE and around %6 by PRED.

6.2.1.4. SVM Results. This method is also applied in two different ways according to the different sizes of the training set. The linear kernel function is used because of its accuracy and fastness. The results are given below in Table 6.16.

Table 6.16. Results for NASA dataset with SVM (%)

	SVM (50 + 10)	SVM (40+20)
MMRE (std. dev.)	23.11 (8.56)	23.43 (11.52)
mdMMRE (std. dev.)	7.53 (3.93)	8.71 (4.98)
PRED (std. dev.)	77 (13.80)	75 (11.47)

In the results, MMRE and PRED values are good. By analyzing the values, it can be said that the performance is pretty good and majority of the instances have less than %30 MMRE making the PRED value high. SVM can be regarded as a good estimator.

The difference between the results according to the size of the training and test sets are as in the other methods. The choice of a bigger training set improves performance around %1 by MMRE and around %3 by PRED value.

6.2.2. USC Dataset

The dataset that have been collected by USC includes 63 projects. In the experiments that will be explained below, there is an initial application of PCA algorithm which makes dimensionality reduction. For this dataset, the number of dimensions that this dataset will be reduced to is 13. The scree graph explaining this is given in Figure 6.2 below. As can be seen from this graph, most of the data can be explained by 13 attributes. After this

operation, four different machine learning methods are applied to the dataset obtained from PCA and the acquired result are given below.

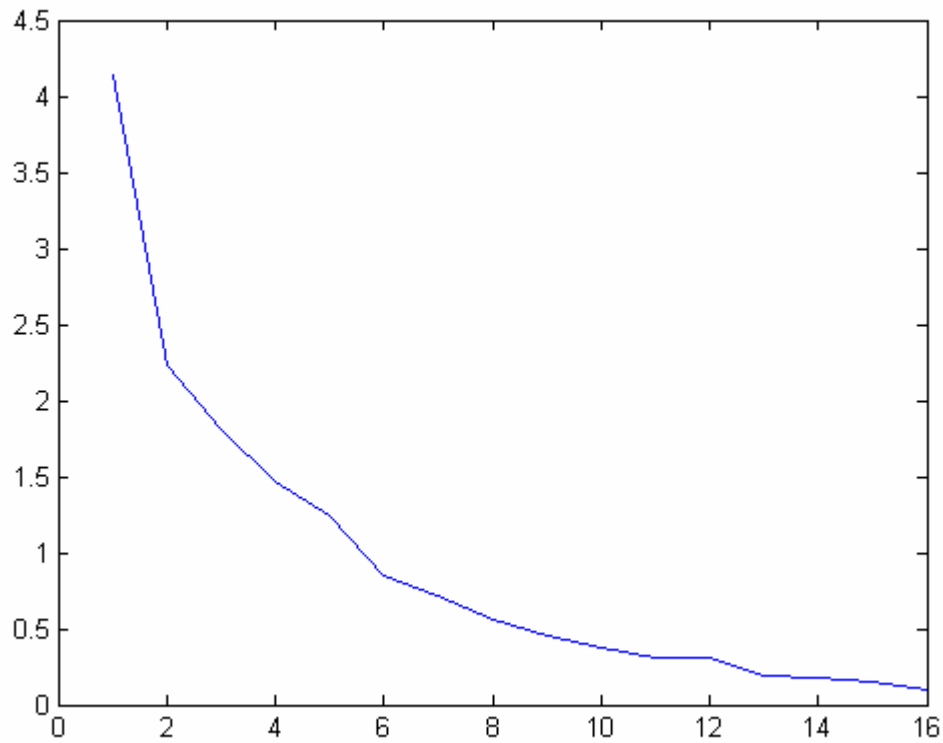


Figure 6.2. Scree graph for USC dataset

In these experiments, there are two types of experiments according to the size of the training and test sets. Every machine learning method is applied firstly to the dataset with 53 projects in training set and 10 projects in test set and secondly to the dataset with 42 projects in training set and 21 projects in test set.

6.2.2.1. MLP Results. This method is also applied in two different ways according to the different sizes of the training set. In these experiments, there are common parameters concerning the algorithm.

One of the parameters is the learning rate. It is taken to be 0.005. Other learning rates between 0.001 and 0.03 have been tried and this learning rate has found out to produce more efficient results in terms of performance on error.

The second parameter is the number of hidden units. In both of the experiments, many number of hidden units have been tried such as 4, 8, 10, 12, 16, 20, 32. Though there are not significant improvements especially between close numbers, the best number of hidden units is found out to be 8 in both cases.

Last parameter is the maximum number of epochs which is taken to be 200 in both of the experiments.

The results for this algorithm and dataset are given in Table 6.17.

Table 6.17. Results for USC dataset with MLP (%)

	MLP (53 + 10)	MLP (42+21)
MMRE (std. dev.)	101.35 (66.58)	116.43 (96.87)
mdMRE (std. dev.)	80.76 (43.64)	73.64 (32.15)
PRED (std. dev.)	15.62 (13.85)	8.50 (10.27)

It can be assessed from the results that MLP does not perform well on improving MMRE. This can also be supported by the values of PRED, although the parameters of the algorithm are chosen carefully to maximize performance. Clearly, MLP can not be regarded as a good estimator.

The difference between the performances due to the different sizes of training and test sets shows clearly that the choice of a bigger training set improves the performance around %13 by MMRE and around %84 by PRED. This makes the usage of a bigger training set considerable.

6.2.2.2. RBF Results. In these experiments, there are also two different experiments according to the size of the training and test sets, like in MLP experiments. In these experiments, the parameters are selected according to the performance obtained using the dataset.

One of the parameters is the learning rate. It is taken to be 0.01. Other learning rates between 0.001 and 0.03 have been tried and this learning rate has found out to produce more efficient results in terms of performance on error.

The second parameter is the number of hidden units. In both of the experiments, many number of hidden units have been tried such as 4, 8, 10, 12, 16, 20, 32. Though there are not significant improvements especially between close numbers, the best number of hidden units is found out to be 8 in both cases.

Last parameter is the maximum number of epochs which is taken to be 100 in both of the experiments.

The results for this algorithm and dataset are given in Table 6.18.

Table 6.18. Results for USC dataset with RBF (%)

	RBF (53 + 10)	RBF (42+21)
MMRE (std. dev.)	12.15 (3.38)	12.46 (5.01)
mdMMRE (std. dev.)	7.44 (3.48)	7.46 (2.69)
PRED (std. dev.)	90.48 (5.12)	90 (9.73)

In this case, MMRE values are pretty good and all values show that unlike in MLP, RBF doesn't produce high errors, thus keeping the MMRE in a limited range. PRED value is best of all the methods with PCA for this dataset and shows that RBF is a good estimator for all combinations of dataset.

The distinction between the results of different training set sizes indicates that choosing a bigger training set increases accuracy and performance. The usage of a bigger training set improved MMRE by around %2 and PRED by around %0.53.

6.2.2.3. DT Results. Decision tree method is applied in two different ways as in MLP and RBF algorithms. There are no controlling parameters in DT except the threshold value determining the acceptable error which is $1e^{-5}$. The results are given in Table 6.19.

Table 6.19. Results for USC dataset with DT (%)

	DT (53 + 10)	DT (42+21)
MMRE (std. dev.)	33.12 (25.10)	37.97 (22.45)
mdMRE (std. dev.)	9.15 (4)	17.19 (19.01)
PRED(std. dev.)	72.14 (9.68)	66 (16.67)

In the results, MMRE values are very promising. The analysis of mdMRE and MMRE values together suggest that the majority of instances in the test set are estimated with less than %30 accuracy. The value of PRED shows that there are exceptional instances that keep the MMRE high and DT is not a good estimator in this case.

The difference between the results according to the size of the training and test sets are as expected. In this case, the choice of a bigger training set improves the performance according to MMRE and PRED values. However, the increase in performance is around %13 by MMRE and %9 by PRED. These values suggest that the size does not affect the performance a great deal.

6.2.2.4. SVM Results. This method is also applied in two different ways according to the different sizes of the training set. The linear kernel function is used because of its accuracy and fastness. The results are given below in Table 6.20.

Table 6.20. Results for USC dataset with SVM (%)

	SVM (53 + 10)	SVM (42+21)
MMRE (std. dev.)	30.94 (19.25)	31.02 (13.06)
mdMRE (std. dev.)	13.77 (4.79)	15.66 (6.18)
PRED (std. dev.)	82 (11.05)	73.81 (14.29)

In the results, MMRE and PRED values are good. PRED value shows that SVM is a good estimator when a bigger training set is used.

The difference between the results according to the size of the training and test sets affects the results as in the other methods. The choice of a bigger training set improves performance by all means. The usage of a bigger training set improves performance %0.2 by MMRE and around %11 by PRED.

6.2.3. SDR Dataset

The dataset that we have collected includes 23 projects. The difference in these experiments with this dataset is that the dimension of the dataset is reduced to 10. This value is obtained by doing a scree graph analysis. In figure 6.3, it can be seen that the majority of data can be explained by 10 features. This experiment is conducted to see whether the accuracy of the learning system is bettered when the dataset is wiped off those extra features. After this, four different machine learning methods have been applied to this dataset and the acquired results are given below.

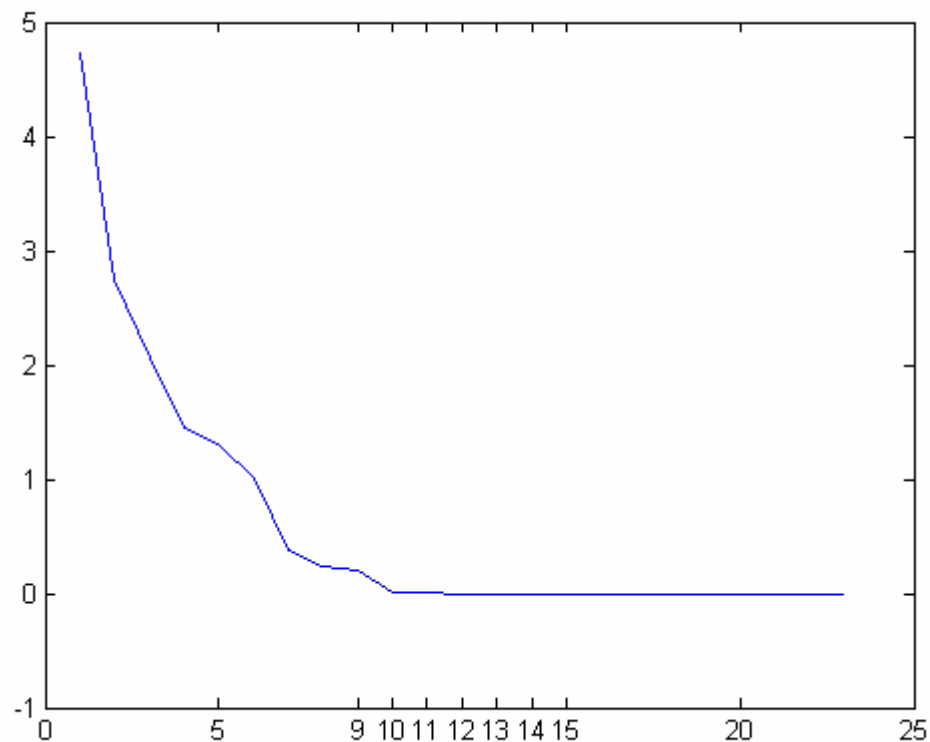


Figure 6.3. Scree graph for SDR dataset

In these experiments, there are two types of experiments according to the size of the training and test sets. Every machine learning method is applied firstly to the dataset with 20 projects in training set and 3 projects in test set and secondly to the dataset with 16 projects in training set and 7 projects in test set.

6.2.3.1. MLP Results. This method is also applied in two different ways according to the different sizes of the training set. In these experiments, there are common parameters concerning the algorithm.

One of the parameters is the learning rate. It is taken to be 0.005. Other learning rates between 0.001 and 0.03 have been tried and this learning rate has found out to produce more efficient results in terms of performance on error.

The second parameter is the number of hidden units. In both of the experiments, many number of hidden units have been tried such as 4, 8, 10, 12, 16, 20, 32. Though there are not significant improvements especially between close numbers, the best number of hidden units is found out to be 8 in both cases.

Last parameter is the maximum number of epochs which is taken to be 200 in both of the experiments.

The results for this algorithm and dataset are given in Table 6.21.

Table 6.21. Results for SDR dataset with MLP (%)

	MLP (20 + 3)	MLP (16+7)
MMRE (std. dev.)	116.16(106.89)	123.27 (84.76)
mdMRE (std. dev.)	68.20 (37.23)	80.35 (29.15)
PRED (std. dev.)	45 (16.31)	27.86 (15.70)

It can be assessed from the results that MLP does not perform well on improving MMRE. This can also be supported by the values of PRED, although the parameters of the

algorithm are chosen carefully to maximize performance. PRED shows also that MLP is not a good estimator.

The difference between the performances due to the different sizes of training and test sets shows clearly that the choice of a bigger training set improves the performance around %6 by MMRE and around %61 by PRED. This makes the usage of a bigger training set considerable.

6.2.3.2. RBF Results. In these experiments, there are also two different experiments according to the size of the training and test sets, like in MLP experiments. In these experiments, the parameters are selected according to the performance obtained using the dataset.

One of the parameters is the learning rate. It is taken to be 0.01. Other learning rates between 0.001 and 0.03 have been tried and this learning rate has found out to produce more efficient results in terms of performance on error.

The second parameter is the number of hidden units. In both of the experiments, many number of hidden units have been tried such as 4, 8, 10, 12, 16, 20, 32. Though there are not significant improvements especially between close numbers, the best number of hidden units is found out to be 8 in both cases.

Last parameter is the maximum number of epochs which is taken to be 100 in both of the experiments.

The results for this algorithm and dataset are given in Table 6.22.

Table 6.22. Results for SDR dataset with RBF (%)

	RBF (20 + 3)	RBF (16 + 7)
MMRE (std. dev.)	21.01 (11.41)	23.44 (6.54)
mdMMRE (std. dev.)	18.67 (14.97)	18.30 (8.03)
PRED (std. dev.)	73.33 (25.59)	68.57 (13.59)

In this case, MMRE values are pretty good. All values show that unlike in MLP, RBF doesn't produce high errors, thus keeping the MMRE in a limited range. PRED values are also good but it cannot be regarded as a good estimator.

The distinction between the results of different training set sizes indicates that choosing a bigger training set increases accuracy and performance. The usage of a bigger training set improved MMRE by around %10 and PRED by around %7.

6.2.3.3. DT Results. DT method is applied in two different ways as in MLP and RBF algorithms. There are no controlling parameters in DT except the threshold value determining the acceptable error which is $1e^{-5}$. The results are given in Table 6.23.

Table 6.23. Results for SDR dataset with DT (%)

	DT (20 + 3)	DT (16 + 7)
MMRE (std. dev.)	27.58 (14.18)	36.42 (37.52)
mdMMRE (std. dev.)	13.57 (5.75)	14.42 (12.10)
PRED (std. dev.)	73.57 (11.61)	70 (18.42)

In the results, MMRE values are very promising. The analysis of the values together suggests that the majority of instances in the test set are estimated with less than %30 accuracy. The value of PRED shows that there are exceptional instances that keep the MMRE high. However, DT cannot be regarded as a good estimator, here.

The difference between the results according to the size of the training and test sets are as expected. In this case, the choice of a bigger training set improves the performance according to MMRE and PRED values. However, the increase in performance is around %24 by MMRE and %5 by PRED.

6.2.3.4. SVM Results. This method is also applied in two different ways according to the different sizes of the training set. The linear kernel function is used because of its accuracy and fastness. The results are given below in Table 6.24.

Table 6.24. Results for SDR dataset with SVM (%)

	SVM (20 + 3)	SVM (16 + 7)
MMRE (std. dev.)	60.45 (58.06)	61.02 (50.63)
mdMMRE (std. dev.)	25.71 (13.49)	35.49 (21.86)
PRED (std. dev.)	56.43 (12.67)	53.33 (31.34)

In the results, MMRE and PRED values are good. This is also supported by the PRED value. These values show that SVM is not a good estimator here for all combinations of dataset.

The difference between the results according to the size of the training and test sets affects the results as in the other methods. The choice of a bigger training set improves performance by all means. The usage of a bigger training set improves performance %1 by MMRE and around %6 by PRED.

6.3. Method Comparisons

The summary of all the results for NASA dataset are given below in Table 6.25. From the table, it is observable that DT is the best performer. One common behavior of all the algorithms are that the error increases when the size of the training set decreases, although in cases such as RBF and DT, the decrease is little in comparison. The decrease is acceptable since the system learns more about the data once there are more instances to use at the training phase.

Another common behavior for MLP, RBF and SVM is the decrease in error when PCA is applied as an initial method to decrease number of dimensions. This behavior may suggest the existence of unnecessary attributes in COCOMO model, especially considering the difference in MMRE values. However, the application of PCA before DT results in an increase in error. This may be due to the fact that decision trees are better estimators on categorical values. Our datasets are also categorical although numerical ratings are used instead of categorical values. When PCA is applied these categorical values are a bit

distorted. So the decision tree performs worse when PCA is applied. Altogether, these results show that PCA must be optional according to the type of learning algorithm used.

Table 6.25. Summary of experimental results for NASA dataset (%)

NASA (# of training)	Without PCA		With PCA	
	MMRE	PRED	MMRE	PRED
	(std. dev.)	(std. dev.)	(std. dev.)	(std. dev.)
	mdMRE		mdMRE	
	(std. dev.)		(std. dev.)	
MLP (50)	110.94(67) 68.01(33)	33 (19.49)	109.96(71.28) 65.75(35.37)	34 (20.88)
MLP (40)	230.62 (437.53) 76.89(33.84)	23 (18.45)	166.57 (306.01) 74.58(27.04)	24.50 (16.38)
RBF (50)	22.11 (5.24) 14.44 (7.41)	73.25 (9.63)	20.19 (4.92) 11.78 (6.49)	75.25 (9.03)
RBF (40)	22.39 (4.68) 15.47 (5.90)	73 (14.90)	20.27 (4.87) 11.99 (6.74)	75 (11)
DT (50)	12.85 (4.54) 5.31 (1.50)	83.75 (7.27)	22.51 (7.74) 11.03 (6.24)	72 (13.99)
DT (40)	13.10(7.18) 6.29(5.41)	83.50 (12.68)	23.22(15.82) 12.58(6.36)	67.75 (10.70)
SVM (50)	25.72(13.83) 9.39 (5.48)	75.75 (8.63)	23.11 (8.56) 7.53 (3.93)	77 (13.80)
SVM (40)	47.56(87.20) 14.95(27.97)	69.50 (14.13)	23.43 (11.52) 8.71 (4.98)	75 (11.47)

When COCOMO's own parametric model that has been explained in Section 2.1.1 is used, the mdMRE is 18.63, MMRE is 29.50 and PRED(30) is 73.33. Considering this, the results of RBF, DT and SVM are better than or equal to the results for COCOMO parametric model. This shows that the learning system is highly compatible with the

COCOMO model in estimating effort. DT method is satisfying as an estimator since PRED is greater than 75, whereas in the parametric model PRED is less than 75.

Table 6.26. Summary of experimental results for USC dataset (%)

USC (# of training)	Without PCA		With PCA	
	MMRE	PRED	MMRE	PRED
	(std. dev.)	(std. dev.)	(std. dev.)	(std. dev.)
	mdMRE (std. dev.)		mdMRE (std. dev.)	
MLP (53)	113.13 (85.35) 81.46(45.75)	13.49 (11.27)	101.35 (66.58) 80.76(43.64)	15.62 (13.85)
MLP (42)	121.97 (145.32) 82.55(42.46)	8.12 (10.10)	116.43 (96.87) 73.64(32.15)	8.50 (10.27)
RBF (53)	13.38 (4.52) 8.42 (3.90)	89 (8.52)	12.15 (3.38) 7.44 (3.48)	90.48 (5.12)
RBF (42)	14.56 (3.07) 9.61 (3.65)	87.86 (5.67)	12.46 (5.01) 7.46 (2.69)	90 (9.73)
DT (53)	13.57 (6.04) 3.98 (2.20)	85.24 (6.54)	33.12(25.10) 9.15 (4)	72.14 (9.68)
DT (42)	14.25 (9.13) 5.18 (3.90)	85 (9.46)	37.97(22.45) 17.19(19.01)	66 (16.67)
SVM (53)	33.28(30.02) 14.83 (6.12)	75.5 (15.38)	30.94(19.25) 13.77 (4.79)	82 (11.05)
SVM (42)	33.39(17.95) 16.45 (7.09)	70 (15.46)	31.02(13.06) 15.66 (6.18)	73.81 (14.29)

The summary of all the results for USC dataset are given in Table 6.26. For this dataset, the general observations are the almost the same as stated for NASA dataset. Here, MMRE increases when the training set's size decreases. Again, the decrease is low for methods such as RBF and DT.

For this dataset, for MLP, RBF and SVM methods, the initial application of PCA causes a decrease in MMRE. However, it causes an increase in error for DT method. But, unlike NASA dataset, PCA is considerable here, since the best performing combination of methods is RBF and PCA. These results show the existence of unnecessary attributes in the dataset probably caused by low deviation.

When COCOMO's own parametric model that has been explained in Section 2.1.1 is used, the mdMRE is 28.47, MMRE is 49.16 and PRED(30) is 50.79. Considering this, the results of RBF, DT and SVM are better than or equal to the results for COCOMO parametric model. This shows that the learning system is highly compatible with the COCOMO model in estimating effort.

The summary of all the results for SDR dataset are given below in Table 6.27. For this dataset, the general observations are almost the same as stated for NASA and USC datasets. Here, MMRE increases when the training set's size decreases. Again, the decrease is low for methods such as RBF and DT.

For this dataset, in MLP, RBF and SVM methods, the initial application of PCA causes a decrease in MMRE. However, it causes an increase in error for DT method. The case here is alike to the case for NASA dataset where DT outperforms the rest of the methods. DT method is satisfying as an estimator since PRED is greater than 75.

When COCOMO II's own parametric model that has been explained in Section 2.1.2 is used, the mdMRE is 1860.32, MMRE is 3368.15 and PRED(30) is 8.7. Considering this, the results of all the methods are better than the results for COCOMO II parametric model. This shows that the learning system outperforms the parametric model. This result might be due to the fact that current software practices include tools that automatically produce most of the code itself. This approach makes the size bigger when the effort is low considering size. Although there is a metric called TOOL in the COCOMO II model, it fails to adopt this practice and does not reflect the consequences to the estimations. Thus, the strength of our model can be seen more apparently in this case.

Table 6.27. Summary of experimental results for SDR dataset (%)

SDR (# of training)	Without PCA		With PCA	
	MMRE (std. dev.) mdMRE (std. dev.)	PRED (std. dev.)	MMRE (std. dev.) mdMRE (std. dev.)	PRED (std. dev.)
MLP (53)	121.67 (134.25) 63.21(37.72)	41.67 (14.81)	116.16 (106.89) 68.20(37.23)	45 (16.31)
MLP (42)	150.78 (110.38) 86.27 (33)	25.71 (12.78)	123.27 (84.76) 80.35(29.15)	27.86 (15.70)
RBF (53)	22.60(12.12) 18.99(16.26)	72.14 (11.79)	21.01(11.41) 18.67(14.97)	73.33 (25.59)
RBF (42)	23.47 (7.34) 16.12 (8.28)	65 (25.31)	23.44 (6.54) 18.30 (8.03)	68.57 (13.59)
DT (53)	14.55 (9.42) 11.06 (5.94)	90 (15.67)	27.58(14.18) 13.57 (5.75)	73.57 (11.61)
DT (42)	20.41 (9.48) 12.78 (8.85)	74.29 (15.09)	36.42(37.52) 14.42(12.10)	70 (18.42)
SVM (53)	187.26 (174.64) 97.29(57.07)	22.86 (15.65)	60.45 (58.06) 25.71(13.49)	56.43 (12.67)
SVM (42)	264.30 (279.57) 168.99(95.93)	18.33 (20.16)	61.02 (50.63) 35.49(21.86)	53.33 (31.34)

For all the datasets, the standard deviation values for all metrics are also given. This is done to see how much the values change between minimum and maximum value. It is observable here that the best performing methods also produce the least standard deviation which suggests the stability of the system.

7. CONCLUSIONS

In this research, we have conducted experiments on software effort estimation using various machine learning methods on three different datasets one of which is a native dataset. We have obtained acceptable results in the context of software effort estimation, especially considering the results of the parametric models are used. As a result, we have seen that parametric models are insufficient for software effort estimation and the problem must be handled using an evolving system rather than a static one. The metrics based on those models might be changed or calibrated or new metrics can be added to improve the model.

Models based on limited data sets such as COCOMO and COCOMO II tend to incorporate the particular characteristics of the data. This results in a high degree of accuracy for similar projects, but restricts the application of the model. As we have seen in the experiments, COCOMO model is good at estimating effort for the two datasets: USC and NASA. It is not surprising that for these two datasets the results of the model is good. However, COCOMO II model fails to perform good on the SDR dataset which we have collected. We think that this result is due to the fact that most of the software have been developed using some readily available software tools such as Visual Studio which provides automatically created code. This way a considerable percentage of the code is created by the program rather than the coders. Although there is a related metric called TOOL in both models COCOMO and COCOMO II, the effect of this metric does not suffice. This is why considering the huge values of size, the results for effort is also very high causing a huge inaccuracy in SDR dataset, because most of the projects in that dataset are developed within 2-3 years and they include heavy usage of tools. This clearly shows that using a learning system instead of a parametric model is highly preferable since the parametric model may fail to give good estimates because it's based on restricted data in this domain.

The main objective of this research is making an analysis of software effort estimation to overcome problems related to budget and schedule overruns. Our proposed solution not only brings another point of view into software engineering effort estimation

but it also tries to improve the software effort estimation process. If practitioners have more accurate estimations then they would be able to manage risks and prevent any losses that may have occurred due to these risks. Academically, our model brings two different practices in two different areas: software engineering and machine learning. The combination of different models of cost estimation and different machine learning methods has brought out good results and yet another perspective.

For future work, the learning system might be calibrated or different methods can be incorporated to test the system further. Another future work can be developing the dataset that we have collected to obtain a dataset that reflects the software development characteristics of the software development organizations in Turkey and come up with a better model to fulfill the needs of the industry in this context. This is necessary, because there is an obvious lack of work and knowledge in industry about this particular problem. Another future work is working in a different feature set for software effort estimation so that every instance in the dataset is identifiable. For example, for Turkish software industry more metrics might be needed or some metrics might be discarded or changed to reflect a particular property of a software project.

REFERENCES

1. Boehm B., C. Abts and S. Chulani, "Software Development Cost Estimation Approaches – A Survey", University of Southern California, 1998.
2. Briand L. C., K. E. Emam, D. Surmann, I. Wieczorek and K. D. Maxwell, "An Assessment and Comparison of Common Software Estimation Modeling Techniques" *Proceedings of the International Software Engineering*, pp. 313-322, 1999.
3. Mendes E., C. Lokan, R. Harrison and C. Triggs, "A Replicated Comparison of Cross-company and Within-company Effort Estimation Models using the ISBSG Database", *Proceedings IEEE International Software Metrics Symposium*, pp. 36-46, 2005.
4. Moses, J., "Learning how to improve effort estimation in Small Software Development Companies", *24th International Computer Software and Applications Conference*, pp. 522-527, 2000.
5. Heemstra, F. J., "Software Cost Estimation Models", *IEEE Software*, pp. 286-297, September 1990.
6. Boehm, B. W., *Software Engineering Economics*, Prentice Hall, 1981.
7. Boehm, B. W., "Software Engineering Economics", *IEEE Transactions on Software Engineering*, pp. 4-21, January 1984.
8. Rubin, H., "ESTIMACS", *IEEE Software*, pp. 27-29, 1983.
9. L. Wu, *The Comparison of the Software Cost Estimating Methods*, <http://www.computing.dcu.ie/~renaat/ca421/LWu1.html>, 1997.
10. University of Southern California, *USC COCOMO Reference Manual*, USC, 1994.
11. Quality Development Costs and Schedule, <http://www.ii.metu.edu.tr/~ion545/demo/section1.3.html>, METU, 2006.
12. COCOMO, <http://www.webopedia.com/TERM/C/COCOMO.html>, 2006.

13. Shepperd, M. and M. Schofield, "Estimating Software Project Effort Using Analogies", *IEEE Transactions on Software Engineering*, Vol. 23, No. 12, pp. 736-743, November 1997.
14. Cowderoy, A. J. C. and J. O. Jenkins, "Cost estimation by analogy as a good management practice", *Proceeding in Software Engineering*, Liverpool: IEE/BCS, 1988.
15. Kutlubay O, M. Balman, D. Gül and A. Bener, *A Machine Learning Based Model For Software Defect Prediction*, M.S. Thesis, Boğaziçi University, Computer Engineering Department, 2004.
16. University of Southern California, *USC COCOMO II Reference Manual*, USC, 1998.
17. Banker, R., H. Chang and C. Kemerer, "Evidence of Economies of Scale in Software Development", *Information and Software Technology*, Vol. 36, No. 5, pp. 275-282, 1994.
18. Clark, B., S. Chulani and B. Boehm, "Calibrating the COCOMO II Post Architecture Model", *International Conference on Software Engineering*, pp. 477-480, 1998.
19. Chulani, S., "Incorporating Bayesian Analysis to Improve the Accuracy of Quality Model Extension", *Ph.D. Qualifying Exam Report*, University of Southern California, 1998.
20. Mitchell, T. M., *Machine Learning*, McGraw-Hill and MIT Press, 1997.
21. Alpaydın, E., *Introduction to Machine Learning*, MIT Press, 2004.
22. Artificial Neural Networks, *Wikipedia*, http://en.wikipedia.org/wiki/Artificial_Neural_Network, 2006.
23. Sutton, R. S. and A. G. Barto, *Reinforcement Learning : An Introduction*, MIT Press, 1998.
24. Hornik, K., M. Stinchcombe and H. White, "Multilayer feedforward networks are universal approximators", *Neural Networks*, Vol. 2, pp. 359-366, 1989.

25. Haykin, S., *Neural Networks - A Comprehensive Foundation*, Prentice-Hall, Englewood Cliffs, 1998.
26. Honkela, A., *Nonlinear Switching State-Space Models*, <http://www.cis.hut.fi/ahonkela/dippa/node41.html>, 2006.
27. Buhmann, M. D., *Radial Basis Functions*, Cambridge University Press, 2004.
28. Radial Basis Functions, <http://www.nd.com/models/rbf.htm>, 2006.
29. Li, S. T., *Robust Radial Basis Function Networks*, Institute of Information Management National Cheng Kung University, 2003.
30. K-means, http://www.resample.com/xlminer/help/kMClst/KMClust_intro.htm, 2006.
31. Regression trees, http://www.resample.com/xlminer/help/rtree/rtree_intro.htm, 2006.
32. Decision trees, http://en.wikipedia.org/wiki/Decision_tree#Decision_tree_advantages, 2006.
33. Duda, R. O., P. E. Hart and D. Storck, *Pattern Classification*, Wiley publishing, 2000.
34. Winston, P., *Learning by Building Identification Trees*, Addison-Wesley Publishing Company, 1992.
35. Decision Trees Tutorial, <http://www.decisiontrees.net/tutorial>, 2006.
36. Witten, I., H. and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann Publishers, San Francisco, CA, 2000.
37. Wittig, G., "Estimating Software Development Effort with Connectionist Models", Working Paper Series 33/95, Monash University, 1995.
38. Kay, A., Artificial Neural Networks, <http://www.computerworld.com/softwaretopics/software/appdev/story/0,10801,57545,00.html>, 2006.
39. Data Clustering, http://en.wikipedia.org/wiki/Cluster_analysis, 2006.

40. Quinlan, J. R., *C4.5: Programs for Machine Learning*, San Mateo, Morgan Kaufmann, 1993.
41. Breiman, L., J. H. Friedman, R. A. Olshen and C. J. Stone, *Classification and Regression Trees*, Wadsworth International Group, 1984.
42. University of Southern California, *USC COCOMO II Model Manual*, 1998.
43. Support Vector Machines, http://en.wikipedia.org/wiki/Support_vector_machine, 2006.
44. Platt, J., *Fast Training of Support Vector Machines using Sequential Minimal Optimization*, Advances in Kernel Methods - Support Vector Learning, MIT Press, 1998.
45. Vapnik, V., S. Golowich and A. Smola, "Support vector method for function approximation, regression estimation, and signal processing", *Advances in Neural Information Processing Systems*, No. 9, Cambridge, MA, MIT Press, 1997.
46. Principal Components Analysis, *Wikipedia*, http://en.wikipedia.org/wiki/Principal_components_analysis, 2006.
47. Principal Components Analysis, <http://www.eng.man.ac.uk/mech/merg/Research/datafusion.org.uk/pca.html>, 2006.
48. Basili, V., F. McGarry, R. Pajerski and M. 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*, Orlando, Florida, pp. 69-79, 2002.
49. Potts, C., *Software Engineering Research Revisited*, IEEE Software, September 1993.
50. COCOMO II Cost Estimation Questionnaire, <http://sunset.usc.edu/research/COCOMOII/Docs/cform22.pdf>, 2006.
51. Software Cost Estimation : Metrics and Models, <http://sern.ucalgary.ca/courses/seng/621/W98/johnsonk/cost.htm#Original%20COCOMO>, 2006.

52. Fenton, N. E. and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, International Thomson Computer Press, 1997.
53. Support Vector Machine Toolbox Version 2.0, <http://www.isis.ecs.soton.ac.uk/isisystems/kernel/svm.zip>, 1998.
54. Pontil, M. and A. Verri, “Direct Aspect-based 3-D Object Recognition”, *Proceedings of the International Conference on Image Analysis and Processing*, Vol. 2, pp. 300-307, 1997.
55. Joachims, T., “Text Categorization with Support Vector Machines: Learning with Many Relevant Features”, *Proceedings of the European Conference on Machine Learning*, pp. 137-142, 1998.
56. Stensrud, E., T. Foss, B. Kitchenham and I. Myrtveit, “A Further Empirical Investigation of the Relationship Between MRE and Project Size”, *Empirical Software Engineering*, Vol. 8, pp. 139-161, June 2003.
57. Conte, S. D., H. E. Dunsmore and V. Y. Shen, *Software Engineering Metrics and Models*, Benjamin-Cummings Publishing, Menlo Park CA, 1986.
58. University of South California Software Engineering Research, <http://sunset.usc.edu/research/>, 2006.