CLASSIFICATION BASED COST ESTIMATION MODEL FOR EMBEDDED SOFTWARE

by

Ayşe Bakır

B.S., Computer Engineering, Gebze Institute of Technology, 2006

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

Graduate Program in Computer Engineering

Boğaziçi University

2008

CLASSIFICATION BASED COST ESTIMATION MODEL FOR EMBEDDED SOFTWARE

APPROVED BY:

Asist. Prof. Ayşe B. Bener …….
(Thesis Supervisor)

Prof. Oğuz Tosun …….

Assoc. Prof. Necati Aras …….

DATE OF APPROVAL: 16.06.2008

ACKNOWLEDGEMENTS

I would like to thank those people who have helped me while I was doing this research. First of all, I would like to thank my supervisor, Dr. Ayşe Başar Bener. With her knowledge, experience, and support, I was able to finish this thesis and gain a valuable insight of academic research discipline.

Secondly, I would like to thank to Prof. Oğuz Tosun and Assoc. Prof. Necati Aras who accepted to be in my thesis committee.

I would also like to thank to Burak Turhan, who is a PhD student and also a member of the Software Engineering Research Laboratory, for his patience to answer all my questions and willingness to share his knowledge with me.

I am grateful to my parents who are always ready to give all their love and support to me. My brother Mustafa and my sisters Emine and Emel were always there to encourage me whenever I need help. This thesis would not have been possible without their support.

Finally, I would like to thank all of the anonymous people working in NASA, PROMISE, and ISBSG, and the graduate students of computer engineering department who contributed to the constitution of the datasets which I have used in this thesis. I am grateful to TUBITAK (The Scientific & Technological Research Council of Turkey) for providing me scholarship during my graduate study and supported this research under grant no EEEAG 108E014 and ARDEB 1001. I would also like to thank to Boğaziçi University Research Fund for supporting this research in part under grant number BAP 06HA104

ABSTRACT

CLASSIFICATION BASED COST ESTIMATION MODEL FOR EMBEDDED SOFTWARE

Software development companies face many problems in order to complete their projects successfully: on time, within budget and with no defects. Scheduling and resource allocation directly affect financial performance and market position of a software company. The challenge is how the project managers will decide what level of skill set, for how long and at what cost they will need for a given project. Therefore, practitioners increasingly need intelligent oracles to help them make these decisions. These oracles can be defined as the learning based prediction models for effort and cost estimation. Such predictive models prevent project managers to take wrong decisions due to inaccurate estimations.

In this research, we focus on building learning based predictive models for cost estimation in embedded systems domain. Our proposed model tackles the prediction accuracy problem from both data usage and model development aspects. Firstly, we focus on data usage and investigate what kind of and how much training data should be used for software cost estimation in embedded systems. Secondly, we focus on model development and propose a new cost estimation model for embedded software. We believe that our results would assist the software managers while selecting the data to train the cost models and allocating available resources more efficiently by using more accurate analysis.

In literature, there has not been any study that focused on embedded software cost estimation yet. We aim to fill in this gap for embedded systems domain. In addition, we present a new cost estimation model which achieves high accuracy rates. In our empirical work, we utilize a wide range of machine learning techniques in order to make our results be independent from the techniques used. Also, we used datasets from three different sources in order to be able to generalize our results under different set-ups.

ÖZET

GÖMÜLÜ YAZILIMLAR İÇİN SINIFLANDIRMA BAZLI MALİYET TAHMİNİ MODELİ

Yazılım geliştirme firmaları projelerini başarılı bir şekilde, yani zamanında, bütçeyi aşmadan ve hatasız bitirmek için birçok zorlukla karşılaşmaktadırlar. Planlama ve kaynakların paylaşımı bir yazılım şirketinin finansal performansını ve pazardaki konumunu doğrudan etkilemektedir. Asıl zorluk; proje yöneticilerinin, verilen bir proje için ne kadar süreyle ve maliyetle hangi seviyede yetenek gerekeceğine nasıl karar vereceğidir. Bu yüzden proje yöneticileri bu kararları vermeye yardımcı olacak akıllı yol göstericilere giderek atan bir şekilde ihtiyaç duymaktadırlar. Bu yol göstericiler, işgücü ve maliyet tahmini için öğrenme tabanlı tahmin modelleri olarak tanımlanabilir. Bu tip modeller, proje yöneticilerinin hatalı tahminlere bağlı olarak yanlış kararlar vermelerini önlemektedir.

Bu araştırmada, gömülü yazılım alanında maliyet tahmini için öğrenme tabanlı tahmin modelleri geliştirmeye odaklanıyoruz. Modelimiz, tahmin doğruluğu problemini, hem verinin kullanımı hem de model geliştirilmesi açılarından ele almaktadır. İlk olarak, verinin kullanımına odaklanıp, gömülü sistemlerde yazılım maliyeti tahmini için ne tür ve ne kadar veri kullanılması gerektiğini araştırıyoruz. İkinci olarak, model geliştirilmesi üzerine odaklanıp, gömülü yazılımlar için yeni bir maliyet tahmini modeli öneriyoruz. Sonuçlarımızın, maliyet modellerini eğitecek verinin seçimi ve eldeki kaynakların daha etkin bir şekilde paylaşımı konularında proje yöneticilerine destek olacağına inanıyoruz.

Yazında, gömülü yazılımların maliyet tahminiyle ilgili herhangi bir çalışma bulunmamaktadır. Biz, bu boşluğu gömülü sistemler alanında doldurmayı amaçlıyoruz. Buna ek olarak, yüksek doğruluk oranı elde eden yeni bir maliyet tahmini modeli sunuyoruz. Deneylerimizde, sonuçlarımızın kullanılan modelden bağımsız olmasını sağlamak için, geniş çapta yapay öğrenme tekniklerinden faydalanıyoruz. Bir de, deneylerimizin değişik durumlarda da aynı sonuçları verdiğini göstermek ve dolayısıyla bu sonuçları genelleştirebilmek için üç farklı kaynaktan veri setleri kullanıyoruz.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

COCOMO	Constructive Cost Model
COCLAM	Cost Classification Model
DT	Decision Tree
EM	Effort Multiplier
GS	Gaussian Smoother
ISBSG	International Software Benchmarking Standards Group
KNN	K-Nearest Neighbor
KS	Kernel Smoother
LD	Linear Discrimination
LOC	Lines of Code
LR	Linear Regression
MdMRE	Median Magnitude of Relative Error
MLP	Multi-layer Perceptron
MMRE	Mean Magnitude of Relative Error
MR	Misclassification Rate
MRE	Magnitude of Relative Error
NASA	National Aeronautics and Space Administration
PCA	Principal Components Analysis
PM	Person Month
PRED(25)	Prediction at level 25
REVIC	Revised Embedded Cocomo
RQ1	Research Question 1
RQ2	Research Question 2
S1	Setup 1
S2	Setup 2
S3	Setup 3
SoftLab	Software Research Laboratory
SVR	Support Vector Regression
UFP	Unadjusted Function Points
WoCAD	Within or Cross Application Domain

1. INTRODUCTION

Cost estimation is one of the critical steps of the software development life cycle as it assists project managers when they make important decisions such as bidding for a new project, planning and resource allocation. As Leung and Fan states, underestimation results in approving projects that would exceed their budgets whereas overestimation results in wasting of resources [5]. Therefore, modeling accurate and robust software cost estimators is still a key challenge for managing successful software projects. A wide variety of methods have been proposed in order to achieve both high accuracy and robustness. Most of these methods reveal from statistics and machine learning fields [12].

In cost estimation, there are two main ways that can be followed in order to improve prediction accuracy. The first one is to understand the structure of the cost data better and the second one is to develop a better predictor. In this research, we focus on both methods and propose our solutions for each of them in the domain of embedded software.

In order to understand the data better, previous studies mostly concentrate on the comparison of either *cross- vs. within-company* data or *cross- vs. within-business domain* data [13, 14]. *Cross-company data* is gathered from different companies and is mostly preferred by the companies which have no historical data of completed software projects or are newly established. On the other hand, *within-company data* is gathered from one company and is used when the company has enough historical data to train and validate its estimation models on. When we look at previous studies on this subject, it was found out that estimation models that are developed on within-company data (a.k.a. within-company model) give either similar or better results than those developed on cross-company data (a.k.a. cross-company model) [13]. In none of the studies, cross-company models were significantly better than within-company models. As an extension of the research on cross- vs. within company data, there was an effort to examine the homogeneity of cost data from *business domain* point of view [14]. In contrast to cross- vs. within-company data, both *cross-* and *within-business domain* data are collected from different companies. Within-business domain data is gathered from different companies with the same operation area (such as finance, medical, telecommunications, etc.) whereas cross-business domain data is

gathered from different companies with different operation areas. Previous studies that focus on business-domain data also found out that within-business domain data are better for developing estimation models on than cross-business domain data [14]. In another study that focuses on *data homogeneity*, the hypothesis of more homogenous analogues for a project produce a more reliable cost estimate was tested. As a result, a large variation in reliability was observed between high and low homogeneity level projects [15].

In contrast to previous studies, in this research, we treat data homogeneity from *application domain* point of view. By application domain we mean domains such as embedded, real-time, desktop application, etc. and we focus on embedded domain in this research. Our aim is to find out what type of data should be preferred for embedded software cost estimation. In addition, we aim to find out how much training data should be used for the estimation process.

There are some reasons that play important role for choosing embedded software domain to focus on in our research. In the last decade, development of multimedia and wireless applications on mobile devices cause an increase on attention for embedded systems [16]. These systems were formerly implemented in hardwired uni-processor architectures whereas they now contain programmable-multiprocessors which mean high complexity and cost. As a natural consequence of this, main challenge in embedded software development becomes the cost-effective implementations that meet performance, functional, timing, and physical requirements. Previous research on embedded systems mostly concentrates on power cost analysis [17-20]. There is not any study that specifically focuses on the software development cost analysis of embedded systems. This is the main reason why we have chosen embedded software domain as the application domain to focus on.

As we have stated formerly, the second way to improve prediction accuracy in cost estimation is to develop a better predictor. In literature, there are two types of studies that focus on modeling a cost predictor: those produce point estimates and those produce interval estimates. In point estimation, cost estimation is perceived as a regression problem and a single value of effort is tried to be estimated. On the other hand, in interval estimation, cost estimation is perceived partially as a classification problem and either the

confidence intervals for point estimates or the posterior probabilities of predefined effort intervals is tried to be estimated. In literature, there are far more studies that produce point estimates than those produce interval estimates and mostly used methods are regression, various machine learning techniques, estimation by analogy, and neural networks [21-26]. For interval estimation, regression-based methods such as ordinal regression and multinomial logistic regression have been used generally [27-32].

In practice, for project staffing and scheduling, managers generally use point estimates [28]. However, they may easily make wrong decisions if they rely only on these point estimates. In their study, Stamelos and Angelis states that producing interval estimates is safer since it can be used for the prediction of the current project's cost in terms of the completed ones [30]. In addition, while bidding for a new project, an interval estimate can be easily converted to a point estimate by evaluating the values that fall into the same interval.

In this research, in order to build a better predictor, we propose an approach that is based on interval estimation and handles cost estimation problem in a different manner. In contrast to previous studies that use regression-based methods, our approach integrates different methods from machine learning. In addition, we provide point estimation results of our proposed approach for comparison purposes.

1.1. Motivation

The motivation of this research is two folds. Firstly, in order to understand cost data better, we introduce a new aspect of data homogeneity, application domain, and investigate what type of data to use for embedded software cost estimation. In addition, we investigate the effect of training data size on the prediction performance.

Secondly, in order to model a better predictor, we develop a cost estimation model by using interval estimation as a tool. The proposed approach integrates different machine learning techniques and it is validated on various datasets. We also obtain point estimates with different estimators in order to show the efficiency of our approach.

Our main contributions are to find out what kind of and how much training data should be used for software cost estimation in embedded systems and to present a new cost model which we think will yield higher accuracy than those proposed in literature.

1.2. Outline

Until this part, an introduction to cost estimation is made and the motivations of this research are given. In the following chapter, Chapter 2, some background information about embedded software and cost estimation techniques from literature are given.

The main problems that we search solutions for are stated in Chapter 3 and the proposed solutions for each of them are given in Chapter 4.

The details of the methodology used, the machine learning methods applied, and the experiments conducted are presented in Chapter 5. Also, some information about the datasets and the performance measures that are used in the experiments is given in this chapter.

In Chapter 6, the experimental results are given for each research question and are compared with other studies. The results of the statistical analyses made are also given in this chapter.

Finally, the main contributions made, the discussion of the results and future work are given in Chapter 7.

2. BACKGROUND

2.1. Overview

This thesis is a combination of embedded software and cost estimation each of which is a wide research area. Thus, some background information about either concept should be given before presenting a new approach. Firstly, embedded software is explained in the following section. Then, the cost estimation methods that are present in the literature are given. From these methods, only the ones that are related to our research will be explained in details.

2.2. Embedded Software

Embedded systems are the architectures that are designed to execute a particular function while meeting performance, cost, size, weight, and power requirements [1]. These systems are heterogeneous systems that consist of dedicated hardware (such as programmable processors) and software which is called *embedded software*. Embedded software's principal role is to provide the interaction with the physical world. We can see examples of embedded software on machines such as cars, airplanes, etc [4].

For embedded systems, there are three types of cost metrics: hardware cost metrics, interface cost metrics, and software cost metrics [1, 2]. Hardware cost metrics include hardware area and hardware execution time. Interface cost metrics include communication time which is the cost of transporting data from one processing unit to another. Software cost metrics include:

- Number of System Lines of Code (SLOC or LOC)
- Effort required for developing software (PM)
- Defects introduced per LOC
- Time required for repairing each software defect

In our proposed approach, we try to predict the effort metric by using the LOC metric with high accuracy. The third and fourth metrics are used for defect prediction which is out of the scope of our research.

Today, different system-level design and test methodologies are used for embedded systems. These methodologies can be mainly categorized as [3]:

- *Standard COTS (Commercial Off-The-Shelf)—minimum hardware cost.* This methodology tries to minimize only the hardware costs.
- *COTS with cost modeling—minimum system cost.* This methodology tries to minimize both hardware and software costs while maximizing profits.
- *Full custom—custom hardware and software.* This approach aims to develop custom hardware and software and is usually schedule and cost intensive.
- *COTS with cost modeling and virtual prototyping.* By using virtual prototyping, this approach uses hardware and software models to facilitate integration and test.
- *COTS with cost modeling, virtual prototyping, advanced top-down programming, and reuse.* This methodology is the most advanced one among all. It is likely to achieve the best cost objectives.

Although standard COTS methodology has many disadvantages (such as use of written requirements, long prototyping times, high design costs, and lack of systematic reuse), it is the mostly used one among all for high-performance embedded systems. To minimize overall system costs, it attempts to minimize only hardware costs forgetting an important issue of software prototyping. In contrast, the fifth methodology is the best approach that does not only guarantee successful software development but also state the urgent need for developing accurate cost models. This research aims to meet this need and find a specific solution for embedded software.

In literature, embedded systems are studied in terms of power cost analysis [17-20]. The first study on this subject introduces a new approach to model processors' behaviors based on Colored Petri Net and to estimate power consumption distribution [17]. Another study describes the first systematic attempt to model power cost of embedded software is given in [19]. It presents a power analysis technique that could help verifying if a design

meets its specified power constraints. In [18], a methodology and a software tool which evaluates system development, fabrication, and testing costs were introduced. Lastly, Zotos et al. introduce the notion of energy complexity of an algorithm for estimating the required energy consumption [20]. To the best of our knowledge, there is not any study that focuses on modeling/estimation of software development costs of embedded systems. In this paper, our main aim is to fill in this gap from both data and model perspectives.

For estimating the cost of embedded software, in practice, parametric models like COCOMO, REVIC and some commercial tools are used [34, 35, and 36]. Especially, the embedded mode of COCOMO and REVIC are the most applicable tools for the software costing of embedded systems [11, 33]. COCOMO uses nominal effort equations that are derived from labor effort and related to the size of the software system. REVIC is US Air Force's embedded-mode REVised Intermediate Cocomo cost model. It includes many functions such as those for calculating development effort, development time, annual maintenance effort, multi-objective cost function, effort adjustment and utilization, processors and memory capacity. However, these models are designed rather generic and should be calibrated before used by other companies [13].

2.3. Software Cost Estimation

2.3.1. Motivation

The cost of software development is mostly due to human effort, so, software cost estimation is defined as the process of predicting the effort required to develop a software system [5]. The estimates are given in terms of person-months (PM) and used in various phases in software development lifecycle such as budgeting, risk analysis, project planning, and, software improvement investment analysis [12].

Research on cost estimation began in 1966 with the study presented in [6] which then led to more robust models in the 1970s such as SLIM [7], Checkpoint [8], PRICE-S [9], SEER [10], and COCOMO [11]. Since then, there have been two challenges that all the researchers agree on. The first one is that as the software size and complexity grows, the estimation accuracy decreases. The second one is that software development methods are

changing continuously and a single parametric model can not achieve high accuracies for all software domains. In literature, a large number of research have been made on software cost estimation each introducing a new method to overcome these challenges.

2.3.2. Cost Estimation Techniques

We can group cost estimation techniques into two main groups according to the estimates they produce; those produce *point estimates* and those produce *interval estimates*.

Existing cost estimation techniques that produce point estimates can be classified into six categories which are model-based techniques, expertise-based techniques, learning oriented techniques, dynamics-based techniques, regression-based techniques, and composite techniques [12].

In *model-based techniques*, cost is estimated by using some specified parameters and functions which are determined by theoretical and experimental analysis. In literature, there are a number of cost models and mostly used ones are Putnam's Software LIfe-cycle Model (SLIM), Checkpoint, PRICE-S, ESTIMACS, SEER-SEM, COCOMO, and COCOMO II [11, 12, 37]. As these models are calibrated based on past experience, their main disadvantage is their unsuitability to unprecedented situations. *Expertise-based techniques* are used when historical data is not available. In this method, project managers are responsible for estimating the costs based on their previous experience and the prevailing industry norms. Mostly used expertise-based techniques are the Delphi technique and the Work Breakdown Structure [12]. However, these techniques have two main disadvantages. The process can not be tested and it is difficult to find highly experienced estimators for every new project. *Learning oriented techniques* include case studies and neural networks [12]. In case studies, project planners try to learn heuristics by extrapolation from past examples whereas in neural networks automated models are built for this purpose. In contrast to previous techniques, *dynamics-based techniques* assume that cost factors are dynamic rather than static over software development lifecycle and are mostly based upon the system dynamics approach to modeling originated nearly forty years ago [12]. In literature, *regression-based techniques* are the mostly used ones due to

their simplicity and wide acceptance. These techniques are used together with model-based techniques and include ordinary least squares (OLS) and robust regression [12]. Each of the techniques mentioned earlier has its own disadvantages. Thus, in order to have a better estimation method, *composite techniques* are used by combining two or more techniques. One of the most important composite techniques is the Bayesian approach which uses Bayes' theorem to produce the posterior distribution for the model parameters [12].

Cost estimation studies that produce interval estimates can be grouped into two categories. The first group consists of studies those calculate *confidence intervals* for point estimates and the second group includes studies those calculate the *probabilities of predefined intervals*. In the first case, interval estimates are generated during the estimation process whereas in the second case the intervals are predefined before the estimation process.

In the first group, the first study that empirically evaluates effort prediction interval models is [29]. It compares the effort prediction intervals derived from a bootstrap-based model with the prediction intervals derived from regression-based effort estimation models. But, there was a confusion of terms in that study and a critique is then made by Jorgensen in [38] to clarify the ambiguity. In another study, an interval estimation method based on expert judgment is proposed [32]. Some statistical simulation techniques to calculate confidence intervals for project portfolios are also given in [30].

Two important studies for the second group are [28] in which ordinal regression is used to model the probabilities of both effort and productivity intervals and [27] which uses multinomial logistic regression for modeling productivity intervals. Both studies also include point estimate results of the proposed models. Also, in [31], predefined intervals of productivity are used in a Bayesian Belief Network to support expert opinion. An empirical comparison of the models that produce point estimates and that produce predefined interval estimates is given in [39]. A study that focuses on predefined interval uncertainty is given in [38].

Up to now, we wanted to give a general outline of existing cost estimation techniques. From these techniques, only the ones that are used in our research will be explained in details in the following subsections.

2.3.2.1. Model Based Techniques. The COCOMO (COnstructive COst Model, a.k.a. COCOMO 81) cost and schedule estimation model was developed by Barry Boehm in 1981 for estimating effort and schedule of software projects [11, 40]. *Effort* is the number of months one person would need to develop a given project and *schedule* is the actual number of months needed for development by a properly staffed full-time development team. The model parameters and formulas are constructed by the analysis of 63 aerospace software projects that belongs to NASA [51].

In COCOMO, effort and schedule are highly related to the development difficulty of the software. For this purpose, three software development modes are defined [41]:

- *Organic mode* is used for projects which are small in size and require little innovation. The constraints upon development are not tight and development environment is stable.
- *Semi-detached mode* is used for projects which has medium size and complexity. The constraints upon development are mostly rigid, but, there is still some flexibility.
- *Embedded mode* is used for large projects with tightly defined hardware and software constraints. They require great amount of innovation and complex hardware/customer interfaces.

COCOMO consists of three simplicity levels which are Basic COCOMO, Intermediate COCOMO, and Detailed COCOMO. Basic COCOMO is a static model that estimates effort by using a single equation that takes project size as an input:

$$PM_{nominal} = A \times (Size)^{B} \tag{2.1}$$

where *PM* is the effort given in units of Person-Month, *Size* is the estimated number of delivered thousands Lines Of Code (KLOC) for the project, and *A* and *B* are the coefficients. Schedule is estimated by using the formula below:

$$TIME_{dev} = C \times (PM)^{D} \tag{2.2}$$

where *PM* is the effort calculated by using equation (2.1), *TIME* is the schedule in months, and *C* and *D* are the coefficients. For the three development modes, the values of the coefficients are given in Table 2.1.

Table 2.1. Coefficients for Basic COCOMO and Intermediate COCOMO [41]

	Basic COCOMO				Intermediate COCOMO	
Mode	A	B	C	D	A	B
Organic	2.4	1.05	2.5	0.38	3.2	1.05
Semi-detached	3.0	1.12	2.5	0.35	3.0	1.12
Embedded	3.6	1.20	2.5	0.32	2.8	1.20

Basic COCOMO is useful for rough estimates of software costs, however, it can not adapt to the changes in requirements or project attributes. Thus, Intermediate COCOMO is developed by taking into account not only the project size but also a set of cost drivers. These drivers which also affect the development productivity are grouped into four categories [40]:

- *Product attributes* are the constraints and requirements specified for the project to be developed.
- *Computer attributes* are the limitations caused by the hardware and operating system being used for the project.
- *Personnel attributes* are the skills that the development personnel have such as professional ability, programming ability, experience with the development environment and familiarity with the application domain.
- *Project attributes* are the conditions under which project is developed.

Each category includes three or four cost drivers (a.k.a. *effort multipliers*) each of which is associated with a rating on a six-point scale such as *very low*, *low*, *nominal*, *high*, *very high,* and *extra high*. Each rating has a corresponding real number based on the degree to which the factor can influence productivity. The cost drivers which are 15 in total and corresponding ratings are given in Table 2.2.

Table 2.2. Cost drivers for COCOMO 81 [5]

Cost Factors	Description	Rating					
		Very low	Low	Nominal	High	Very high	Extra high
Product Attributes							
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
Computer Attributes							
TIME	execution time constraint	-	-	1.00	1.11	1.30	1.66
TURN	computer turnaround time	-	0.87	1.00	1.07	1.15	-
STOR	main storage constraint	-	-	1.00	1.06	1.21	-
VIRT	virtual machine volatility	-	0.87	1.00	1.15	1.30	-
Personnel Attributes							
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	-	-
Project Attributes							
TOOL	use of 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	-
MODP	modern programming	1.24.	1.10	1.00	0.91	0.82	-

These cost drivers are also used while calculating the estimated effort and schedule in Intermediate COCOMO:

$$PM_{total} = A \times (Size)^{B} \times \prod_{i=1}^{15} EM_i \tag{2.3}$$

where PM_{total} is the effort given in units of Person-Month, *Size* is the estimated number of delivered thousands Lines Of Code (KLOC) for the project, A and B are the coefficients, and EM_i is the numerical value of the rating of the i^{th} effort multiplier. For the three development modes, the values of the coefficients are given in Table 2.1. Schedule is estimated by using the effort calculated in formula (2.3) in the equation (2.2) with the same coefficient values.

Detailed COCOMO includes all characteristics of the Intermediate COCOMO with an assessment of the cost drivers' effect on each step (analysis, design, etc.) of the software development process.

As the software development technology moved from mainframe and batch processing to desktop applications and object oriented approaches, COCOMO became insufficient and COCOMO II was published in the Annals of Software Engineering in 1995 [37, 40]. COCOMO II does not explicitly partition projects by development modes (organic, semidetached, and embedded) as in COCOMO 81. Instead, it uses five scale factors which are selected based on the rationale that they are a significant source of exponential variation on a project's effort or productivity variation. The scale factors and corresponding values for ratings are shown in Table 2.3.

Table 2.3. Scale factors for COCOMO II [40]

Scale Factors	Description	Rating					
		Very low	Low	Nominal	High	Very high	Extra high
PREC	precedentedness	4.05	3.24	2.43	1.62	0.81	0.00
FLEX	development flexibility	6.07	4.86	3.64	2.43	1.21	0.00
RESL	architecture/risk resolution	4.22	3.38	2.53	1.69	0.84	0.00
TEAM	team cohesion	4.94	3.95	2.97	1.98	0.99	0.00
PMAT	process maturity	4.54	3.64	2.73	1.82	0.91	0.00

COCOMO II has three different sub models which are namely *Application Composition*, *Early Design* and *Post-Architecture*. The *Application Composition* model is used to estimate effort and schedule on systems that use modern computer aided tools for rapid integration of the application components. Typical examples of these systems are graphic user interface (GUI) builders, database or object managers, middleware for distributed processing or transaction processing, etc [12]. The model is based on Object Points which are a count of the screens, reports and third-generation-language modules developed in the application, each weighted by a three-level (simple, medium, difficult) complexity factor [42]. The equation for estimating effort is:

$$PM = \frac{NOP}{PROD} \tag{2.4}$$

where *PM* is the effort given in units of Person-Month, *NOP* is the New Object Points determined, and *PROD* is a productivity rate estimated from an average of developer's experience and Integrated Computer Aided Software Environment (ICASE) maturity [40].

The *Early Design* model is used to make prior estimates of cost and duration before the entire architecture of the project is not determined. It uses 7 cost drivers (a.k.a effort multipliers) and new equations in which sizing metric is thousand lines of code (KSLOC) converted from unadjusted function points [43, 44]. The cost drivers are obtained by combining the Post-Architecture model cost drivers from Table 2.4 and are *personnel capability (PERS)*, *product reliability and complexity (RCPX)*, *required reuse (RUSE)*, *platform difficulty (PDIF)*, *personnel experience (PREX)*, *facilities (FCIL)*, and *schedule (SCED)*. More details on these drivers can be found in [40]. The effort equation for Early Design model is given below:

$$PM = A \times [Size']^{B} \times \prod_{i=1}^{7} EM_i + PM_M \tag{2.5}$$

where *PM* is the effort given in units of Person-Month, PM_M is the maintenance effort, *Size* is the estimated number of delivered thousands Lines Of Code (KLOC) for the project, *A* and *B* are the coefficients, and EM_i is the numerical value of the rating of the i^{th} effort multiplier. Detailed information about how to calculate these parameters can be found in [40].

The *Post-Architecture* model is a detailed extension of the Early-Design model as it involves estimates for the entire development life-cycle. It has been calibrated to a database of 161 projects collected from commercial, aerospace, government and non-profit organizations. It is used when the software life-cycle architecture has been developed and detailed information about the project is available. The model uses source lines of code as the sizing measure and includes 17 effort multipliers which are grouped into four categories (product factors, platform factors, personnel factors, project factors) as shown in

Table 2.4. These four categories are parallel to those of COCOMO 81. Also, most of the effort multipliers are similar to those of COCOMO 81 except the removed and added ones. *DOCU*, *RUSE*, *PVOL*, *PEXP*, *LTEX*, *PCON*, *SITE* are the new multipliers in COCOMO II and *VIRT*, *TURN*, *VEXP*, *LEXP*, *MODP* are the multipliers removed from COCOMO 81. The ratings for those multipliers retained in COCOMO II were changed in order to reflect more up-to-date calibrations.

Table 2.4. Cost drivers for COCOMO II [40]

Cost Factors	Description	Rating					
		Very low	Low	Nominal	High	Very high	Extra high
Product Attributes							
RELY	required software reliability	0.75	0.88	1.00	1.15	1.39	-
DATA	database size	-	0.93	1.00	1.09	1.19	-
CPLX	product complexity	0.75	0.88	1.00	1.15	1.30	1.66
RUSE	required reusability	-	0.91	1.00	1.14	1.29	1.49
DOCU	documentation match to lifecyle needs	0.89	0.95	1.00	1.06	1.13	-
Platform Attributes							
TIME	execution time constraint	-	-	1.00	1.11	1.31	1.67
STOR	main storage constraint	-	-	1.00	1.06	1.21	1.57
PVOL	platform volatility	-	0.87	1.00	1.15	1.30	-
Personnel Attributes							
ACAP	analyst capability	1.50	1.22	1.00	0.83	0.67	-
PCAP	programmer capability	1.37	1.16	1.00	0.87	0.74	-
PCON	personnel continuity	1.24	1.10	1.00	0.92	0.84	-
AEXP	applications experience	1.22	1.10	1.00	0.89	0.81	-
PEXP	platform experience	1.25	1.12	1.00	0.88	0.81	-
LTEX	language and tool experience	1.22	1.10	1.00	0.91	0.84	-
Project Attributes							
TOOL	use of software tools	1.24	1.12	1.00	0.86	0.72	-
SITE	multisite development	1.25	1.10	1.00	0.92	0.84	0.78
SCED	required development schedule	1.29	1.10	1.00	1.00	1.00	-

The equation for estimating effort is the same as (2.5) except that *i* changes from *1* to *17* since we have 17 multipliers [40]. For schedule estimation, in COCOMO II, the same equation is used for all three models:

$$Scedule = \left[A \times PM^{(0.33+0.2\times(B-1.01))}\right] \times \frac{SCED\%}{100} \quad \text{and} \quad B = 1.01 + 0.01\sum_{j=1}^{5} SF_j \qquad (2.6)$$

where *PM* is the estimated effort given in units of Person-Month, *A* is the constant (provisionally set to 3.0), *SCED* is the compression/expansion percentage in the SCED effort multiplier, are the coefficients, and SF_j is the numerical value of the rating of the j^{th} scale factor.

2.3.2.2. Learning Oriented Techniques. As mentioned earlier, learning oriented techniques include case studies (a.k.a. analogy based estimation) and neural networks [12].

In *case studies*, past projects' information are used for the estimation process of the new projects. By extrapolating from the past projects' attributes, project managers estimate the cost of the current projects. For this purpose, during the development phase of previous projects, all the information about the environmental conditions and constraints, the decisions made, and the results obtained must be kept. Projects managers use this information and find out the projects whose attributes are similar to the new projects'. The underlying assumption is that similar projects have similar attributes, and also, similar costs. Thus, the source of the past projects plays an important role. If available, the projects should be chosen inside the organization as they are likely to be more similar to the organization's future projects. If there is not enough project data, other organizations' data can also be used as long as they are well-documented [12].

In one of the previous studies [48], it is shown that analogy-based cost estimation outperforms traditional algorithmic methods. The study also describes an automated tool (known as ANGEL) in order to estimate the effort of a new project. The tool supports the collection, storage and identification of the projects that are most similar to the new one. A disadvantage of estimation by analogy is that it requires a considerable amount of computation.

Neural networks is the mostly used cost estimation technique after regression [49]. Neural networks are algorithmic structures that can be trained by using the historical data. According to the differences between actual and estimated values, they can adjust their parameter values to reduce the total error. The network has a number of parameters to be specified: the number of layers of neurons, the number of neurons within each layer, the functions between the nodes and the specific training algorithm to be used. After the network is built, it must be trained by using the data of past projects with their actual cost values. When the total error is small enough, training can be stopped and new projects' data can be fed into the network to estimate their corresponding cost values. Here, the specification of error value is important, because, if it is chosen a too small, the network is over trained; else, the network is not trained enough.

2.3.2.3. Regression Based Techniques. Previous work on software cost estimation mostly produces point estimates by using regression methods [21, 22]. According to Boehm [12], the two most popular regression methods are *ordinary least square regression (OLS)* and *robust regression*.

Ordinary least square regression [21] is a general linear model that uses least squares. The model parameters are calculated by minimizing the least squares error which is the difference between the desired output and the predicted output. Thus, all inputs have an equivalent influence on the output. According to [12], OLS is ideal when:

- a lot of data are available
- no data items are missing
- there are no outliers
- the predictor variables are not correlated
- the predictor variables have an easy interpretation when used in the model
- the regressors are either all continuous or all discrete variables

Robust regression [22] is the improved version of OLS. It overcomes the problem of outliers observed in cost data. Outliers may be caused by the extreme values when the projects are very different from each other. There are many techniques that can be considered as robust regression. The first one is a technique based on the Least Median

Squares method. The second one is a technique that automatically gets rid of outliers and can be used only when there is a sufficient number of input data. Most of the existing parametric cost models mentioned before (COCOMO II, SLIM, Checkpoint etc.) use regression based techniques because they are simple but effective [12].

3. PROBLEM STATEMENT

Since software development has become an important part of the product development lifecycle, the costs occur during the development phase have also become serious matters for companies to think about. Effort is one of these costs which have to be estimated carefully as it directly affects the acceptance of new software projects. Today, in software industry, the commonly used effort estimation method is expert judgment in which experts make estimations based on their past experiences and capabilities. However, this method is generally unable to produce reliable estimates because new projects are rarely the same as the old ones. In addition, the hardware and software tools, the staff, and the techniques being used change very fast. Thus, instead of expert judgment, scientific methods that produce accurate and robust estimates should be used for software cost estimation.

As we have stated before, in software cost estimation, prediction accuracy can be improved by two ways. The first one is to understand data better and the second one is to develop a better predictor. In literature, there are various studies that focus on either subject. For the first subject, latest studies investigate the relationship between the homogeneity of cost data and the prediction accuracy. Nearly all of them conclude that as the homogeneity increases, the prediction accuracy also increases. In this research, we bring a new perspective to data homogeneity and focus on application domain. The domain on which a software project is developed plays a very important role as it affects the whole software development lifecycle. Especially, embedded domain is promising in today's world where embedded systems are so popular. Thus, we focus on embedded software domain and investigate what type of data should be used for embedded software cost estimation. Also, we try to find out how much training data should be used for the estimation process.

For the second subject of building better predictors, there are a large number of studies in literature and each one introduces a new method for predicting software development effort. Mostly methods from statistics and machine learning areas are used in these studies. However, there is not yet a single parametric model that can achieve high

accuracies for all software domains. Thus, in this research, we aim to develop a cost estimation model for embedded software domain in order to obtain high accuracy.

In this context, we have the following research questions:

RQ1. How can we understand the embedded software data better?

a) What type of training data should we use for embedded software cost estimation?

b) How much training data should we use for embedded software cost estimation?

RQ2. How can we develop a better predictor for embedded software cost estimation?

4. PROPOSED SOLUTION

In this research, two new approaches are developed to solve the research questions given previously. For the first research question (RQ1), in order to understand embedded software data better, we introduce the concepts of *within-application domain* data and *cross-application domain* data. Within-application domain data is gathered from different companies that operate on the same application domain (which we chose as embedded domain in this research) whereas cross-application domain data is gathered from different companies from different application domains. We propose an approach for RQ1 (given in Figure 4.1) that compares different machine learning methods that are trained on *within-application domain* data with those trained on *cross-application domain* data in order to find out what type of data to use for embedded software cost estimation. We call this approach as WoCAD (Within or Cross Application Domain) in the rest of the thesis. These methods are *linear regression (LR), kernel smoother (KS), support vector regression (SVR), multi-layer perceptron (MLP), k-nearest neighbors (K-NN),* and *voting* which will be explained in details in Chapter 5.2.2.

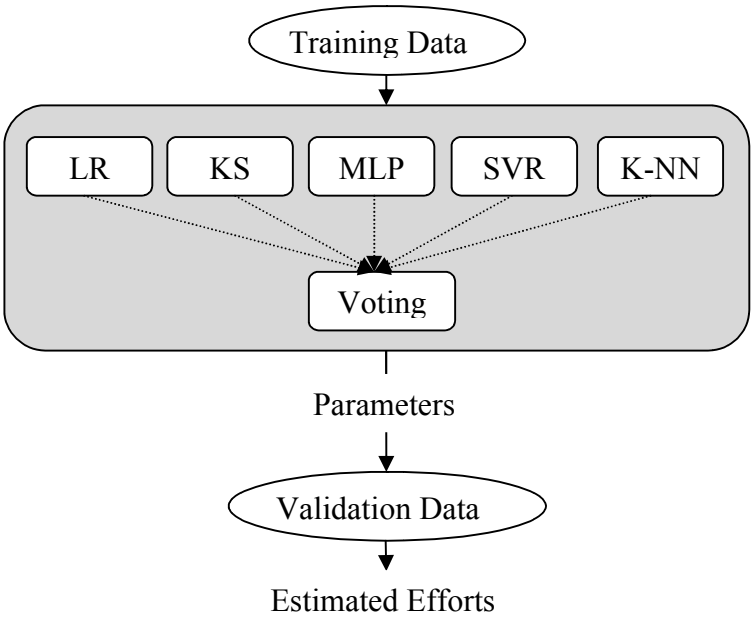

Figure 4.1. WoCAD model for RQ1

In WoCAD, firstly, five of the methods are trained on the training data to learn the model parameters and validated on the validation data to estimate the effort values for each validation sample. Secondly, the results of these methods are combined with voting algorithm to produce one more effort estimate. Also, in order to find out how much data should be used as training data in the estimation process, the same model in Figure 4.1 is used with training data either containing all samples or a subset of samples. This process will be explained in Chapter 5.3.1 in details.

In this research, for the second research question (RQ2), we aim to develop a cost model for embedded software by converting cost estimation into a pure classification problem [71, 73, and 74]. We call this model as COCLAM (COst CLAssification Model) in the rest of the thesis. Up to now, cost estimation has been perceived as a regression problem and most of the studies produce point estimates. Although there are a few studies that produce interval estimates and convert cost estimation partially into a classification problem, it cannot be said that cost estimation is fully converted into a classification problem. Our approach integrates classification methods with cluster analysis and contains three main steps which are [71]:

1. grouping similar projects together by cluster analysis
2. determining the effort intervals for each cluster and specifying the effort classes
3. classifying new projects into one of the effort classes by classification methods

In step (1), Leader Cluster algorithm is used in order to determine the clusters and cluster means. Here, each cluster corresponds to a group of software projects that have similar attributes. In step (2), in order to calculate the effort intervals that correspond to each cluster, firstly, the minimum and maximum of the efforts of the projects that reside in the same cluster are found. Secondly, these minimum and maximum values are chosen as the upper and lower bounds of the interval that will represent that cluster. Finally, each cluster is given a class label (such as 1, 2 …) which will be used to classify the new projects into. In step (3), a new project's class is estimated by using the class-labeled data generated in the previous step. The resulting class shows the effort interval that contains the new project's effort value. Three different classification algorithms are used for this step; one is parametric (Linear Discrimination) and the others are non-parametric (K-

Nearest Neighbor and Decision Tree). These three algorithms are chosen to show how our approach performs with the algorithms of different complexities. Linear Discrimination is the simplest one whereas Decision Tree is the most complex one. K-Nearest Neighbor has a moderate complexity depending on the training set size. These clustering and classification methods will be explained in Chapter 5.2.3 in details.

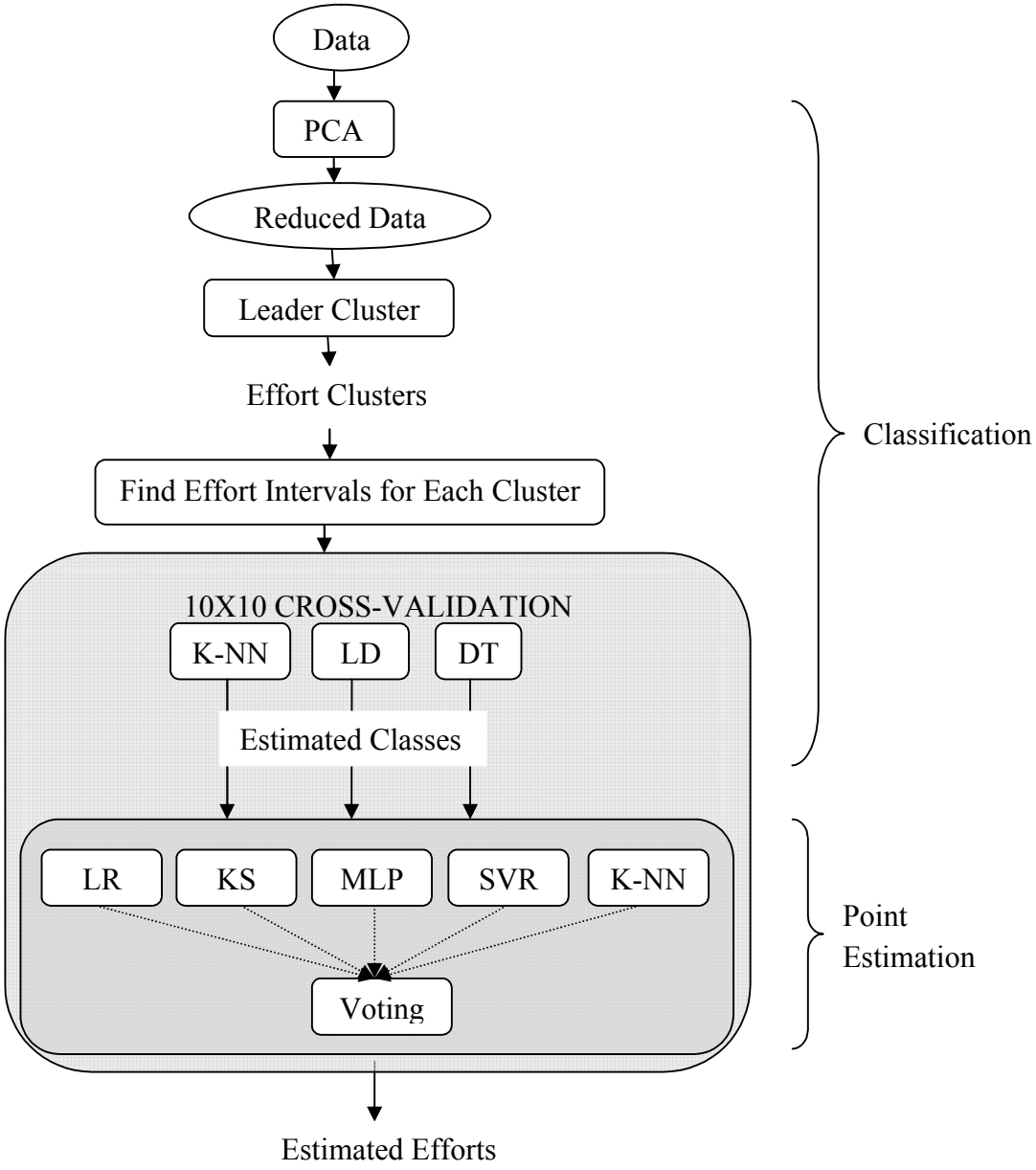

Figure 4.2. COCLAM model for RQ2

Our proposed model for effort classification is given in Figure 4.2. Firstly, in order to extract relevant features, Principal Component Analysis (PCA) is applied to whole input

data except the *Effort* value [45]. Secondly, clustering algorithm is applied to the normalized data to obtain project groups. Thirdly, each cluster is assigned a class label and the effort intervals for each of them are determined. Then, the effort data containing projects with corresponding class labels is given to each of the classification algorithms mentioned above. Since there are not separate training and validation sets, the classification process is done in a 10x10 cross-validation loop. In the cross-validation loop, data is 10 times shuffled into random order and then divided into 10 bins. Training set is built from nine of the bins, and the remaining bin is used as validation set. Classification algorithms are firstly trained on the training set and then validated on the validation set. Additionally, just after the classification step, point estimates are calculated for each classification method in order to use for comparison with the results obtained in WoCAD. For this process, the six estimators used in WoCAD (*linear regression, kernel smoother, support vector regression, multi-layer perceptron, k-nearest neighbors,* and *voting*) are applied to the projects that belong to the class that is found by the classification algorithms [73].

5. METHODOLOGY

5.1. Dataset Description

In this research, datasets from three main sources are used. These sources are PROMISE Data Repository, Bogazici University Software Engineering Research Laboratory repository (SoftLab), and International Software Benchmarking Standards Group (ISBSG) repository.

Our first data source, PROMISE Data Repository, is a public repository that contains data about software projects from NASA and different universities located in United States [51]. These data are collected by the software engineering researchers from all round the world and have been used in a large number of studies. The repository holds data in five main areas which are namely defect prediction, effort prediction, model-based software engineering, text mining, and general. The number of datasets increases as the new ones are added from day to day. For cost estimation, there are currently ten datasets most of which are collected based on either COCOMO I or COCOMO II cost estimation models mentioned in Chapter 2.

For COCOMO-based datasets, there are 15 nominal attributes, one numerical attribute, and a target attribute which is the effort value in person-month. The other numerical attribute is the size of the project in source lines of code (LOC) unit. Remaining 15 attributes are the COCOMO I cost drivers described in Chapter 2 (*RELY, DATA, CPLX, TIME, STOR, VIRT, TURN, ACAP, AEXP, PCAP, VEXP, LEXP, MODP, TOOL, LEXP*). For COCOMO II-based datasets, there are 22 nominal attributes, one numerical size attribute (LOC), and the target effort attribute as in COCOMO I. 17 of the nominal attributes are the effort multipliers (*RELY, DATA, CPLX, RUSE, DOCU, TIME, STOR, PVOL, ACAP, AEXP, PCAP, PEXP, LTEX, PCON, TOOL, SCED, SITE*) and the remaining 5 nominal attributes are the scale factors (*PREC, FLEX, RESL, TEAM, PMAT*) defined in Chapter 2.

Our second data source, Bogazici University Software Engineering Research

Laboratory repository (SoftLab), contains data about software projects that belongs to various companies in Turkey [50]. These data are collected by the master students of computer engineering department of Bogazici University for their Software Quality Modeling course. While collecting the data, COCOMO II Data Collection Questionnaire that was prepared by Barry Boehm is used [53]. The questionnaire collects information at both project and component levels. In the project level, application type and development activity information are collected. Component level explains size, cost and component cost drivers. Interviews with project managers in various software companies are made and the questionnaires for their one/more completed/ongoing projects are filled. Project level data is collected once for each project and component level data is collected for each component of a project if it has more than one component.

Our third and last data source, International Software Benchmarking Standards Group (ISBSG), is a non-profit organization that maintains a software project management database from a variety of organizations [52]. In the following subsections, the datasets used in our research will be explained in details.

5.1.1. PROMISE Data Repository

Four of the cost estimation datasets in PROMISE Data Repository are used in this research. These datasets are *coc81*, *cocomonasa_v1*, *desharnais_1_1* and *nasa93.*

Coc81 was firstly published in Boehm's book [11] in 1981 and then converted to artff file format in 2004. The data was collected by University of South California from different software organizations according to COCOMO I model format. It includes 63 projects' data each consisting of 17 attributes in total (15 for the effort multipliers, one for LOC, and one for actual development effort). In 2008, the development mode information (embedded, organic, or semidetached) for each project was added and the name of this dataset is changed to *coc81_1_1*. As we focus on embedded software domain in this research, by choosing only the projects with embedded development mode, we form a new dataset which we call *coc81_e* in our experiments. This new dataset contains 28 projects with 17 COCOMO I attributes.

Nasa93 contains data from different centers for NASA projects from the years between 1971 and 1987. It was collected according to COCOMO I model format by Jairus Hihn who is the manager of SQIP Measurement & Benchmarking Elements at JPL NASA. The projects belong to applications such as avionics, monitor control, batch data processing, communications, data capture, launch processing, mission planning, science, simulation, and utility. The dataset includes 93 projects each consisting of 24 attributes in total (15 for the effort multipliers, 7 for describing the projects, one for LOC, and one for actual development effort). Since this dataset includes the development mode attribute for each project, we decided to use this dataset for our research. Again by choosing only the projects with embedded development mode, we form a new dataset which we call *nasa93_e* in our experiments. This new dataset contains 21 projects with 17 COCOMO I attributes.

Cocomonasa_v1 contains data about NASA projects that belong to three different centers and the period between the 1980s and 1990s. It was collected according to COCOMO I model format by Jairus Hihn who is the manager of SQIP Measurement & Benchmarking Elements at JPL NASA. It includes data about 60 projects that belong to applications such as sequencing, avionics, and mission planning. Each project consists of 17 attributes in total (15 for the effort multipliers, one for LOC, and one for actual development effort).

Desharnais is collected by J. Desharnais when he was a master student in Montreal University, Canada, in 1989 [1]. It contains 77 projects with 12 numeric attributes which are different than COCOMO format. These attributes are *project id*, *team experience* (measured in years), *manager experience* (measured in years), *year end*, *length*, *actual effort value*, *count of basic logical transactions in the system*, *number of entities in the systems data model*, *pointsadjusted*, *envergure*, *pointsnonadjust*, and *language.* Language is a categorical attribute which can have only the values 1, 2 or 3.

5.1.2. SoftLab Data Repository

Three cost estimation datasets from SoftLab Data Repository are used in this research: *cost_data*, *sdr*, and *sdr2.* All of these datasets are collected from different

companies in Turkey in COCOMO II model format. This means that, for each project, there are 22 nominal attributes, one numerical size attribute, and the actual effort value. 17 of the nominal attributes are the effort multipliers and the remaining 5 are the scale factors. *Cost_data* has 24 projects, *sdr* has 25 projects, and *sdr2* has 40 projects in total.

As we have only two within-domain datasets that contain embedded projects' data (*coc81_e* and *nasa93_e*), we want to form a third one by combining both of them. We call this new dataset which contains 49 projects in total as *both* in our experiments for WoCAD.

In order to form cross-domain datasets to be used in WoCAD, firstly, all the datasets except *nasa93_e*, coc81_e, and *both* (i.e. *cocomonasa_v1*, *cost_data1, sdr, sdr2*) are merged into one large dataset, which is called *others1*. Secondly, one more cross-domain dataset is formed by adding the non-embedded projects in *coc81* and *nasa93* to *others1*. This second dataset is called as *others2*. While merging the datasets, only the common attributes, which are 15 in total, are selected from each dataset. The reason for different number of features is that they are collected in either COCOMO or COCOMO II model format.

5.1.3. ISBSG Data Repository

In this research, we use ISBSG Release 10 which contains 4106 projects each with around 106 attributes. Nevertheless, there are many projects with missing values in the dataset, thus, only a portion of the dataset is used generally by the researchers. These attributes can be classified into 15 categories as shown in Table 5.1. It is necessary to preprocess the dataset as inconsistencies and missing values can confuse the cost estimation process. The preprocessing of ISBSG dataset in order to form an embedded dataset includes three steps:

1. The projects whose *Application Type* attribute is equal to "*Embedded*" are chosen. There are only 21 projects that provide this condition. All of these projects are either new development or enhancement projects belonging to different organization types such as consumer goods, defense, telecommunication, retail,

Table 5.1. ISBSG Release 10 attributes

Category	Attributes
Rating	Data Quality Rating, Unadjusted Function Point Rating
Sizing	Count Approach, Functional Size, Adjusted Function Points, Value Adjustment Factor
Effort	Normalized Level 1 Work Effort, Normalized Work Effort, Summary Work Effort
Productivity	Normalized Level 1 Productivity Delivery Rate, Normalized Productivity Delivery Rate, Pre 2002 Productivity Delivery Rate
Schedule	Project Elapsed Time, Project Inactive Time, Implementation Date, Project Activity Scope, Effort Breakdown, Effort Unphased
Quality	Defects Delivered, Total Defects Delivered
Grouping Attributes	Development Type, Organization Type, Business Area Type, Application Type, Package Customization, Degree of Customization
Architecture	Architecture, Client Server, Client Roles, Server Roles, Type of Server, Client/Server Description, Web development
Documents & Techniques	Plan Documents, Specification Documents, Specification Techniques, Design Documents, Design Techniques, Build Products, Build Activity, Test Documents, Test Activity, Implementation Documents, Implementation Activity, Development Techniques, Functional Sizing Techniques, FP Standard, FP Standards All, Reference Table Approach
Project Attributes	Development Platform, Language Type, Primary Programming Language, 1st Hardware, 1st Operating System, 1st Language, 1st Database System, 1st Component Server, 1st Web Server, 1st Message Server, 1st Debugging Tool, 1st Other Platform, 2nd Hardware, 2nd Operating system, 2nd Language, 2nd Database System, 2nd Component Server, 2nd Web Server, 2nd Message Server, 2nd Other Platform, CASE Tool Used, Used Methodology, How Methodology Acquired
Product Attributes	User Base-Business Units, User Base-Locations, User Base-Concurrent Users, Intended Market
Effort Attributes	Recording Method, Resource Level, Max Team Size, Average Team Size, Ratio of Project Work Effort to Non-Project Activity, Percentage of Uncollected Work Effort
Size Attributes	Input count / Output count / Enquiry count / File count / Interface Count, Added count / Changed count / Deleted count, COSMIC Entry / COSMIC Exit / COSMIC Read / COSMIC Write
Size Other Than FSM	Lines of Code, Lines of Code Not Statements, Other Size Units
Software Age	Year of Project

manufacturing, and computer & software. They are developed between the years 2002 and 2005.

2. The projects whose *Size* or *Effort* attributes are empty are removed and the number of projects decreases to 17.
3. The sizes of these 17 projects are measured in mostly Lines of Code (LOC). There are only 3 projects whose size is measured in Unadjusted Function Points (UFP). In order to standardize the size unit, the sizes of these three projects are converted into LOC by using the Table 11 in COCOMO Model Definition Manual [40].
4. Unnecessary attributes and the attributes having empty value for all projects are removed. The remaining attributes which are 8 in total are *summary work effort*, *project elapsed time*, *development platform*, *language type*, *primary programming language*, *1st operating system*, *1st language*, and *Lines of Code*.
5. Among these 8 attributes, those which are categorical (*development platform*, *language type*, *primary programming language*, *1st operating system*, *1st language*) are converted into numerical format by assigning a number (1, 2, 3 …) for each category.

After these steps, the remaining 17 projects each with 8 attributes are used to form an embedded (within-domain) dataset and this dataset is called as *ISBSG_e*. As this dataset has different attributes than the previous ones, (those taken from PROMISE and SoftLab repositories) we need to form a new cross-domain dataset to use for comparing to *ISBSG_e* in WoCAD. For this reason, a second preprocessing of ISBSG dataset is done with the following steps:

1. The projects whose *Application Type* attribute is different from "*Embedded*" are chosen. There are totally 4084 projects that provide this condition. All of these projects are either new development, or enhancement, or re-development projects belonging to different organization types such as banking, telecommunication, media, software products, aerospace, insurance, and health care. They are developed between the years 2001 and 2005.
2. The projects whose *Size* or *Effort* attributes are empty are removed and the number of projects decreases to 3405.

3. As this cross-domain dataset will be used as training data for *ISBSG_e*, it should have the same 8 attributes (*summary work effort*, *project elapsed time*, *development platform*, *language type*, *primary programming language*, *1st operating system*, *1st language*, and *Lines of Code*). Thus, the projects with empty values for these attributes are removed and 1820 projects are remained.
4. The sizes of the projects in *ISBSG_e* are measured in Lines of Code (LOC). Thus, the projects whose size unit is other than LOC are removed and 104 projects are remained.
5. As done in the last step of preprocessing for ISBSG_e, the attributes which are categorical (*development platform*, *language type*, *primary programming language*, *1st operating system*, *1st language*) are converted into numerical format by assigning a number (1, 2, 3 …) for each category.

After these steps, the remaining 104 projects each with 8 attributes are used to form a cross-domain dataset and this dataset is called as *ISBSG*.

By taking into account all of the data repositories used, we have four within-domain datasets and three cross-domain datasets for our experiments. An overview of the contents of these datasets is given in Table 5.2.

Table 5.2. Datasets used for WoCAD

Datasets			# of Projects	Total # of Projects
Domain	Name	Content		
Within-Domain (Embedded)	coc81_e	embedded projects from coc81	28	28
	nasa93_e	embedded projects from nasa93	21	21
	both	coc81_e nasa93_e	49	49
	ISBSG_e	embedded projects from ISBSG	17	17
Cross-Domain	others1	cocomonasa_v1 cost_data sdr sdr2	60 24 25 40	149
	others2	others1 remaining ones from coc81 remaining ones from nasa93	149 35 72	256
	ISBSG	remaining ones from ISBSG	104	104

For COCLAM, the same four within-domain datasets are used in the experiments since we propose a new approach for embedded software cost estimation. Furthermore, in order to make our classification approach proposed for RQ2 externally valid, the remaining cost datasets from PROMISE and SoftLab repositories (*cocomonasa_v1, desharnais, cost_data, sdr, sdr2*) are used in COCLAM. An overview of these datasets is given in Table 5.3.

Table 5.3. Datasets used in further experiments for COCLAM

Datasets	Number of Projects
cocomonasa_v1	60
desharnais	77
cost_data	24
sdr	25
sdr2	40

5.2 Methods Used

In the last decade, learning-based models have produced promising results in cost estimation domain. There are a number of studies that focus on applying machine learning techniques to cost data in literature [23-26]. In this research, we have used a wide range of machine learning methods in order to implement our proposed approach for each research question stated in Chapter 4. Six of these methods are used for effort estimation by regression, one method is used for clustering, and three methods are used for classification. In addition to these methods, a dimension reduction algorithm (Principal Components Analysis) is used to extract the relevant features in cost data. All of these methods will be explained in details in the following subsections.

5.2.1. Principal Components Analysis

The main purpose of Principal Components Analysis (PCA) is to reduce the dataset dimensions so that the dataset can be efficiently used without losing much data [45]. In this research, reducing the data and representing the variance in the data in fewer dimensions can be quite important for obtaining results.

Variance is the key factor in PCA. The objective is to maximize the variance, thus, covariance of the training set is used in the construction of eigenvectors. After eigenvectors and eigenvalues of the covariance matrix are calculated, they are sorted in accordance with the eigenvalues in descending order. Then, their part on the explanation of the variance is calculated as a *proportion of variance* value. If these values, starting from the eigenvector with the highest eigenvalue, exceed a predefined threshold, then that point is defined as a cut-off point for dimension reduction. In our research, we took the most common threshold value in machine learning as *0.90* [45]. The eigenvectors with a proportion of variance values below this threshold are considered as *principal components*. As a final step, a projection operation is applied to principal components. The same principal components are applied on both training set and validation set. These transformed sets represent the reduced version of their original forms.

5.2.2. Methods Used for WoCAD

Six different machine learning methods are used in WoCAD for estimating the effort value for embedded software projects. These methods are linear regression, kernel smoother, support vector regression, multi-layer perceptron, k-nearest neighbors, and voting algorithm.

These methods all have different complexity levels. *Linear regression* is the simplest of all as it tries to fit a linear model to the input data. *Kernel smoother* is also simple but more powerful than linear regression as the importance of data instances is not the same for all inputs. *Support Vector Regression (SVR)* is the most complex one among the regression algorithms used in our research. Since it is a non-linear algorithm, it moves the inputs to a higher dimensional space where the output becomes linearly separable. Another complex model we used in our research is the *multi-layer perceptron (MLP)* with back-propagation, where the cost is estimated as a nonlinear combination of input attributes. We also include an unsupervised method, *k-nearest neighbor algorithm* (*K-NN*), that provides a baseline for comparing the similarity of projects by using the Euclidean distance.

Finally, voting algorithm is used in order to combine estimated effort values from five estimators. The estimated cost value of each estimator is given a particular weight and this weighted sum of each estimate is taken as the final estimation.

5.2.2.1. Linear Regression. Despite its simplicity, Linear Regression has been widely used for cost estimation in previous studies [29, 46-48]. In regression [45], given a set of samples, $X\ (x^t,r^t)$, the aim is to estimate the numeric value of a new sample, x. In linear regression, a linear model is fitted to the given samples as shown below:

$$r^t = w_0 + w_1 x^t \tag{5.1}$$

The numeric output r is written as a linear function which is the weighted sum of several input variables, $x_1,...,x_d$, and noise. When there is more than one variable, the linear model becomes like that:

$$\begin{aligned} r^t &= g(x^t \mid w_0, w_1, ..., w_{1d}) + \in \\ &= w_0 + w_1 x_1^{\ t} + w_2 x_2^{\ t} + ... + w_d x_d^{\ t} + \in \end{aligned} \tag{5.2}$$

When we assume □ to be normal with zero mean and constant variance, maximizing the likelihood is equivalent to minimizing the sum of squared errors:

$$E(w_0, w_1, ...w_d \mid X) = \frac{1}{2}\sum_t (r^t - w_0 - w_1 x_1^{\ t} - ... - w_d x_d^{\ t})^2 \tag{5.3}$$

For example, if we have the following vectors and matrix:

$$X\begin{bmatrix} 1 & x_1^{\ 1} & x_1^{\ 2} & . & x_d^{\ 1} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ 1 & x_1^{\ N} & x_2^{\ N} & . & x_d^{\ N} \end{bmatrix}, w = \begin{bmatrix} w_0 \\ w_1 \\ . \\ . \\ w_d \end{bmatrix}, r = \begin{bmatrix} r^1 \\ r^2 \\ . \\ . \\ r^N \end{bmatrix} \tag{5.4}$$

Then, the normal equations can be written as:

$$X^T X w = X^T r \tag{5.5}$$

And we can solve for the parameters as:

$$w = (X^T X)^{-1} X^T r \tag{5.6}$$

Then, by putting *w* into the equation (5.1), the output for new sample is calculated.

5.2.2.2. Kernel Smoother. The second regression algorithm used in this research is *Kernel Smoother* [45]. It is similar to linear regression except that the importance of data instances is not the same for all inputs. In this method, to get a smooth estimate, a smooth weight function called *kernel function* is used. It gives less weight to distant samples by dividing the data into bins and fitting the kernel function to the data that are in the same bin.

Let x be our new sample and x^t be the training samples, r^t be the output values for training samples, and h be the window width. *Kernel smoother* is given as below:

$$\hat{g}(x) = \frac{\sum_t K\left(\frac{x - x^t}{h}\right) r^t}{\sum_t K\left(\frac{x - x^t}{h}\right)} \tag{5.7}$$

The most popular kernel function and also the one we used in this research is the *Gaussian* kernel:

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{u^2}{2}\right] \tag{5.8}$$

When h is small, each training instance has a large effect in a small region and no effect on distant points. When h is larger, there is more overlap of the kernels and we get a smoother estimate. In this research, we choose h to be equal to 0.5.

5.2.2.3. Support Vector Regression. Support Vector Machines (SVM) can be used for both classification and regression [55]. The main aim is to construct a hyper plane that lies as close to as many data points as possible and minimize total error.

For example, if we have a training data in the form of (x_i,y_i) where y is the target attribute and x is the input attribute, the linear cost function used in SVR is as follows:

$$f(x) = \langle w, x \rangle + b \tag{5.9}$$

Cost function is formed in (5.9) such that the data points fit with □ deviation from target points. It calculates dot product of weight and data points in $\langle w, x \rangle$. If we convert the problem into an optimization problem, our main aim becomes to minimize (5.10) subject to the condition formulized in (5.11).

$$\min \frac{1}{2}\|w\|^2 \tag{5.10}$$

$$\begin{aligned} y_i - \langle w, x_i \rangle - b &\leq \in \\ \langle w, x_i \rangle + b - y_i &\leq \in \end{aligned} \tag{5.11}$$

If some errors are tolerated, *slack variables* are added into (5.10) and (5.11). Besides, there are various loss functions that determine the amount up to which deviations (□) are tolerated. In our case, we have used *ε-insensitive* loss function. The key idea after that point is to construct *Lagrange function* from the optimization conditions. All explanations for these formulas can be found in [55] in details.

There are a number of kernel functions that are available to be used in support vector machine technique. These functions are namely *polynomial function*, *radial basis function*, *spline function*, and *exponential radial basis function*. We have tried all of them in order to decide which one fits best to our cost data. Then, we decided to use *spline function* with ε-insensitive value for loss function is equal to *0.5*. Algorithm can run with several other combinations of these variables; however we have done such assumptions for our research.

For most formulas and variables explained above, Steve Gunn's SVR algorithm [56] is used by modifying its functions for our problem. Also, we have used the values assigned to parameters of each kernel function and the loss function from [56] to see the best fit with these curves.

5.2.2.4. Multi-Layer Perceptron. Multi-layer Perceptron (MLP) is a well-known neural network structure. In this structure, as shown in Figure 5.1, there are three layers which are namely *input layer*, *hidden layer,* and *output layer* [58]. To train this neural network, *back-propagation* algorithm is used.

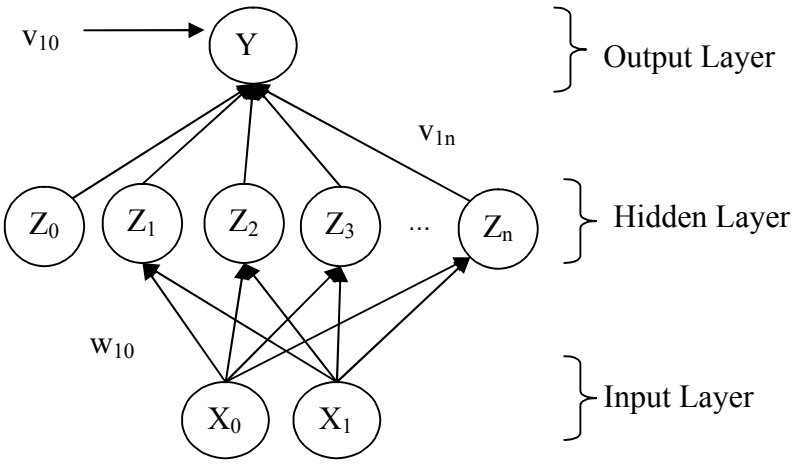

Figure 5.1. Multi-layer Perceptron

Input data is given to the input layer and the outputs of input layer are estimated by a *nonlinear sigmoid function* of weighted sum of its inputs. These outputs of the first layer are given to the hidden layer as the secondary inputs. Estimation is performed here as the weighted sum of these secondary inputs, but, in this case, only weighted sum is estimated without using the sigmoid function. This output represents the estimated effort value for our data. For each training sample, the estimated outputs are fed back to the previous layer and the weight values are updated in accordance with them. This process is called *back-propagation*. This weight values are updated after each training example; and over the time, the network learns the convenient weight parameters representing the model best. The magnitudes of the updates depend on the inputs which mean that as inputs gets greater; then, the update for its weights gets greater.

There are a number of parameters that need to be specified while constructing the network structure. The most important ones are:

- the number of hidden neurons
- the functions for the hidden neurons and the output neuron
- the number of iterations or the convergence rule
- the learning rates for each layer

In this research, Phil Brierley's neural network with back-propagation code is used [57]. In our implementation, we use 10 hidden neurons with *tanh* function, an output neuron with *linear function*, *learning rate* that is equal to 0.9 for input layer and 0.09 for the hidden layer. Learning stops when the error is smaller than 0.001 or total number of iterations (which is 500 in our research) is reached.

5.2.2.5. K-Nearest Neighbor. K-Nearest Neighbor (K-NN) algorithm is a simple but also powerful learning method that is particularly suited for classification problems [45]. It can also be used for regression after some changes made. K-NN assumes that all samples correspond to points in the *n-dimensional Euclidean space*, R^n, where n is the number of the sample attributes.

In K-NN regression, when we are given a set of training samples and a new sample, firstly, using a measure of similarity, the similarity of the new sample to the given ones are computed. Mostly, *Euclidean distance* is used as the similarity measure. That is, given:

$$\begin{aligned} x &= (x_1,...,x_n) \\ y &= (y_1,...,y_n) \end{aligned} \tag{5.12}$$

the distance $d(x,y)$ between x and y is as follows:

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{5.13}$$

After the distances are calculated from the new sample to each training sample, *k* nearest neighbors who have minimum distances are found. Then, the mean of the output values of these neighbors are taken as the output of the new sample:

$$\hat{f}(y) = \frac{\sum_{i=1}^{k} f(x_i)}{k} \tag{5.14}$$

The accuracy of the K-NN algorithm can be severely degraded by the presence of noisy or irrelevant features in training data. Although the algorithm is easy to implement, it is computationally intensive, especially when the size of the training set grows. In our implementation, different values for *k* are compared (1, 2, 3…) and 3 is chosen to be used in our research.

5.2.2.6. Voting Method. Voting is one of the methods for combining multiple learners and is used to obtain more accurate results [59]. Furthermore, it is used to decrease the effect of noise. In the voting method (shown in Figure 5.2), estimated output values from each learner (*linear regression, support vector regression*, and so on) are multiplied with a particular weight and this weighted sum is taken as the final output. In our research, we take equal weight values for each estimator which is simply equal to *1/number of learners*.

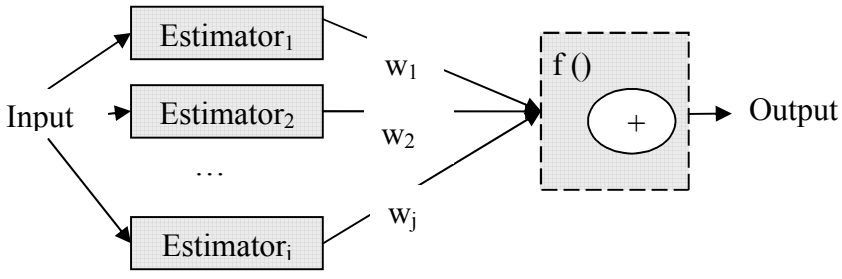

Figure 5.2. Voting method with the summation function *f()*

For example, assume that we have two learners with estimated outputs y_i and y_j. Let the weights for each of them be equal to 0.5. Then, the estimated output of the voting method will be $(y_i*0.5+y_j*0.5)$.

5.2.3. Methods Used for COCLAM

COCLAM has two main steps which are clustering and classification. In the clustering step, an incremental clustering algorithm called *leader cluster* is used. For the classification step, three different methods which are namely *linear discrimination, k-nearest neighbors,* and *decision tree* are used. Linear discrimination is a parametric method whereas k-nearest neighbor and decision tree are non-parametric methods. These three algorithms are chosen to show how our approach performs with algorithms having different complexities. Linear discrimination is the simplest one whereas decision tree is the most complex one. K-nearest neighbor has a moderate complexity depending on the training set size.

As we have stated before, in order to compare the results obtained after classification with the ones obtained in WoCAD and other studies, point estimates are also produced in COCLAM. For this process, the same estimators used in WoCAD are applied again in COCLAM. These estimators are *linear regression, kernel smoother, support vector regression, multi-layer perceptron, k-nearest neighbors,* and *voting algorithm*. These methods are presented in the previous sub-sections, so, only the methods used for clustering and classification will be explained in the following sub-sections.

5.2.3.1. Cluster Analysis. Cluster analysis is a technique for finding similar structures in the data [45]. In software cost estimation domain, clustering corresponds to grouping projects into clusters based on their attributes. Similar projects are assigned to the same cluster whereas dissimilar projects belong to different clusters.

Clustering analysis is not a new concept in software cost estimation domain. For example, Lee et al. integrate clustering with neural networks in order to estimate development cost [60]. They find similar projects with clustering and use them to train the network. In another study, cost data is clustered and then different regression models are fitted to each cluster [61]. Similar to these studies, we also use cluster analysis for grouping similar projects in our research. The difference of our research with these studies is that we combine clustering with classification methods for effort estimation.

In our experiments, we use an incremental clustering method called *leader cluster* for cluster analysis [45]. In this method, total number of clusters to be created is not predefined; instead, the clusters are generated incrementally. The following are the steps of the leader cluster algorithm [62]:

a. Assign the first data item to the first cluster.
b. Consider the next data item. Either assign this item to one of the existing clusters or assign it to a new cluster. This assignment is done based on a criterion such as the distance between the new item and the existing cluster centers. In that case, after every addition of a new item to an existing cluster, the value for the center is recomputed.
c. Repeat step 2 until total squared error is minimized.

The pseudo code of the leader cluster algorithm is given in Figure 5.3. In order to determine the similarity between two projects, *Euclidean distance* is used in our research, because, it is a widely preferred distance metric for cost datasets [60].

```
Input:
      D= {t1, t2, …, tn}     // Set of elements
      A                      // Adjacency matrix showing
                             distance between elements

Output:
      K                      // Set of clusters

Nearest neighbor algorithm:
      K1 = {t1};
      K = {K1};
      k   = 1;

For i=2 to n do
      Find the tm in some cluster Km in K such that
      dis (ti, tm) is the smallest;
            If dis(ti, tm )≤t then
                  Km = KmUti
            Else
                  K = k+1;
                  Kk = {ti};
```

Figure 5.3. Pseudo code for Leader Cluster algorithm [62]

5.2.3.2. Linear Discrimination. *Linear discrimination (LD)* is a discriminant-based approach that tries to fit a model directly to the discriminant between the class regions, without first estimating the likelihoods or posteriors [45]. It assumes that the members of a class are linearly separable from the members of other classes and requires no knowledge of the densities inside class regions. The linear discriminant function that is used by LD is as follows:

$$g_i \langle x | w_i, w_{i0} \rangle = \sum_{j=1}^{d} w_{ij} x_j + w_{i0} \tag{5.15}$$

where g_i is the model, w_i, w_{i0} are the model parameters, and x is the software project with d attributes. This function is used to separate two or more classes.

As done in linear regression, learning involves the optimization of model parameters to maximize the classification accuracy on a given set of projects. Because of its simplicity and understandability, linear discrimination is frequently used before of trying a more complicated model.

5.2.3.3. K-Nearest Neighbor. As we have stated before, *K-NN* assumes that all data samples correspond to points in the *n*-dimensional *Euclidean space* R^n, where n is the number of the sample attributes. The algorithm's output is the class which has the most examples among the k neighbors of the input samples. Here, the choice of k is very important and it is generally taken to be an odd number [45]. In our implementation, different values for k are compared (1, 2, 3…) and 1 is chosen since it gives the lowest misclassification error.

5.2.3.4. Decision Tree. *Decision trees (DT)* are hierarchical data structures that are based on divide-and-conquer strategy [63]. They can be used for both classification and regression and require no assumption about the data. In the case of classification, they are called *classification trees*.

Classification trees are composed of structures called *nodes* and *leaves* as shown in Figure 5.4. Nodes correspond to the attributes that best split the data into disjoint groups

while leaves correspond to the values of the target attribute (class). The goodness of any split is determined by using an *impurity measure*. The tree is constructed by partitioning the data recursively until no further partitioning is possible, each time choosing the split that minimizes the impurity [45].

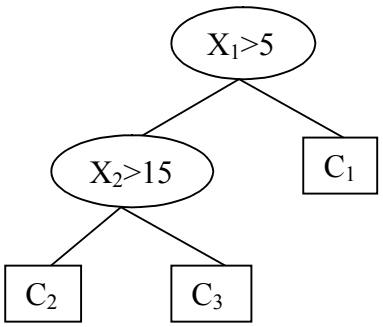

Figure 5.4. A simple classification tree [45]

For example, if N_m is the number of training instances reaching node m, then, the estimate for the probability of class C_i is:

$$\hat{P}(C_i \mid x, m) \equiv p^i{}_m = \frac{N^i m}{N_m} \tag{5.16}$$

Node m is pure if p^i_m for all i are either 0 or 1. The constructed tree is then used for classification process which means traversing the tree by starting at the root until reaching a leaf node.

There are many algorithms that have been developed for learning phase of decision trees, but, the most important one is *ID3*. It was originally developed J. Ross Quinlan in 1975 in his book [63]. The algorithm for ID3 is given in Figure 5.5. ID3 searches through the attributes of the training instances and extracts the attribute that best separates the data samples. If the attribute perfectly classifies the training sets, then it stops; otherwise it recursively operates on the m (the number of possible values of an attribute) partitioned subsets to get their best attribute.

```
Function ID3

Input: (R: a set of non-target attributes,
        C: the target attribute, S: a training set)
Output: a decision tree

Begin

If S is empty,
       return a single node with value Failure;
If S consists of records all with the same value for the
       target attribute,
       return a single leaf node with that value;
If R is empty,
       return a single node with the value of the most
       frequent of the values of the target attribute
       that are found in records of S;
Let A be the attribute with largest Gain(A,S) among
       attributes in R;
Let {aj|j=1,2,..,m} be the values of attribute A;
Let {Sj|j=1,2,..,m} be the subsets of S consisting
       respectively of records with value aj for A;
Return a tree with root labeled A and arcs labeled
       a1,a2,..,am going respectively to the trees
       (ID3(R-{A},C,S1), ID3(R-{A},C,S2),.....,
       ID3(R-{A},C,Sm);
Recursively apply ID3 to subsets {Sj|j=1,2,..,m} until
       they are empty

End
```

Figure 5.5. ID3 Decision Tree algorithm [72]

5.3 Experimental Design

In our experiments, we used MATLAB 7.0.1 Release 14 for implementing the aforementioned algorithms [65]. For support-vector regression and multi-layer perceptron, we used the code packages written in MATLAB [56, 57]. All the remaining algorithms are implemented and integrated by us in MATLAB.

5.3.1. Experimental Design for WoCAD

In WoCAD, our aim is to find out what type of and how much data should be used for embedded software cost estimation. Thus, three experimental setups, that use the proposed model given in Chapter 4, are designed:

- S1: train and validate the estimators only on the embedded dataset
- S2: train the estimators on a subset of the cross-domain dataset by randomly selecting the *same* number of projects as in the embedded dataset, and then validate on the embedded dataset
- S3: train the estimators on the cross domain dataset by using *all* projects and validate on the embedded dataset

Among these setups, S1 defines a baseline where the estimations are carried out with within-domain datasets. In order to investigate how much data should be used for training, S2 and S3 are designed. S2 is a cross-domain set-up, where the training dataset is different from the within-domain dataset, but has the same size as in S1. S3 is the same as S2 except that all of the available data is used as the training set.

There are four within-domain (embedded) and three cross-domain datasets that can be used to compare these three setups with each other (Table 5.2). Embedded datasets are *coc81_e*, *nasa93_e*, *both,* and *ISBSG_e* whereas cross-domain datasets are *others1*, *others2,* and *ISBSG*. In all these datasets, there are great variations between different attributes (e.g. between nominal COCOMO attributes and numerical *size* attribute). Thus, before performing any experiment, all of the datasets are normalized in order to remove scaling effects in different dimensions. *Min-max normalization* is applied to all attributes and attribute values are converted to the (0, 1) interval [64].

In S1, since the estimators are both trained and validated on embedded datasets, *10x10 cross-validation* is used to create various training and validation sets from the same dataset [45]. Firstly, the normalized dataset is 10 times shuffled into random order and then divided into 10 bins. Training data is built from nine of the bins, and the remaining bin is set for validation. Secondly, *Principal Component Analysis* is applied on both training and validation sets to extract relevant features for each of them [45]. Thirdly, the model shown in Figure 4.1 in Chapter 4 is used to estimate effort values for each validation sample. Finally, the results on the validation set and the associated errors are collected for all 100 cross-validation iterations. The pseudo code of S1 is given below:

```
M = 10                                          // num of iterations
N = 10                                          // num of bins
DATA = coc81_e,nasa93_e,both,ISBSG_e            // embedded datasets
REDUCER = PCA                                   // dimensionality reducer
ESTIMATOR = (LR KS SVR MLP KNN)                 // estimators
VOTINGS = (Voting)                              // voting algorithm

for data in DATAS
      N_DATA = NORMALIZE(data)
      repeat M times
            data' = randomize order in N_DATA
            generate N bins from data'
            for i=1 to N
                  validationData = data'(i)
                  trainingData = data'-validationData
                  for reducer in REDUCER
                        data'' = reducer(trainingData)
                        data''' = reducer(validationData)
                  for estimator j in ESTIMATOR
                        predictor = estimator(data'')
                        RESULTS1(j) = apply predictor to data'''
for voter in VOTINGS
      RESULTS2 = voter(RESULTS1)
```

Figure 5.6. Pseudo code of S1

In S2, there is no need for cross-validation because there are separate training (cross-domain datasets) and validation sets (embedded datasets). As we want to use a training set with the same size as the embedded dataset, a group of projects in the cross-domain dataset are selected randomly and used as the training set. After normalization and dimensionality reduction with PCA, as done in S1, the model shown in Figure 4.1 in Chapter 4 is used to estimate effort values for each validation sample. In order to obtain 100 results as in S1, the whole setup is run for 100 times. The pseudo code of S2 is given below:

```
M = 100
TRA_DATA = others1,others2,ISBSG          // training data
VAL_DATA = coc81_e,nasa93_e,both,ISBSG_e  // validation data
REDUCER = PCA                             // dimensionality reducer
ESTIMATOR = (LR KS SVR MLP KNN)           // estimators
VOTINGS = (Voting)                        // voting algorithm

TRA_DATA' = NORMALIZE(TRA_DATA)
VAL_DATA' = NORMALIZE(VAL_DATA)
repeat M times
      tra_data = randomize order in TRA_DATA'
      SIZE = size(VAL_DATA')
      tra_data' = tra_data (1…SIZE)
      for reducer in REDUCER
            tra_data'' = reducer(tra_data')
            val_data = reducer(VAL_DATA')
      for estimator j in ESTIMATOR
            predictor = estimator(tra_data'')
            RESULTS1(j) = apply predictor to val_data
for voter in VOTINGS
      RESULTS2 = voter(RESULTS1)
```

Figure 5.7. Pseudo code of S2

S3 also uses the model shown in Figure 4.1. The only difference is that, instead of using a random subset of projects from cross-domain datasets, all of the projects in them are used as the training set. Here, the aim is to find out how much training data we should use in order to get best performance. The pseudo code S3 is given in Figure 5.8.

```
M = 100
TRA_DATA = others1,others2,ISBSG          // training data
VAL_DATA = coc81_e,nasa93_e,both,ISBSG_e  // validation data
REDUCER = PCA                             // dimensionality reducer
ESTIMATOR = (LR KS SVR MLP KNN)           // estimators
VOTINGS = (Voting)                        // voting algorithm

TRA_DATA' = NORMALIZE(TRA_DATA)
VAL_DATA' = NORMALIZE(VAL_DATA)
repeat M times
      tra_data = randomize order in TRA_DATA'
      val_data = randomize order in VAL_DATA'
      for reducer in REDUCER
            tra_data' = reducer(tra_data)
            val_data' = reducer(val_data)
      for estimator j in ESTIMATOR
            predictor = estimator(tra_data')
            RESULTS1(j) = apply predictor to val_data'
for voter in VOTINGS
      RESULTS2 = voter(RESULTS1)
```

Figure 5.8. Pseudo code of S3

For all setups, after the effort values are estimated for each project in the validation set, three performance measures are calculated in order to compare the setups with each other. These measures will be explained in Chapter 5.4.

5.3.2. Experimental Design for COCLAM

In COCLAM, our aim is to convert effort estimation problem into a classification problem in order to build a predictor with high accuracy. Our proposed approach has three phases (as shown in Figure 4.2): (1) clustering the effort data, (2) labeling each cluster with a class number and determination of the effort intervals for each cluster, (3) classification of the new projects into created effort classes.

The assumption behind applying cluster analysis on effort data is that similar projects have similar development efforts. The class-labeled clusters then become the input data for the classification algorithm which converts cost estimation into a classification process. Different classifiers are used to make our approach as model-independent. In addition, point estimates are produced with the same estimators used in WoCAD to compare our results with the ones generated both in WoCAD and in other studies.

In the experiments, the same within-domain (embedded) datasets as WoCAD, shown in Table 5.2, are used in COCLAM as we want to develop a new cost estimation model for embedded software. Before the clustering phase, as done in WoCAD, all of these datasets are normalized by using Min-Max normalization [64]. Then, normalized effort data is given as the input to the model given in Figure 4.2 and errors and performance measures are calculated during the 10x10 cross-validation loop.

5.4. Performance Measures

The models proposed in this research produce either effort estimates (WoCAD) or effort class estimates (COCLAM). Thus, in our experiments, we must present the results by using two types of performance measures:

- *Mean Magnitude of Relative Error (MMRE)*, *Median Magnitude of Relative Error (MdMRE)*, and *Prediction at level r (PRED(r))* for point estimates
- *misclassification rate* for effort class estimates.

Since, at the end of 10x10 cross-validation, we obtain 100 results for a single estimator, MMRE, MdMRE, and PRED are used to come up with a single result for each estimator. They are the measures calculated from *Magnitude of Relative Error (MRE)* between the actual and estimated values:

$$MRE = \frac{|predicted - actual|}{actual} \tag{5.17}$$

Simply, *Mean Magnitude of Relative Error (MMRE)* is the mean of the MRE values and *MedianMRE (MdMRE)* is the median of the MRE values that are calculated for each validation sample. MMRE and MdMRE give an indication of the capability of a predictive model by measuring its estimation accuracy. The mean, used by MMRE, takes into account the numerical value of every single observation in the data distribution and thus MMRE it is not a good option for skewed distributions (i.e. not symmetric to mean) [67]. On the other hand, the median, which is used by the MdMRE, is able to measure the central tendency of a skewed distribution of data. For these reasons, we want to use both of them to present our results.

Prediction at level r or *PRED(r)* is used to measure the robustness of a predictive model by examining the cumulative frequency of MRE for a specific error level, *r* [66]. In this research, we take the desired error level, *r*, as *25*. For example, if we have *n* projects and there are *m* of them whose MRE is smaller than 25 per cent, then PRED(25) is equal to *m/n*:

$$PRED(N) = \frac{100}{T}\sum_{i}^{T}\begin{cases} 1 & if\ MRE_i \le \frac{N}{100} \\ 0 & otherwise \end{cases} \tag{5.18}$$

One important thing while evaluating the results is that we want MdMRE and MMRE values to be low and PRED(25) values to be high in order to say that a model performs well. Conte et al. consider MMRE≤0.25 and PRED(25)≥0.75 as an acceptable level for effort prediction models [66].

We need a measure of the reliability, or consistency, in addition to the accuracy to better assess the methods [70]. Thus, we include *standard deviations* of MdMRE, MMRE, and PRED measures for each methods used.

Misclassification rate is simply the proportion of the number of the misclassified software projects in a validation set to the total number of projects to be classified in the same validation set. It is calculated for each classification algorithm in the proposed model by using the formula below:

$$MR = \frac{1}{N_t} \sum_{n=1}^{N} \begin{cases} 1 & if\ y' \neq y \\ 0 & otherwise \end{cases} \tag{5.19}$$

where N_t is the total number of training samples, y' is the estimated effort, and y is the actual effort. Misclassification rate can be thought as the complement of the *hit rate* that is mentioned in the studies that perform interval prediction [27, 28]:

$$100\% = MR + HitRate \tag{5.20}$$

Since we have a number of competing techniques that are used to predict the same dataset, *t-test* is used for comparing different sets of predictions [70]. The t-test is a test of the null hypothesis that means of two normally distributed populations are equal. Once a *t-value* is determined, a probability value *p* can be used to evaluate the likelihood of whether two sets of predictions are statistically similar or not. The t-value is given by the following equation:

$$t = \frac{\overline{X}_1 - \overline{X}_2}{s\sqrt{2/n}} \tag{5.21}$$

where s is the pooled sample standard deviation, n is the number of sample size, and $\overline{X}_1$ and $\overline{X}_2$ are the mean value of set 1 and 2 respectively. The *null hypothesis* is that two sample sets of predictions are not significantly different, and the *alternative hypothesis* is that the two sets of predictions are significantly different. If there is not enough statistical evidence at the 95 per cent significance level (α=0.05) to reject the null hypothesis, then we can be confident that the two sets of predictions are not significantly different. The statistical significance indicates if the differences in the means or medians are caused by chance alone or whether the samples were drawn from different underlying populations [69].

5.5. Threats to Validity

As a threat to internal validity of our results, firstly, we constructed four embedded datasets which are small in size. To overcome this issue, we used a 10x10 cross validation framework in our experiments so that, at each iteration, different training and validation sets are generated. Secondly, the use of metrics based on absolute relative error (*MMRE, MdMRE, PRED*) may be inaccurate for experimental evaluation [67, 68]. Although they are the most widely used evaluation criteria for assessing the performance of different prediction models, there are some studies that argue their accuracy. For example, according to Foss et al., both MMRE and MdMRE are inherently biased and do not always select the best model [67]. Also, Korte and Port stated that most of the results of these measures are questionable due to large possible variations resulting from population sampling error [68]. In our research, we try to minimize the population sampling error by using datasets each with different sizes. In order to compare our results with other studies in the literature, we prefer to use these three measures. However, we present the standard deviations for each result obtained in cross-validation loop and use statistical tests (*t-test*) in order to obtain a more accurate comparison of the models developed.

We can say that our results are externally valid, because, both the datasets collected from software companies in Turkey and the datasets from PROMISE Data Repository are used in our experiments instead of relying only on datasets from a single source. Furthermore, ISBSG dataset which contains data about current software development projects from different organizations in the world is used to generalize our results.

6. EXPERIMENTAL RESULTS

In our research, there are two main research questions that we search answers for. The first one (RQ1) is what type of and how much data we should use for training cost models in embedded software cost estimation. We answer this by comparing cost models that are trained on within-domain to those trained on cross-domain and by using different types of training sets (WoCAD). The second research question (RQ2) is that whether we can increase embedded software effort prediction accuracy by converting cost estimation into a classification problem (COCLAM).

In WoCAD, we use four within-domain datasets and three cross-domain datasets from three different sources. In COCLAM, we use the same four within-domain datasets in order to see our proposed model's performance. In addition, we test our model on five more datasets in order to find out if it can also be used for other domains.

For each research question, a number of experiments are conducted by either using 10x10 cross-validation or performing 100 runs in order to obtain consistent results. The results are collected at each iteration or run in terms of four performance measures (MdMRE, MMRE, PRED(25), and misclassification rate). Additionally, standard deviations are calculated in order to see the consistency of the results obtained. For comparing different competing methods, we use t-tests which is a widely-used statistical analysis method.

In the following subsections, results of the experiments will be presented for each dataset and for each research question.

6.1. Experimental Results for WoCAD

In WoCAD, according to the training set used, we have three different setups which are:

- S1: train and validate the estimators only on the embedded dataset

- S2: train the estimators on a subset of the cross-domain dataset by randomly selecting the *same* number of projects as in the embedded dataset, and then validate on the embedded dataset
- S3: train the estimators on the cross domain dataset by using *all* projects and validate on the embedded dataset

For our experiments, there are four within-domain (embedded) datasets which are *coc81_e*, *nasa93_e*, *both*, and *ISBSG_e* and three cross-domain datasets which are *others1*, *others2*, and *ISBSG*. Thus, there are five different cases for each embedded dataset, except *ISBSG_e,* as shown in Table 6.1. *ISBSG_e* dataset can use only *ISBSG* dataset as training data, because, the attributes in *others1* and *others2* are different; so, there are three cases for it: S1 S2, and S3.

Table 6.1. Experimental cases for embedded datasets

Case	Setup	Training Data	Validation Data
C1	S1	-	coc81_e/nasa93_e/both
C2	S2	others1	coc81_e/nasa93_e/both
C3	S2	others2	coc81_e/nasa93_e/both
C4	S3	others1	coc81_e/nasa93_e/both
C5	S3	others2	coc81_e/nasa93_e/both

Six different estimators are applied in all setups that mentioned above. These estimators are support vector regression (SVR), linear regression (LR), Gaussian smother (GS), k-nearest neighbor (KNN), multi-layer perceptron (MLP), and voting. Before applying the estimators, the embedded dataset is firstly normalized and then PCA is applied in order to extract relevant features. Proportion of variance value which we selected as 0.90 is used to decide the optimum number of dimensions to reduce the dataset to.

In WoCAD, our second aim is to investigate how much data we should use in order to train estimators for embedded software cost estimation. Using either a subset (in S2) or all of the projects in cross-domain datasets (in S3), does not give us an idea about the direct relationship between the training set size and the performance of the estimators. In order to observe this relationship, we perform an additional experiment for embedded datasets. In

this experiment, we begin training the embedded datasets by using the same number of projects from the cross-domain dataset and then increase training set size one by one, calculating the results for each estimator at each step. This gives us the opportunity to observe the performances of each estimator for each different training set size. We call this experiment as "Performance Experiment" in the following subsections.

6.1.1. Coc81_e Dataset

This dataset contains information about 28 NASA projects each including 17 COCOMO attributes. By using PCA, the dimensions are reduced to 9.

6.1.1.1. Setup 1 (S1) Results. All of the estimators are both trained and validated on *coc81_e* dataset in the 10x10 cross-validation framework. The results are given in terms of MdMRE, MMRE, and PRED(25) in Table 6.2 with their standard deviation values in parenthesis. We should note that MdMRE and MMRE values must be low and PRED(25) values must be high in order to say that a model performs well.

Table 6.2. Results for coc81_e dataset with S1

Estimator	MdMRE	MMRE	PRED(25)
GS	1528 (36.4)	1528 (36.4)	10.89 (21.9)
KNN	213 (2.7)	213 (2.7)	18.31 (26.1)
LR	718 (14.6)	718 (14.6)	10.89 (20.7)
MLP	1630 (30.3)	1630 (30.3)	10.39 (21.5)
SVR	2988 (68.7)	2988 (68.7)	7.42 (17.8)
Voting	933 (19.5)	933 (19.5)	9.40 (20.87)

Since there are 28 projects in *coc81_e*, when we perform cross-validation, the validation set consists of 2 projects. Thus, the results for MdMRE and MMRE measures are the same because the mean and median of any two MRE values are the same. When we look at the results, we can see that all of the MdMRE and MMRE values are very high and PRED values are very low. The best values for all measures are obtained when KNN is used for effort estimation, but, these values can still be perceived as bad when compared with the optimum values (<0.25 for MMRE and MdMRE, and >75 for PRED).

6.1.1.2. Setup 2 (S2) Results. All of the estimators are trained on the cross-domain dataset (*others1* or *others2*) by using random 28 projects as training set and then validated on *coc81_e* dataset. The results obtained for each training data are given in Table 6.3.

Table 6.3. Results for coc81_e dataset with S2

When others1 is used			
Estimator	MdMRE	MMRE	PRED(25)
GS	97 (0.1)	1278 (10.3)	10.14 (4.8)
KNN	216 (1.1)	944 (4.5)	12.02 (4.8)
LR	436 (2.6)	1608 (7.8)	7.92 (4.8)
MLP	394 (2.1)	1790 (9.4)	8.20 (4.1)
SVR	931 (2.6)	4038 (13.2)	5.41 (2.5)
Voting	436 (1.6)	1596 (4.9)	7.74 (4.7)
When others2 is used			
Estimator	MdMRE	MMRE	PRED(25)
GS	91 (0.06)	754 (8.5)	11.35 (5.0)
KNN	122 (0.7)	658 (5.0)	15.20 (6.1)
LR	377 (3.1)	1314 (11.2)	10.04 (5.3)
MLP	246 (1.9)	1224 (8.0)	10.29 (5.0)
SVR	916 (2.6)	4250 (13.3)	6.01 (2.8)
Voting	326 (1.5)	1473 (5.7)	10.32 (5.3)

When *others1* is used as training data, we can see that MdMRE and MMRE values are very high and PRED values are very low when compared with the optimum values (<0.25 for MMRE and MdMRE, and >75 for PRED). The best values for MMRE and PRED measures are obtained when KNN is used for effort estimation. For PRED, GS gives the best value which is below 100 per cent. When *others2* is used as training data, as when *others1* is used, KNN gives the best MMRE and PRED values whereas GS gives the best PRED value.

6.1.1.3. Setup 3 (S3) Results. All of the estimators are trained on the cross-domain dataset (*others1* or *others2*) by using all projects as training set and then validated on *coc81_e* dataset. The results obtained for each training data are given in Table 6.4.

Table 6.4. Results for coc81_e dataset with S3

When others1 is used			
Estimator	MdMRE	MMRE	PRED(25)
GS	98 (0.2)	1375 (11.8)	10.32 (5.0)
KNN	124 (0)	926 (0)	7.07 (0.7)
LR	670 (0)	1370 (0)	3.53 (0.3)
MLP	411 (1.8)	2265 (9.7)	7.03 (3.6)
SVR	909 (0)	3511 (0)	3.53 (0.3)
Voting	479 (0.9)	1530 (3.5)	1.77 (2.1)
When others2 is used			
Estimator	MdMRE	MMRE	PRED(25)
GS	92 (0.07)	743 (6.9)	11.42 (5.6)
KNN	87 (0)	464 (0)	10.60 (1.0)
LR	476 (0)	1288 (0)	7.07 (0.7)
MLP	268 (1.3)	1327 (5.7)	9.97 (5.0)
SVR	942 (1.6)	3780 (8.3)	3.53 (0.3)
Voting	433 (1.4)	1402 (0)	6.61 (1.9)

When *others1* is used as training data, we can see that MdMRE and MMRE values are very high and PRED values are very low. The best values for MdMRE and PRED measures are obtained when GS is used for effort estimation. For MMRE, KNN gives the best value. When *others2* is used as training data, as when *others1* is used, GS gives the best PRED value. On the other hand, KNN gives the best MdMRE and MMRE values.

6.1.1.4. Performance Experiment Results. We perform the performance experiment by training the estimators on *others1* dataset and validating on *coc81_e* dataset. We begin with 28 projects in training set and continue by increasing the number one by one until all of the 149 projects in *others1* are used. The results obtained are given in Figure 6.1.

When we look at Figure 6.1, we can see that there is not a general tendency, such as effort estimation performance gets better or worse as the training set size increases. However, we will try to come up with a conclusion for each algorithm. SVR gives better MMRE and MdMRE values as the training set size increases whereas PRED values always change between 5.95 per cent and 2.97 per cent independently from the training set size. Linear regression gives better values for MdMRE and PRED when the training set size is

around 100, and for MMRE when size is around 75. MLP gives better values for all measures when the training set size is around 40. KNN gives better values for MdMRE and MMRE when the size is around 120 and 85 for PRED. For Gaussian smoother, MdMRE values change between 80 per cents and 90 per cents independently from the training set size. For both MMRE and PRED, better values are obtained when the size is around 46. Voting gives better values for MdMRE and MMRE when the size is around 100 and 98 for PRED.

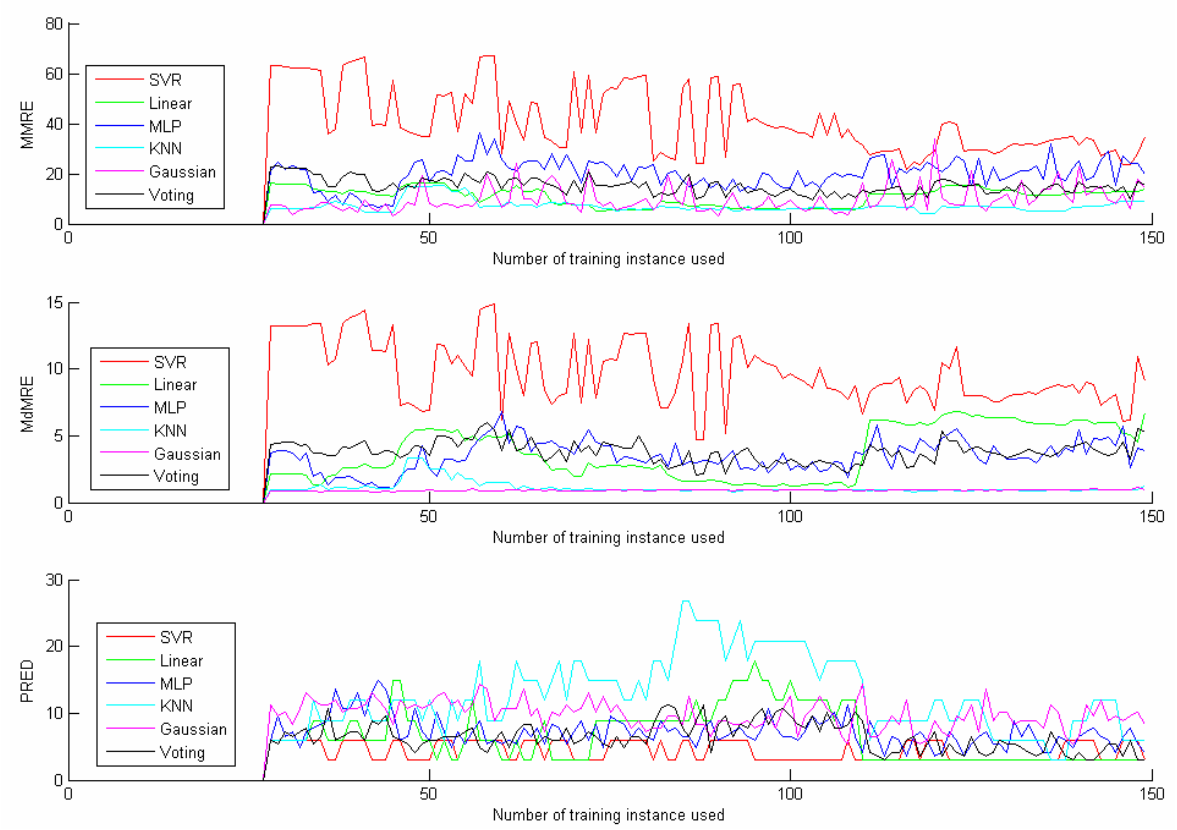

Figure 6.1. Performance results for coc81_e

6.1.2. Nasa93_e Dataset

This dataset contains information about 21 NASA projects each including 24 COCOMO II attributes. By using PCA, the dimensions are reduced to 7.

6.1.2.1. Setup 1 Results. All of the estimators are both trained and validated on *nasa93_e* dataset in the 10x10 cross-validation framework. The results are given in terms of

MdMRE, MMRE, and PRED(25) in Table 6.5 with their standard deviation values in parenthesis.

Table 6.5. Results for nasa93_e dataset with S1

Estimator	MdMRE	MMRE	PRED(25)
GS	138 (2.6)	138 (2.6)	32.67 (31.1)
KNN	193 (3.2)	193 (3.2)	27.22 (31.2)
LR	116 (0.8)	116 (0.8)	24.75 (30.5)
MLP	175 (2.2)	175 (2.2)	28.21 (33.4)
SVR	444 (7.6)	444 (7.6)	16.37 (25.6)
Voting	187 (2.7)	187 (2.7)	26.23 (31.3)

Since there are 21 projects in *naas93_e*, when we perform cross-validation, the validation set consists of 2 projects. Thus, the results for MdMRE and MMRE measures are the same because the mean and median of any two MRE values are the same. When we look at the results, we can see that all of the MdMRE and MMRE values are over 100 per cent and PRED values are around 20 per cents. These values can be perceived as bad when compared with the optimum values (<0.25 for MMRE and MdMRE, and >75 for PRED). The best values for MMRE and MdMRE are obtained when LR is used for effort estimation whereas best value for PRED is obtained when GS is used.

6.1.2.2. Setup 2 Results. All of the estimators are trained on the cross-domain dataset (*others1* or *others2*) by using random 21 projects as training set and then validated on *nasa93_e* dataset. The results obtained for each training data are given in Table 6.6.

When *others1* is used as training data, there are some MdMRE values below 100 per cent which can be accepted as good. On the other hand, MMRE values are still high and PRED values are still low. The best values for MdMRE and PRED measures are obtained when KNN is used for effort estimation. For MMRE, GS gives the best value which is 317 per cent. When *others2* is used as training data, Voting gives the best MdMRE and PRED values whereas GS gives the best MMRE value.

Table 6.6. Results for nasa93_e dataset with S2

When others1 is used			
Estimator	MdMRE	MMRE	PRED(25)
GS	92 (0.1)	317 (2.5)	12.73 (6.2)
KNN	75 (0.1)	456 (2.7)	19 (7.6)
LR	108 (0.4)	675 (4.0)	15.7 (7.1)
MLP	112 (0.5)	546 (3.4)	16.36 (7.9)
SVR	274 (0.9)	922 (2.8)	12.58 (6.3)
Voting	86 (0.2)	533 (2.3)	18.81 (7.8)
When others2 is used			
Estimator	MdMRE	MMRE	PRED(25)
GS	88 (0.06)	189 (1.8)	12.02 (6.3)
KNN	72 (0.1)	212 (1.1)	16.36 (8.7)
LR	97 (0.4)	422 (3.1)	15.60 (6.3)
MLP	100 (0.4)	297 (1.8)	15.18 (7.7)
SVR	276 (1.0)	864 (2.6)	12.91 (5.5)
Voting	69 (0.2)	334 (1.6)	20.03 (7.3)

6.1.2.3. Setup 3 Results. All of the estimators are trained on the cross-domain dataset (*others1* or *others2*) by using all projects as training set and then validated on *nasa93_e* dataset. The results obtained for each training data are given in Table 6.7.

Table 6.7. Results for nasa93_e dataset with S3

When others1 is used			
Estimator	MdMRE	MMRE	PRED(25)
GS	90 (0.08)	283 (2.8)	12.63 (6.1)
KNN	73 (0)	1114 (0)	14.14 (1.4)
LR	78 (0)	871 (0)	33 (3.3)
MLP	125 (0.4)	736 (3.58)	14.09 (6.0)
SVR	169 (0)	619 (0)	9.42 (0.9)
Voting	72 (0.1)	683 (0.9)	22.96 (7.3)
When others2 is used			
Estimator	MdMRE	MMRE	PRED(25)
GS	90 (0.04)	179 (1.9)	11.12 (5.0)
KNN	59 (0)	175 (0)	23.57 (2.3)
LR	72 (0)	478 (0)	28.28 (2.8)
MLP	91 (0.3)	368 (2.2)	16.50 (7.1)
SVR	174 (0.1)	919 (1.8)	14.05 (1.7)
Voting	60 (0.5)	372 (1.5)	21.97 (7.6)

When *others1* is used as training data, we can see that four of the MdMRE values are below 100 per cent whereas MMRE and PRED values are still far away from the optimum values. As in S2, Voting gives the best MdMRE and PRED values whereas GS gives the best MMRE value. It is also remarkable that KNN, LR and Voting give similar MdMRE values. When *others2* is used as training data, LR gives the best PRED value. On the other hand, KNN gives the best MdMRE and MMRE values. Voting also gives a similar MdMRE value to KNN's.

6.1.2.4. Performance Experiment Results. We perform the performance experiment by training the estimators on *others1* dataset and validating on *nasa93_e* dataset. We begin with 21 projects in training set and continue by increasing the number one by one until all of the 149 projects in *others1* are used. The results obtained are given in Figure 6.2.

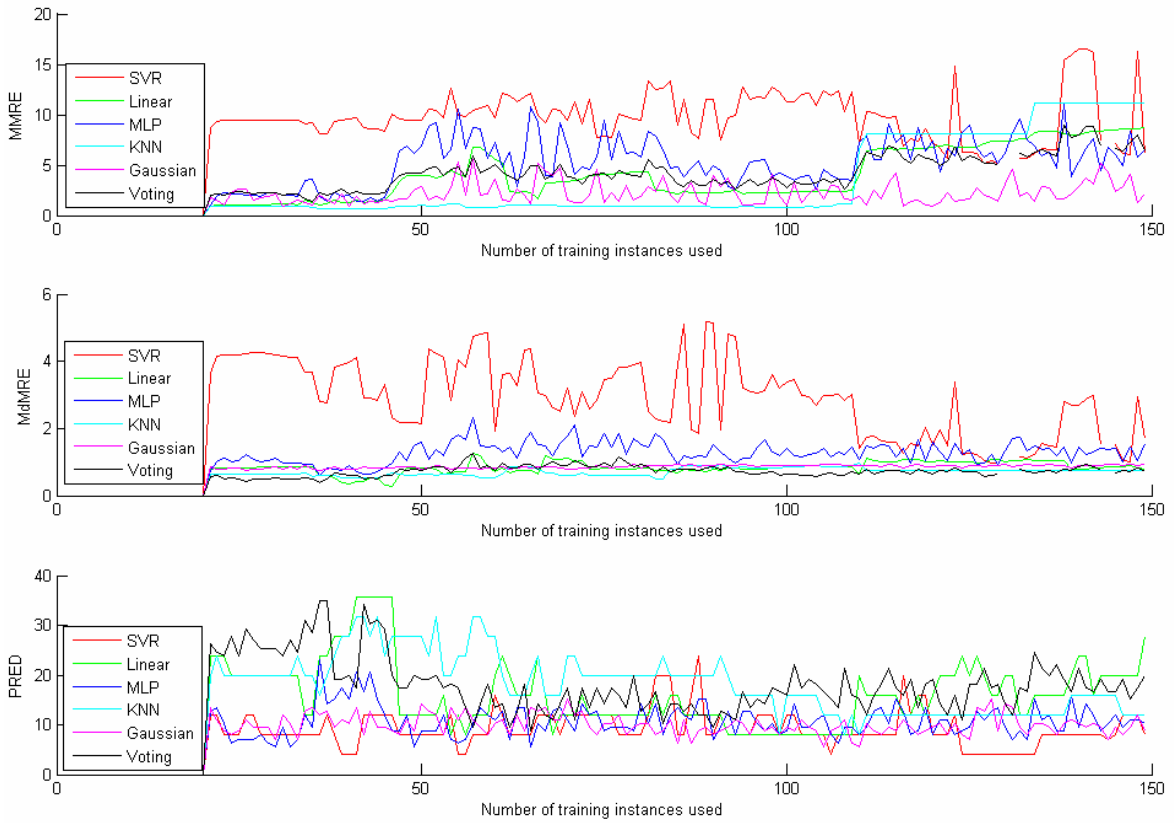

Figure 6.2. Performance results for nasa93_e

When we look at Figure 6.2, as in *coc81_e*, there is not a general tendency about the estimators' performances, so, we will try to come up with a conclusion for each algorithm. SVR gives better values for all measures when the training set size is around 128. For

Linear regression, MdMRE values always change between 82 per cent and 102 per cent independently from the training set size whereas it gives better values for MMRE when size is around 100, and for PRED when size is around 43. MLP gives better values for all measures when the training set size is around 42. For KNN, MMRE values seem to get worse as the training set size increases and is best when the size is 42. It gives better values for MdMRE when the size is around 59 and 57 for PRED. For Gaussian smoother, MdMRE values change between 70 per cents and 90 per cents independently from the training set size. For both MMRE and PRED, better values are obtained when the size is around 34. Voting gives MdMRE values that change between 57 per cent and 94 per cent independently from the training set size whereas it gives better values for MMRE when the size is around 30 and 36 for PRED.

6.1.3. Both Dataset

This dataset contains information about 49 projects each including 15 COCOMO attributes. By using PCA, the dimensions are reduced to 9.

6.1.3.1. Setup 1 Results. All of the estimators are both trained and validated on *both* dataset in the 10x10 cross-validation framework. The results are given in terms of MdMRE, MMRE, and PRED(25) in Table 6.8 with their standard deviation values in parenthesis.

Table 6.8. Results for both dataset with S1

Estimator	MdMRE	MMRE	PRED(25)
GS	344 (13.4)	1442 (35.1)	46.78 (16.4)
KNN	183 (2.2)	635 (10)	13.36 (16)
LR	273 (4.2)	1026 (20.2)	12.12 (13.8)
MLP	539 (13.3)	2244 (62.1)	8.66 (13.6)
SVR	1039 (12.7)	4698 (89.8)	7.92 (13)
Voting	363 (5.8)	1553 (33.1)	9.40 (13.6)

When we look at the results, we can see that all of the MdMRE and MMRE values are over 100 per cent and PRED values are around 10 per cents except one (46.78 per cent for GS). These values can be perceived as bad when compared with the optimum values

(<0.25 for MMRE and MdMRE, and >75 for PRED). The best values for MMRE and MdMRE are obtained when KNN is used for effort estimation whereas best value for PRED is obtained when GS is used.

6.1.3.2. Setup 2 Results. All of the estimators are trained on the cross-domain dataset (*others1* or *others2*) by using random 49 projects as training set and then validated on *both* dataset. The results obtained for each training data are given in Table 6.9.

Table 6.9. Results for both dataset with S2

When others1 is used			
Estimator	MdMRE	MMRE	PRED(25)
GS	95 (0.03)	1984 (13.8)	8.99 (3.5)
KNN	119 (0.4)	2717 (16.3)	12.54 (4.9)
LR	233 (0.8)	3780 (13.3)	8.83 (4.7)
MLP	258 (0.9)	3779 (21.9)	8.04 (3.1)
SVR	597 (1.9)	5864 (14.6)	5.29 (2.0)
Voting	231 (0.6)	3405 (8.9)	9.73 (4.2)
When others2 is used			
Estimator	MdMRE	MMRE	PRED(25)
GS	92 (0.03)	1133 (11.2)	10 (3.7)
KNN	93 (0.1)	1514 (9.4)	12.81 (4.6)
LR	188 (0.9)	2587 (11.9)	9.67 (4.8)
MLP	189 (0.8)	2349 (15.6)	9.45 (4.0)
SVR	603 (2.0)	5495 (11.5)	5.79 (2.5)
Voting	182 (0.5)	2385 (6.6)	10.32 (4)

When *others1* is used as training data, there is only one MdMRE value below 100 per cent. On the other hand, MMRE values are still high and PRED values are still low. The best values for MdMRE and MMRE measures are obtained when GS is used for effort estimation. For PRED, KNN gives the best value which is 317 per cent. When *others2* is used as training data, GS gives the best MdMRE and MMRE values whereas KNN gives the best PRED value. It also gives a similar value of MdMRE to GS's.

6.1.3.3. Setup 3 Results. All of the estimators are trained on the cross-domain dataset (*others1* or *others2*) by using all projects as training set and then validated on *both* dataset. The results obtained for each training data are given in Table 6.10.

Table 6.10. Results for both dataset with S3

When others1 is used			
Estimator	MdMRE	MMRE	PRED(25)
GS	95 (0.03)	1662 (11.5)	9.27 (4.1)
KNN	93 (0)	3863 (0)	8.08 (0.8)
LR	229 (0)	4102 (0)	6.06 (0.6)
MLP	285 (0.9)	4153 (16.9)	7.07 (3.4)
SVR	422 (0)	4960 (0)	6.06 (0.6)
Voting	207 (0.3)	3532 (4.4)	8.54 (3.2)
When others2 is used			
Estimator	MdMRE	MMRE	PRED(25)
GS	92 (0.03)	1003 (9.5)	9.29 (4.08)
KNN	88 (0)	2616 (0)	6.06 (0.6)
LR	188 (0)	2718 (0)	16.16 (1.6)
MLP	175 (0.8)	2438 (14.6)	9.59 (4.5)
SVR	732 (0)	6261 (0)	4.04 (0.4)
Voting	198 (0.2)	2878 (4.0)	9.88 (2.6)

When *others1* is used as training data, we can see that two of the MdMRE values are below 100 per cent whereas MMRE and PRED values are still far away from the optimum values. As in S2, GS gives the best MMRE and PRED values whereas KNN gives the best MdMRE value. When *others2* is used as training data, GS gives the best MMRE value. On the other hand, KNN gives the best MdMRE value and LR gives the best PRED value.

6.1.3.4. Performance Experiment Results. We perform the performance experiment by training the estimators on *others1* dataset and validating on *both* dataset. We begin with 49 projects in training set and continue by increasing the number one by one until all of the 149 projects in *others1* are used. The results obtained are given in Figure 6.3.

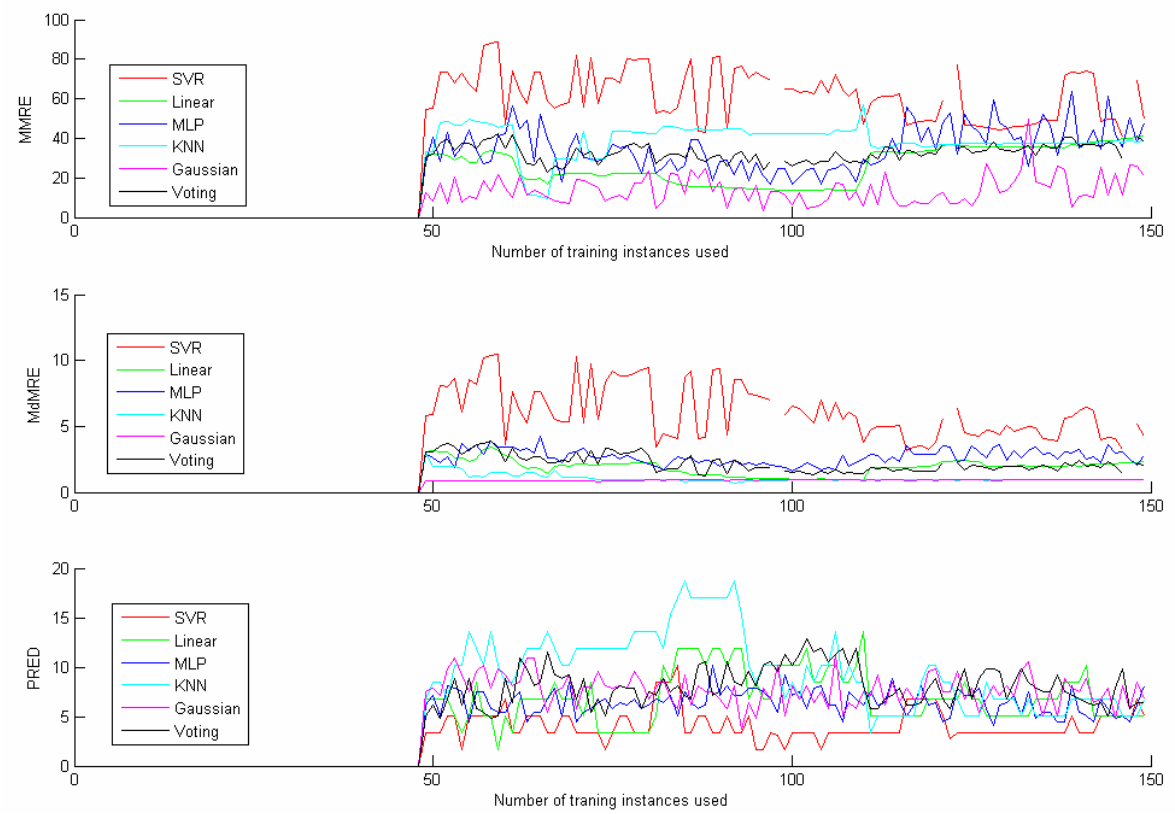

Figure 6.3. Performance results for both

When we look at Figure 6.3, as in *coc81_e* and *nasa93_e*, there is not a general tendency about the estimators' performances, so, we will try to come up with a conclusion for each algorithm. SVR gives better values for MdMRE and MMRE measures when the training set size is around 125 and 118 for PRED. For Linear regression, all values for three measures are better when the training size is around 106. MLP gives better values for all measures when the training set size is around 99. For KNN, MdMRE and PRED values are better when the training set size is around 92. It gives better values for MMRE when the size is around 63. For Gaussian smoother, MdMRE values change between 80 per cents and 90 per cents independently from the training set size. For both MMRE and PRED, better values are obtained when the size is around 102. Voting gives MdMRE values that change between 159 per cent and 346 per cent independently from the training set size whereas it gives better values for MMRE when the size is around 66 and 104 for PRED.

6.1.4. ISBSG_e Dataset

This dataset contains information about 17 projects from different countries and industries each including 8 attributes. By using PCA, the dimensions are reduced to 4.

6.1.4.1. Setup 1 Results. All of the estimators are both trained and validated on *ISBSG_e* dataset in the 10x10 cross-validation framework. The results are given in terms of MdMRE, MMRE, and PRED(25) in Table 6.11 with their standard deviation values in parenthesis.

Table 6.11. Results for ISBSG_e dataset with S1

Estimator	MdMRE	MMRE	PRED(25)
GS	7599 (366)	7599 (366)	14.85 (35)
KNN	187 (3.6)	187 (3.6)	35.64 (48)
LR	1878 (37.3)	1878 (37.3)	6.93 (25)
MLP	91 (2.7)	91 (2.7)	68.3 (46)
SVR	9819 (211)	9819 (211)	22.7 (42)
Voting	3239 (94)	3239 (94)	10.8 (31)

Since there are 17 projects in *ISBSG_e*, when we perform cross-validation, the validation set consists of 2 projects. Thus, the results for MdMRE and MMRE measures are the same because the mean and median of any two MRE values are the same. When we look at the results, we can see that all of the MdMRE and MMRE values are over 100 per cent except those for MLP which are 91 per cent. PRED values are different for each algorithm. The best value for PRED is obtained when MLP is used and is close to the optimum value of 75 per cent.

6.1.4.2. Setup 2 Results. All of the estimators are trained on the cross-domain dataset (*ISBSG*) by using random 17 projects as training set and then validated on *ISBSG_e* dataset. The results obtained are given in Table 6.12.

Table 6.12. Results for ISBSG_e dataset with S2

Estimator	MdMRE	MMRE	PRED(25)
GS	120 (0.5)	1590 (10.0)	13.33 (5.9)
KNN	216 (1.4)	1509 (14.4)	9.14 (7.2)
LR	244 (1.6)	2585 (36.4)	8.21 (4.1)
MLP	299 (1.8)	2754 (34.4)	8.85 (4.5)
SVR	2258 (7.0)	11483 (33.5)	11.53 (4.7)
Voting	623 (2.1)	3800 (16.2)	6.69 (2.3)

When we look at the results, MdMRE and MMRE values are high and PRED values are low. The best values for MdMRE and PRED measures are obtained when GS is used for effort estimation. For MMRE, KNN gives the best value which is still far larger than the optimum value of 25 per cent.

6.1.4.3. Setup 3 Results. All of the estimators are trained on the cross-domain dataset (*ISBSG*) by using all projects as training set and then validated on *ISBSG_e* dataset. The results obtained are given in Table 6.13.

Table 6.13. Results for ISBSG_e dataset with S3

Estimator	MdMRE	MMRE	PRED(25)
GS	123 (0.4)	2418 (33)	11.88 (5.9)
KNN	194 (0)	648 (0)	11.64 (1.1)
LR	198 (0)	1644 (0)	5.82 (0.5)
MLP	524 (4.0)	3026 (22.9)	8.50 (4.7)
SVR	2919 (0)	14248 (0)	17.47 (1.7)
Voting	762 (1.6)	4230 (8.9)	5.94 (1.0)

When we look at the results, we can see that all values for three measures are far away from the optimum values. As in S2, GS gives the best MdMRE value whereas KNN gives the best MMRE value. SVR gives the best PRED value which is 17 per cent.

6.1.4.4. Performance Experiment Results. We perform the performance experiment by training the estimators on *ISBSG* dataset and validating on *ISBSG_e* dataset. We begin with 17 projects in training set and continue by increasing the number one by one until all of the 104 projects in *ISBSG* are used. The results obtained are given in Figure 6.4.

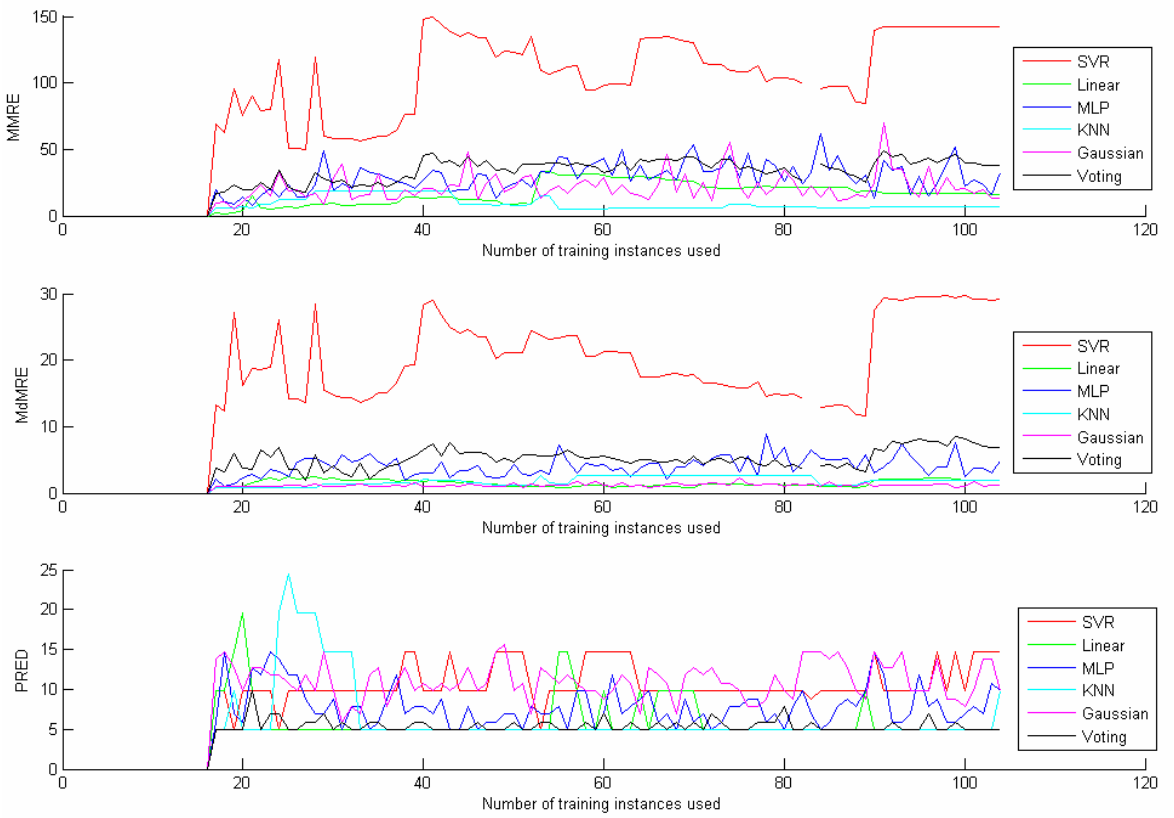

Figure 6.4. Performance results for ISBSG_e

When we look at Figure 6.4, as in the previous datasets, there is not a general tendency about the estimators' performances, so, we will try to come up with a conclusion for each algorithm. SVR gives better values for MdMRE when the training set size is around 25 and 88 for MMRE. PRED values change between 5 per cents and 14 per cents independently from the training set size. For Linear regression, also, PRED values change between these values and MdMRE values change between 81 per cent and 207 per cent independently from the training set size. For MMRE, values are better when the training size is around 18. MLP gives better values for MMRE when the training set size is around 26 and 18 for MdMRE. For PRED, the values are better when the size is around 23. KNN gives better values for MMRE when the size is around 55 and 21 for MdMRE. For PRED, it gives better values when the size is around 26. For Gaussian smoother, MdMRE values change between 90 per cents and 140 per cents independently from the training set size. For both MMRE, better values are obtained when the size is around 92 and 32 for PRED. Voting gives PRED values that change between 490 per cent and 686 per cent

independently from the training set size whereas it gives better values for MdMRE and MMRE when the size is around 32.

6.1.5. Comparison of the Setups

Up to now, the effort estimation results for each setup are presented separately for each dataset. However, we still do not have an idea about which type of training data (within-domain or cross-domain) we should use for embedded software cost estimation. According to the MdMRE, MMRE, and PRED results obtained, most of the best values for either measure were obtained when the estimators are trained on the embedded datasets (S1). However, in order to come up with a neutral conclusion, we performed *t-tests* between different sets of results of each algorithm.

For example, for *coc81_e* dataset, we have five different sets of SVR results because we have five different cases as shown in Table 6.1. In order to find out in which setup SVR algorithm performs best, we compare the SVR results obtained in C1 with those obtained C2, C3, C4, and C5 respectively. Then, the same comparison is made between C2 and C3, C4, and C5 respectively, and so on. In total, we performed 10 t-tests for a single algorithm and for a single measure. Since we have 3 different measures (MdMRE, MMRE, and PRED), we performed 30 t-tests for only one algorithm. While performing the t-tests, we used 0.05 significance level (□=0.05). According to the t-tests performed, the best performing setups obtained for each estimator are given for *coc81_e* dataset in Table 6.14.

Table 6.14. T-test results for coc81_e dataset

Estimator	MdMRE	MMRE	PRED(25)
SVR	S3 with others1	S3 with others1	S2 with others2
GS	S3 with others2	S3 with others2	S3 with others2
LR	S2 with others2	S1	S2 with others2
MLP	S3 with others2	S3 with others2	S3 with others2
KNN	S3 with others2	S1	S2 with others2
Voting	S2 with others2	S1	S2 with others2

When we look at the results, S3 with *others2* performs best for most of the algorithms given in Table 6.14. This means that when the estimators are trained by using

all of the projects in *others2*, best performances are obtained for *coc81_e* dataset.

The best performing setups for *nasa93_e* dataset are given in Table 6.15. When we look at the results, S3 with *others2* again performs best for most of the algorithms. This means that when the estimators are trained by using all of the projects in *others2*, best performances are obtained for *nasa93_e* dataset.

Table 6.15. T-test results for nasa93_e dataset

Estimator	MdMRE	MMRE	PRED(25)
SVR	S3 with others1	S1	S3 with others2
GS	S3 with others1	S3 with others2	S1
LR	S3 with others2	S1	S3 with others2
MLP	S3 with others2	S1	S1
KNN	S3 with others2	S3 with others2	S3 with others2
Voting	S3 with others2	S1	S3 with others2

The best performing setups for *both* dataset are given in Table 6.16. As in *coc81_e* and *nasa93_e*, when the estimators are trained by using all of the projects in *others2*, best performances are obtained for *nasa93_e* dataset.

Table 6.16. T-test results for both dataset

Estimator	MdMRE	MMRE	PRED(25)
SVR	S3 with others1	S3 with others1	S3 with others1
GS	S3 with others2	S3 with others2	S1
LR	S3 with others2	S1	S3 with others2
MLP	S3 with others2	S3 with others2	S3 with others2
KNN	S3 with others2	S1	S2 with others2
Voting	S2 with others2	S1	S3 with others2

The best performing setups for *ISBSG_e* dataset are given in Table 6.17. We should note that *ISBSG_e* dataset can use only *ISBSG* dataset as training data, because, the attributes in *others1* and *others2* are different; so, there are three possible cases: S1 S2, and S3.

Table 6.17. T-test results for ISBSG_e dataset

Estimator	MdMRE	MMRE	PRED(25)
SVR	S2	S2	S3
GS	S3	S2	S2
LR	S3	S3	S2
MLP	S1	S1	S1
KNN	S3	S1	S1
Voting	S2	S2	S2

When we look at the results, as being different from the previous results, S2 performs best for most of the algorithms. This means that when the estimators are trained by using the same number of projects from *ISBSG*, best performances are obtained for *ISBSG_e* dataset.

The summary of the results for WoCAD is given in Table 6.18. As a result, for all of the embedded datasets, the estimators that are trained on cross-domain datasets outperform those trained on the embedded datasets. Thus, we can conclude that cross-domain datasets should be used for training estimators in embedded software cost estimation.

Table 6.18. Summary of the results for WoCAD

Dataset	Best Performing Setup
coc81_e	S3 with others2
nasa93_e	S3 with others2
both	S3 with others2
ISBSG_e	S2

On the other hand, for three of the embedded datasets used, using all the projects in the cross-domain dataset as training data gives the best results. For only one dataset, *ISBSG_e*, using the same number of projects from cross-domain dataset performs best. From these results; it may seem that as training set size increases performance gets better for cross-domain datasets. However, in order to make a conclusion about how much data should be used as training data, we must take the performance experiments into account. In the performance experiments we made, we observed that there is not a direct relationship between training set size and the performances of the estimators. We can only suggest that

trying all possible training set sizes and then selecting the best one to use in experiments.

6.2. Experimental Results for COCLAM

In COCLAM, our aim is to convert effort estimation problem into a classification problem in order to build a predictor for embedded software with high accuracy. Our proposed approach has three phases: (1) clustering the effort data, (2) labeling each cluster with a class number and determination of the effort intervals for each cluster, (3) classification of the new projects into created effort classes.

In the experiments, the same embedded datasets in WoCAD are used: *nasa93_e*, *coc81_e*, *both*, and *ISBSG_e*. Firstly, before the clustering phase, as done in WoCAD, all of these datasets are normalized and then relevant features are extracted by using PCA. Secondly, clustering is applied to these new datasets. The numbers of effort clusters created for each embedded dataset are given in Table 6.19. In order to show the clustering efficiency, the minimum and maximum numbers of projects assigned to a cluster are also given. Finally, three different classifiers are used in the 10x10 cross-validation loop to make our approach model-independent. These classifiers are *k-nearest neighbor (KNN)*, *linear discrimination (LD)*, and *decision tree (DT)*. Misclassification errors are collected during the 10x10 cross-validation loop.

Table 6.19. Clustering results for embedded datasets

Dataset	Number of Clusters	Number of Projects	
		Minimum	Maximum
coc81_e	3	3	14
nasa93_e	4	2	6
both	6	2	17
ISBSG_e	3	3	6

In addition, point estimates are produced with the same six estimators used in WoCAD to compare our results with the ones generated both in WoCAD and in other studies. These estimators are support vector regression (SVR), linear regression (LR), Gaussian smother (GS), k-nearest neighbor (KNN), multi-layer perceptron (MLP), and

voting. The same three performance measures, MdMRE, MMRE, and PRED(25), are calculated for each estimator.

6.2.1. Coc81_e Dataset

This dataset contains information about 28 NASA projects each including 17 COCOMO attributes. By using PCA, the dimensions are reduced to 9. As given in Table 6.19, there are 3 effort classes for this dataset. Classification results and point estimation results for this dataset will be given in the following subsections.

6.2.1.1. Classification Results. Misclassification results of each classifier are given in Table 6.20. Here, we can see that LD outperforms KNN with a slight difference which is 5 per cent. DT is the worst of all giving 19 per cent misclassification rate.

Table 6.20. Classification results for coc81_e

Classifier	Misclassification Rate (per cent)
DT	19
KNN	5
LD	4.5

6.2.1.2. Point Estimation Results. In the cross-validation framework, the estimators are applied just after the effort class is estimated by the classifiers. The effort estimation results for each classifier are given in the following tables. Since there are 28 projects in *coc81_e*, when we perform cross-validation, the validation set consists of 2 projects. Thus, the results for MdMRE and MMRE measures are the same for each classifier because the mean and median of any two MRE values are the same.

Table 6.21. Point estimation results for coc81_e dataset with DT

Estimator	MdMRE	MMRE	PRED(25)
GS	348 (5.1)	348 (5.1)	17.82 (26.6)
KNN	91 (1.1)	91 (1.1)	42.07 (34.4)
LR	43 (0.6)	43 (0.6)	55.44 (33.8)
MLP	52 (0.5)	52 (0.5)	39.60 (35.5)
SVR	99 (0)	99 (0)	0 (0)
Voting	92 (1.3)	92 (1.3)	38.11 (30.1)

When we look at the results of DT, we can see that all values for MdMRE and MMRE are below 100 per cent except those for Gaussian. LR gives the best values for all measures and these values can be accepted as good as they are approximate to the optimum values (per cent25 for MdMRE and MMRE, 75 per cent for PRED).

Table 6.22. Point estimation results for coc81_e dataset with KNN

Estimator	MdMRE	MMRE	PRED(25)
GS	340 (5.1)	340 (5.1)	31.68 (32.9)
KNN	84 (1.1)	84 (1.1)	55.94 (35.5)
LR	35 (0.6)	35 (0.6)	69.30 (31.5)
MLP	35 (0.3)	35 (0.3)	52.97 (37.2)
SVR	99 (0)	99 (0)	0 (0)
Voting	88 (1.3)	88 (1.3)	51.48 (32.7)

When we look at the results of KNN, we can see that all values for MdMRE and MMRE are again below 100 per cent except those for Gaussian. LR gives the best values for all measures.

Table 6.23. Point estimation results for coc81_e dataset with LD

Estimator	MdMRE	MMRE	PRED(25)
GS	339 (5.1)	339 (5.1)	36.13 (33.2)
KNN	85 (1.1)	85 (1.1)	51.48 (34.9)
LR	34 (0.6)	34 (0.6)	73.76 (30.4)
MLP	44 (0.5)	44 (0.5)	52.47 (36.3)
SVR	99 (0)	99 (0)	0 (0)
Voting	90 (1.3)	90 (1.3)	42.57 (35.6)

As in the previous results, we can see that all values for MdMRE and MMRE are again below 100 per cent except those for Gaussian. LR gives the best values for all measures and this time PRED value is very close to the optimum value of 75 per cent. With 34 per cent MdMRE and MMRE values, LR can be accepted a good estimator.

6.2.2. Nasa93_e Dataset

This dataset contains information about 21 NASA projects each including 24

COCOMO II attributes. By using PCA, the dimensions are reduced to 7. As given in Table 6.19, there are 4 effort classes for this dataset. Classification results and point estimation results for this dataset will be given in the following subsections.

6.2.2.1. Classification Results. Misclassification results of each classifier are given in Table 6.24. Here, we can see that KNN outperforms the others with 0 per cent misclassification rate. LD outperforms DT with a slight difference which is 4 per cent.

Table 6.24. Classification results for nasa93_e

Classifier	Misclassification Rate (per cent)
DT	30
KNN	0
LD	26

6.2.2.2. Point Estimation Results. In the cross-validation framework, the estimators are applied just after the effort class is estimated by the classifiers. The effort estimation results for each classifier are given in the following tables. As in *coc81_e*, when we perform cross-validation, the validation set consists of 2 projects for *nasa93_e*, too. Thus, the results for MdMRE and MMRE measures are the same for each classifier.

Table 6.25. Point estimation results for nasa93_e dataset with DT

Estimator	MdMRE	MMRE	PRED(25)
GS	321 (9.6)	321 (9.6)	69.30 (46.3)
KNN	319 (9.5)	319 (9.5)	69.30 (46.3)
LR	333 (10.0)	333 (10.0)	69.30 (46.3)
MLP	329 (9.8)	329 (9.8)	69.30 (46.3)
SVR	99 (0.003)	99 (0.003)	0 (0)
Voting	273 (7.7)	273 (7.7)	64.35 (48.1)

When we look at the results of DT, we can see that all values for MdMRE and MMRE are above 100 per cent and similar to each other except those for SVR and Voting. SVR gives the best values for MdMRE and MMRE and the remaining algorithms give nearly the same PRED values.

When we look at the results of KNN (Table 6.26), we can see that all values for MdMRE and MMRE are very low except those for SVR and LR. For PRED, the results are high which show that the estimators except SVR perform well.

Table 6.26. Point estimation results for nasa93_e dataset with KNN

Estimator	MdMRE	MMRE	PRED(25)
GS	8 (0.1)	8 (0.1)	55.05 (24.1)
KNN	14 (0.2)	14 (0.2)	88.11 (21.9)
LR	31 (0)	31 (0)	99.01 (22.2)
MLP	13 (0.2)	13 (0.2)	87.12 (22.0)
SVR	99 (0.005)	99 (0.005)	0 (0)
Voting	20 (0.07)	20 (0.07)	90.09 (24.0)

As in the results for KNN, for *nasa93_e* results shown below, we can see that all values for MdMRE and MMRE are again low except SVR. LR gives the best values for MdMRE and MMRE. For PRED, GS Voting, and LR give similar values that are also very close to the optimum value of 75 per cent.

Table 6.27. Point estimation results for nasa93_e dataset with LD

Estimator	MdMRE	MMRE	PRED(25)
GS	24 (0.3)	24 (0.3)	74.25 (43.9)
KNN	28 (0.3)	28 (0.3)	67.32 (47.1)
LR	14 (0.3)	14 (0.3)	73.26 (44.4)
MLP	29 (0.3)	29 (0.3)	62.37 (48.6)
SVR	99 (0.3)	99 (0.3)	0 (0)
Voting	28 (0.005)	28 (0.005)	71.28 (45.4)

6.2.3. Both Dataset

This dataset contains information about 49 projects each including 15 COCOMO attributes. By using PCA, the dimensions are reduced to 10. As given in Table 6.19, there are 6 effort classes for this dataset. Classification results and point estimation results for this dataset will be given in the following subsections.

6.2.3.1. Classification Results. Misclassification results of each classifier are given in Table 6.28. Here, we can see that KNN outperforms the others with 5.25 per cent misclassification rate whereas DT is the worst of all.

Table 6.28. Classification results for both

Classifier	Misclassification Rate (per cent)
DT	21
KNN	5.25
LD	8

6.2.3.2. Point Estimation Results. In the cross-validation framework, the estimators are applied just after the effort class is estimated by the classifiers. The effort estimation results for each classifier are given in the following tables.

Table 6.29. Point estimation results for both dataset with DT

Estimator	MdMRE	MMRE	PRED(25)
GS	90 (1.9)	258 (3.3)	41.58 (23.2)
KNN	33 (0.4)	69 (0.8)	59.65 (22.9)
LR	28 (0.4)	56 (0.6)	58.91 (24.6)
MLP	21 (0.2)	45 (0.5)	64.10 (25.5)
SVR	99 (0)	99 (0)	0 (0)
Voting	39 (0.5)	86 (1.0)	46.53 (26.4)

When we look at the results of DT, we can see that all values for MdMRE and MMRE are below 100 per cent except those for SVR and GS. MLP gives the best values for all measures.

Table 6.30. Point estimation results for both dataset with KNN

Estimator	MdMRE	MMRE	PRED(25)
GS	84 (1.9)	253 (3.3)	48.51 (24.1)
KNN	28 (0.4)	64 (0.8)	66.08 (21.9)
LR	21 (0.4)	50 (0.6)	70.29 (22.2)
MLP	14 (0.2)	39 (0.5)	73.51 (22.0)
SVR	95 (0)	99 (0)	0 (0)
Voting	35 (0.5)	83 (1.0)	58.41 (24.5)

When we look at the results of KNN, as in DT, all values for MdMRE and MMRE are below 100 per cent except those for SVR and GS. Again, MLP gives the best values for all measures.

Table 6.31. Point estimation results for both dataset with LD

Estimator	MdMRE	MMRE	PRED(25)
GS	85 (1.9)	254 (3.3)	49.50 (22.6)
KNN	29 (0.4)	66 (0.8)	64.60 (21.8)
LR	22 (0.4)	51 (0.6)	68.60 (22.5)
MLP	18 (0.2)	42 (0.5)	70.29 (21.9)
SVR	99 (0)	99 (0)	0 (0)
Voting	36 (0.5)	83 (1.01)	55.19 (22.9)

For LD classifier, we can see that all MdMRE values are below 100 per cent for all estimators. Also, except those for GS, MMRE values are below 100 per cent but are not low enough. Again, as in KNN and DT, MLP gives the best values for all measures.

6.2.4. ISBSG_e Dataset

This dataset contains information about 17 projects from different countries and industries each including 8 attributes. By using PCA, the dimensions are reduced to 5. As given in Table 6.19, there are 3 effort classes for this dataset. Classification results and point estimation results for this dataset will be given in the following subsections.

6.2.4.1. Classification Results. Misclassification results of each classifier are given in Table 6.32. Here, we can see that KNN outperforms the others with 0 per cent misclassification rate. LD outperforms DT with a slight difference which is 4 per cent.

Table 6.32. Classification results for ISBSG_e

Classifier	Misclassification Rate (per cent)
DT	20
KNN	0
LD	16

6.2.4.2. Point Estimation Results. In the cross-validation framework, the estimators are applied just after the effort class is estimated by the classifiers. The effort estimation results for each classifier are given in the following tables. When we perform cross-validation, the validation set consists of 2 projects for *ISBSG_e*, thus, the results for MdMRE and MMRE measures are the same for each classifier.

Table 6.33. Point estimation results for ISBSG_e dataset with DT

Estimator	MdMRE	MMRE	PRED(25)
GS	584 (13.1)	584 (13.1)	29.70 (45.9)
KNN	483 (11.1)	483 (11.1)	61.38 (48.9)
LR	67 (1.7)	67 (1.7)	79.20 (40.7)
MLP	547 (12.8)	547 (12.8)	77.22 (42.1)
SVR	99 (0.001)	99 (0.001)	0 (0)
Voting	333 (7.7)	333 (7.7)	67.32 (47.1)

When we look at the results of DT, we can see that all values for MdMRE and MMRE are above 100 per cent except those for SVR and LR. LR gives the best values for all measures. PRED value is above the optimum value (75 per cent), but, MdMRE and MMRE are higher than the optimum value (25 per cent).

Table 6.34. Point estimation results for ISBSG_e dataset with KNN

Estimator	MdMRE	MMRE	PRED(25)
GS	39 (0.4)	39 (0.4)	41.58 (49.5)
KNN	24 (0.3)	24 (0.3)	73.26 (44.4)
LR	1 (0.02)	1 (0.02)	99.01 (9.95)
MLP	21 (0.3)	21 (0.3)	83.16 (37.6)
SVR	99 (0.001)	99 (0.001)	0 (0)
Voting	18 (0.1)	18 (0.1)	67.32 (47.1)

When we look at the results of KNN, all values for MdMRE and MMRE are below 100 per cent and PRED values are approximate to the optimum value (75 per cent). LR gives the best values for all measures.

For LD classifier, shown in Table 6.34, the results are similar to those when KNN is used. Again, LR gives the best values for all measures and the values are approximate to the optimum values.

Table 6.35. Point estimation results for ISBSG_e dataset with LD

Estimator	MdMRE	MMRE	PRED(25)
GS	49 (0.6)	49 (0.6)	49.50 (50.2)
KNN	34 (0.4)	34 (0.4)	65.34 (47.8)
LR	32 (0.9)	32 (0.9)	83.16 (37.6)
MLP	32 (0.4)	32 (0.4)	70.29 (45.9)
SVR	99 (0.001)	99 (0.001)	0 (0)
Voting	33 (0.3)	33 (0.3)	53.46 (50.1)

6.2.5. Comparison of WoCAD with COCLAM

In COCLAM, our aim is to convert effort estimation problem into a classification problem in order to build a better predictor for embedded software. We can measure the effectiveness of our classification model by looking at the misclassification rates. The smaller the misclassification rate is, the better the classification performance is. A summary of the classification results obtained are given in Table 6.36.

Table 6.36. Summary of the classification results

Dataset	Misclassification Rates (per cent)		
	DT	KNN	LD
coc81_e	19	5	4.5
nasa93_e	30	0	26
both	21	5.25	8
ISBSG_e	20	0	16

When we look at these results, we can see that the misclassification rate is 0 per cent in the best case and 30 per cent in the worst case. This means that our proposed model performs well as a classification model on embedded datasets. If we look at the classifiers separately, it seems that KNN is the best one giving 0 per cent misclassification rate for two cases, and LD is the second one whereas DT is the worst one.

Since our main aim in COCLAM is to build a better predictor for embedded software cost estimation, we should also check if our proposed model really improves the performance. We can check this by comparing the point estimation results of COCLAM to those of WoCAD. For this purpose, we should find out the best performing classifier for each dataset. As done in WoCAD for comparing the setups, we perform *t-tests* between different sets of results of each algorithm. While performing the t-tests, we used 0.05 significance level (□=0.05). According to the t-tests performed, the best performing classifier obtained for each estimator are given for *coc81_e* dataset in Table 6.37.

Table 6.37. T-test results for coc81_e dataset

Estimator	MdMRE	MMRE	PRED(25)
SVR	KNN=LD=DT	KNN=LD=DT	KNN=LD=DT
GS	KNN=LD=DT	KNN=LD=DT	LD
LR	KNN=LD=DT	KNN=LD=DT	LD
MLP	DT	DT	LD
KNN	KNN=LD=DT	KNN=LD=DT	LD
Voting	KNN=LD=DT	KNN=LD=DT	KNN

When we look at the results, all of the classifiers perform statistically similar for most of the estimators given in Table 6.37. For four cases, LD outperforms the others, so, we can use LD while comparing COCLAM results with those of WoCAD. The best performing classifiers for *nasa93_e* dataset are given in Table 6.38. When we look at the results, KNN is the best classifier for all estimators except SVR. For SVR, all classifiers produce results that are not statistically different.

Table 6.38. T-test results for nasa93_e dataset

Estimator	MdMRE	MMRE	PRED(25)
SVR	KNN=LD=DT	KNN=LD=DT	KNN=LD=DT
GS	KNN	KNN	KNN
LR	KNN	KNN	KNN
MLP	KNN	KNN	KNN
KNN	KNN	KNN	KNN
Voting	KNN	KNN	KNN

The best performing classifiers for *both* dataset are given in Table 6.39. Again for SVR, the results of classifiers are not statistically different. For the remaining estimators,

DT is the best performing classifier.

Table 6.39. T-test results for both dataset

Estimator	MdMRE	MMRE	PRED(25)
SVR	KNN=LD=DT	KNN=LD=DT	KNN=LD=DT
GS	DT	DT	LD
LR	DT	DT	LD
MLP	KNN	DT	LD
KNN	DT	DT	LD
Voting	DT	DT	LD

The best performing classifiers for *ISBSG_e* dataset are given in Table 6.40. When we look at the results, we can see that KNN is the best performing classifier for all algorithms except SVR and GS.

Table 6.40. T-test results for ISBSG_e dataset

Estimator	MdMRE	MMRE	PRED(25)
SVR	KNN=LD=DT	KNN=LD=DT	KNN=LD=DT
GS	LD	LD	LD
LR	LD	LD	KNN
MLP	KNN	KNN	KNN
KNN	KNN	KNN	KNN
Voting	KNN	KNN	DT

As a result, for two of the four datasets (*nasa93_e* and *ISBSG_e*), KNN is the best classifier while producing the point estimation results. For *coc81_e,* LD can be chosen as the best classifier whereas DT is the best one for *both*.

While comparing the point estimation performance of COCLAM to that of WoCAD, for each dataset, *t-tests* are applied to the results of the best classifier in COCLAM and the results of the best setup in WoCAD. The t-test results are given for *coc81_e* in Table 6.41. Here, the null hypothesis is that WoCAD has better value than COCLAM for current estimator and current measure. The result is 1 if you can reject the null hypothesis at the 0.05 significance level and 0 otherwise.

Table 6.41. Comparison of WoCAD with COCLAM for coc81_e dataset

Estimator	MdMRE	MMRE	PRED(25)
SVR	1	1	0
GS	0	1	1
KNN	0	1	1
LR	1	1	1
MLP	1	1	1
Voting	1	1	1

From the results, we can see that for all estimators, COCLAM outperforms WoCAD. Only for GS and LR, WoCAD results are better than COCLAM for MdMRE measure and for SVR for PRED measure.

The t-test results are given for *nasa93_e* in Table 6.42. From the results, we can see that for all estimators, COCLAM outperforms WoCAD. Only for SVR, WoCAD results are better than COCLAM for PRED measure.

Table 6.42. Comparison of WoCAD with COCLAM for nasa93_e dataset

Estimator	MdMRE	MMRE	PRED(25)
SVR	1	1	0
GS	1	1	1
KNN	1	1	1
LR	1	1	1
MLP	1	1	1
Voting	1	1	1

The t-test results are given for *both* in Table 6.43. From the results, we can see that for all estimators, COCLAM outperforms WoCAD. Only for GS and SVR, WoCAD results are better than COCLAM for MdMRE and PRED measures.

Table 6.43. Comparison of WoCAD with COCLAM for both dataset

Estimator	MdMRE	MMRE	PRED(25)
SVR	1	1	0
GS	0	1	1
KNN	1	1	1
LR	1	1	1
MLP	1	1	1
Voting	1	1	1

The t-test results are given for *ISBSG_e* in Table 6.44. From the results, we can see that for all estimators, COCLAM outperforms WoCAD. Only for SVR, WoCAD results are better than COCLAM for PRED measure.

Table 6.44. Comparison of WoCAD with COCLAM for ISBSG_e dataset

Estimator	MdMRE	MMRE	PRED(25)
SVR	1	1	0
GS	1	1	1
KNN	1	1	1
LR	1	1	1
MLP	1	1	1
Voting	1	1	1

The summary of the comparison results with the best values for each measure is given in Table 6.45. As we see from the results, all point estimation results obtained after classification in COCLAM outperforms those obtained in WoCAD. This means that when the estimators are trained on the projects that belong to the effort class found after classification, higher accuracies are obtained which is our main aim in COCLAM. Thus, we can conclude that our proposed model for effort classification is a better predictor for embedded software cost estimation. In addition, we can say that our classification model improves accuracy since it gives lower MMRE and MdMRE values. Our model is also robust as it gives high PRED values which increase up to 80 per cents and 90 per cents.

Table 6.45. Summary of the comparison results

Dataset	Best Performing Model	MdMRE	MMRE	PRED
coc81_e	COCLAM	34	34	73.76
nasa93_e	COCLAM	8	8	95.05
both	COCLAM	14	39	73.51
ISBSG_e	COCLAM	1	1	99.01

6.3. Results for Further Experiments of COCLAM

In our experiments for COCLAM, we found out that our proposed model for embedded software cost estimation improves prediction accuracy on embedded datasets. In

order to investigate if our model can be used for datasets from other (non-embedded systems) domains and to provide external validity of our model, we test our model on five more datasets additionally. For this purpose, we use cocomonasa_v1, desharnais, cost_data, sdr, and sdr2 in our further experiments. All of these datasets contain data from different domains other than embedded systems domain. Both misclassification rates and MdMRE, MMRE, and PRED(25) values are calculated for each estimator. The numbers of effort clusters created for each dataset are given in Table 6.46.

Table 6.46. Clustering results for additional datasets

Dataset	Number of Clusters	Number of Projects	
		Minimum	Maximum
cocomonasa_v1	5	3	36
desharnais	9	2	21
cost_data	3	2	12
sdr	3	3	16
sdr2	4	6	16

6.3.1. Cocomonasa_v1 Dataset

This dataset contains information about 60 NASA projects each including 17 COCOMO attributes. By using PCA, the dimensions are reduced to 10. As given in Table 6.46, there are 5 effort classes for this dataset. Classification results and point estimation results for this dataset will be given in the following subsections.

6.3.1.1. Classification Results. Misclassification results of each classifier are given in Table 6.47. Here, we can see that KNN outperforms the others with 0.2 per cent misclassification rate. LD is the second best classifier whereas DT is the worst of all.

Table 6.47. Classification results for cocomonasa_v1

Classifier	Misclassification Rate (per cent)
DT	15.6
KNN	0.2
LD	4.4

6.3.1.2. Point Estimation Results. The effort estimation results for each classifier are given in the following tables separately. Standard deviations are given in parenthesis.

When we look at the results of DT shown below, we can see that all values for MdMRE and MMRE are below 100 per cent except for GS. KNN gives the best values for all measures.

Table 6.48. Point estimation results for cocomonasa_v1 with DT

Estimator	MdMRE	MMRE	PRED(25)
GS	61 (0.4)	123 (0.9)	27.12 (22.5)
KNN	23 (0.1)	28 (0.1)	54.45 (22.4)
LR	36 (0.1)	47 (0.2)	38.81 (18.9)
MLP	32 (0.2)	58 (0.5)	44.55 (18.5)
SVR	99 (0.0004)	99 (0.0004)	0 (0)
Voting	31 (0.1)	46 (0.2)	43.16 (21)

When we look at the results of KNN shown below, we can see that all values for MdMRE and MMRE are again below 100 per cent except those for GS. LR gives the best values for all measures.

Table 6.49. Point estimation results for cocomonasa_v1 with KNN

Estimator	MdMRE	MMRE	PRED(25)
GS	53 (0.4)	116 (0.9)	41.38 (24)
KNN	16 (0.09)	21 (0.1)	69.70 (21)
LR	25 (0.1)	39 (0.2)	51.88 (19)
MLP	21 (0.1)	53 (0.59)	59.80 (20)
SVR	99 (0.0004)	99 (0.0004)	0 (0)
Voting	25 (0.1)	42 (0.2)	52.67 (23)

According to the results of LD shown below, we can see that all values for MdMRE and MMRE are again below 100 per cent except those for GS. LR gives the best values for all measures.

Table 6.50. Point estimation results for coc81_e dataset with LD

Estimator	MdMRE	MMRE	PRED(25)
GS	56 (0.4)	118 (30)	37.62 (22)
KNN	17 (0.09)	22 (95)	68.11 (21)
LR	26 (0.1)	41 (12)	49.70 (18)
MLP	23 (0.1)	52 (25)	55.05 (20)
SVR	99 (0.0004)	99 (51)	0 (0)
Voting	27 (0.1)	44 (0.0004)	50.09 (21)

6.3.2. Desharnais Dataset

This dataset contains information about 77 projects each including 17 COCOMO attributes. By using PCA, the dimensions are reduced to 6. As given in Table 6.46, there are 9 effort classes for this dataset. Classification results and point estimation results for this dataset will be given in the following subsections.

6.3.2.1. Classification Results. Misclassification results of each classifier are given in Table 6.51. Here, we can see that KNN outperforms LD with a slight difference which is 2.57 per cent. DT is the worst of all.

Table 6.51. Classification results for desharnais

Classifier	Misclassification Rate (per cent)
DT	13.85
KNN	4
LD	6.57

6.3.2.2. Point Estimation Results. The effort estimation results for each classifier are given in the following tables separately. Standard deviations are given in parenthesis.

When we look at the results of DT, we can see that all values for MdMRE and MMRE are below 25 per cent except for SVR. KNN gives the best values for MMRE and PRED whereas LR gives the best MdMRE value.

Table 6.52. Point Estimation results for desharnais with DT

Estimator	MdMRE	MMRE	PRED(25)
GS	12 (0.07)	16 (0.05)	74.11 (17.5)
KNN	9 (0.03)	12 (0.03)	86.28 (13)
LR	7 (0.06)	13 (0.05)	79.06 (15)
MLP	10 (0.06)	17 (0.09)	78.07 (16)
SVR	99 (0)	99 (0)	0 (0)
Voting	22 (0.04)	24 (0.04)	60.82 (20)

When we look at the results of KNN, we can see that all values for MdMRE and MMRE are again below 25 per cent except those for SVR. As in DT, KNN gives the best values for MMRE and PRED whereas LR gives the best MdMRE value.

Table 6.53. Point estimation results for desharnais with KNN

Estimator	MdMRE	MMRE	PRED(25)
GS	10 (0.06)	13 (0.06)	78.92 (17)
KNN	7 (0.02)	9 (0.03)	91.08 (13)
LR	4 (0.04)	10 (0.05)	85.57 (15)
MLP	7 (0.04)	14 (0.09)	84.44 (16)
SVR	99 (0)	99 (0)	0 (0)
Voting	22 (0.03)	22 (0.04)	67.18 (20)

As in the previous results, we can see that all values for MdMRE and MMRE are again below 25 per cent except those for SVR. LR gives the best MMRE whereas KNN gives the best PRED values. For MMRE measure, KNN and LR give the same value which is 10 per cent.

Table 6.54. Point estimation results for desharnais with LD

Estimator	MdMRE	MMRE	PRED(25)
GS	10 (0.06)	14 (0.06)	79.49 (17)
KNN	7 (0.03)	10 (0.04)	89.10 (14)
LR	5 (0.04)	10 (0.05)	85.71 (15)
MLP	8 (0.06)	15 (0.1)	81.75 (17)
SVR	99 (0)	99 (0)	0 (0)
Voting	21 (0.03)	22 (0.03)	70.15 (18)

6.3.3. Cost_data Dataset

This dataset contains information about 24 projects each including 22 COCOMO attributes. By using PCA, the dimensions are reduced to 10. As given in Table 6.46, there are 3 effort classes for this dataset. Classification results and point estimation results for this dataset will be given in the following subsections.

6.3.3.1. Classification Results. Misclassification results of each classifier are given in Table 6.55. Here, we can see that KNN outperforms LD giving 0 per cent misclassification rate. DT follows KNN and LD is the worst of all.

Table 6.55. Classification results for cost_data

Classifier	Misclassification Rate (per cent)
DT	6.5
KNN	0
LD	12.5

6.3.3.2. Point Estimation Results. The effort estimation results for each classifier are given in the following tables separately. Standard deviations are given in parenthesis. Since there are 24 projects in *cost_data*, when we perform cross-validation, the validation set consists of 2 projects. Thus, the results for MdMRE and MMRE measures are the same for each classifier because the mean and median of any two MRE values are the same.

Table 6.56. Point estimation results for cost_data with DT

Estimator	MdMRE	MMRE	PRED(25)
GS	103 (0.6)	103 (0.6)	18.81 (26.3)
KNN	24 (0.1)	24 (0.1)	58.41 (34.6)
LR	6 (0.08)	6 (0.08)	92.57 (19.2)
MLP	17 (0.1)	17 (0.1)	74.25 (31.3)
SVR	98 (0.007)	98 (0.007)	0 (0)
Voting	25 (0.1)	25 (0.1)	50.99 (33)

When we look at the results of DT, we can see that LR gives the best values for all measures. MdMRE and MMRE values are around 25 per cent for all estimators except GS and SVR.

When we look at the results of KNN shown below, we can see that again LR gives the best values for all measures. The results are similar to those when DT is used, except that PRED values are higher.

Table 6.57. Point estimation results for cost_data with KNN

Estimator	MdMRE	MMRE	PRED(25)
GS	99 (0.6)	99 (0.6)	25.24 (29)
KNN	21 (0.1)	21 (0.1)	64.85 (34)
LR	4 (0.04)	4 (0.04)	99 (9.9)
MLP	12 (0.09)	12 (0.09)	84.15 (25.4)
SVR	98 (0.007)	98 (0.007)	0 (0)
Voting	21 (0.09)	21 (0.09)	57.42 (32.7)

When we look at the results of LD shown below, we can see that all values for MdMRE and MMRE are again above 25 per cent except those for LR. PRED values are high for all estimators except those for SVR and GS. LR gives the best values for all measures.

Table 6.58. Point estimation results for cost_data with LD

Estimator	MdMRE	MMRE	PRED(25)
GS	122 (0.06)	122 (0.06)	17.32 (17)
KNN	45 (0.03)	45 (0.03)	60.39 (14)
LR	14.02 (0.04)	14.02 (0.04)	87.12 (15)
MLP	40 (0.06)	40 (0.06)	72.77 (17)
SVR	98 (0)	98 (0)	0 (0)
Voting	309 (0.03)	309 (0.03)	53.46 (18)

6.3.4. Sdr Dataset

This dataset contains information about 25 projects each including 22 COCOMO attributes. By using PCA, the dimensions are reduced to 11. As given in Table 6.46, there are 3 effort classes for this dataset. Classification results and point estimation results for this dataset will be given in the following subsections.

6.3.4.1. Classification Results. Misclassification results of each classifier are given in Table 6.59. Here, we can see that KNN outperforms others by giving 0 per cent misclassification rate. LD follows KNN and DT is the worst of all.

Table 6.59. Classification results for sdr

Classifier	Misclassification Rate (per cent)
DT	19
KNN	0
LD	6

6.3.4.2. Point Estimation Results. The effort estimation results for each classifier are given in the following tables separately. Standard deviations are given in parenthesis. Since validation set consists of 2 projects, as in *cost_data*, the results for MdMRE and MMRE measures are the same for each classifier.

Table 6.60. Point estimation results for sdr with DT

Estimator	MdMRE	MMRE	PRED(25)
GS	71 (0.5)	71 (0.5)	34.15 (33)
KNN	55 (0.4)	55 (0.4)	42.57 (35)
LR	113 (2.8)	113 (2.8)	59.40 (35)
MLP	40 (0.4)	40 (0.4)	52.47 (38)
SVR	99 (0.001)	99 (0.001)	0 (0)
Voting	54 (0.7)	54 (0.7)	40.59 (36)

When we look at the results of DT, we can see that MLP gives the best values for MdMRE and MMRE measures whereas LD gives the best value for PRED.

Table 6.61. Point estimation results for sdr with KNN

Estimator	MdMRE	MMRE	PRED(25)
GS	51 (0.4)	51 (0.4)	51.98 (35)
KNN	35 (0.2)	35 (0.2)	61.38 (33)
LR	19 (0.1)	19 (0.1)	78.21 (28)
MLP	19 (0.1)	19 (0.1)	69.30 (34)
SVR	99 (0.001)	99 (0.001)	0 (0)
Voting	26 (0.1)	26 (0.1)	55.44 (35)

When we look at the results of KNN, we can see that again LR gives the best values for all measures and these results are approximate to optimum values (25 per cent for MdMRE and MMRE, 75 per cent for PRED).

Table 6.62. Point estimation results for sdr with LD

Estimator	MdMRE	MMRE	PRED(25)
GS	75 (1.1)	75 (1.1)	50 (35)
KNN	73 (1.4)	73 (1.4)	59.40 (34)
LR	51 (1.0)	51 (1.0)	72.27 (31)
MLP	62 (1.4)	62 (1.4)	66.33 (30)
SVR	99 (0.002)	99 (0.002)	0 (0)
Voting	53 (0.9)	53 (0.9)	53.46 (36)

We can see that all values for MdMRE and MMRE are all above 25 per cent. PRED values are high for all estimators. LR again gives the best values for all measures.

6.3.5. Sdr2 Dataset

This dataset contains information about 40 projects each including 22 COCOMO attributes. By using PCA, the dimensions are reduced to 11. As given in Table 6.46, there are 4 effort classes for this dataset. Classification results and point estimation results for this dataset will be given in the following subsections.

6.3.5.1. Classification Results. Misclassification results of each classifier are given in Table 6.63. Here, we can see that KNN and DT perform similar by giving 6 per cent misclassification rate. LD is the worst of all with 4 per cent difference.

Table 6.63. Classification results for sdr2

Classifier	Misclassification Rate (per cent)
DT	6
KNN	6
LD	11

6.3.5.2. Point estimation results. The effort estimation results for each classifier are given in the following tables separately. Standard deviations are given in parenthesis.

Table 6.64. Point estimation results for sdr2 with DT

Estimator	MdMRE	MMRE	PRED(25)
GS	20 (0.08)	20 (0.06)	66.99 (25)
KNN	9 (0.07)	11 (0.06)	83.82 (21)
LR	1 (0.02)	3 (0.03)	96.04 (13)
MLP	6 (0.04)	8 (0.04)	88.77 (19)
SVR	99 (0)	99 (0)	0 (0)
Voting	16 (0.05)	17 (0.04)	81.51 (21)

When we look at the results of DT, we can see that LD gives the best values for all measures. All values for MdMRE, MMRE, and PRED are approximate to the optimum values for all estimators except SVR.

Table 6.65. Point estimation results for sdr2 with KNN

Estimator	MdMRE	MMRE	PRED(25)
GS	20 (0.08)	20 (0.06)	66.99 (25)
KNN	9 (0.07)	11 (0.06)	83.82 (21)
LR	1 (0.02)	3 (0.03)	96.04 (13)
MLP	6 (0.04)	8 (0.04)	89.43 (18)
SVR	99 (0)	99 (0)	0 (0)
Voting	16 (0.05)	17 (0.04)	81.51 (22)

When we look at the results of KNN, we can see that again LR gives the best values for all measures and these results are very good when compared with the optimum values (25 per cent for MdMRE and MMRE, 75 per cent for PRED).

Table 6.66. Point estimation results for sdr2 with LD

Estimator	MdMRE	MMRE	PRED(25)
GS	23 (0.1)	27 (0.1)	59.07 (28)
KNN	10 (0.1)	15 (0.1)	72.93 (24)
LR	2 (0.09)	9 (0.1)	92.07 (18)
MLP	7 (0.09)	11 (0.1)	84.48 (23)
SVR	99 (0)	99 (0)	0 (0)
Voting	17 (0.07)	18 (0.07)	80.85 (22)

We can see that all values for MdMRE and MMRE are all around 25 per cent except those for SVR. PRED values are high for all estimators except again SVR. LR gives the best values for all measures.

The summary of the further results for COCLAM is given in Table 6.67. When we look at the results for all five datasets, it is obvious that misclassification rates are very low, around 19 per cent in the worst case and 0 per cent in the best case. This shows that the classifiers used are really effective. For all datasets, MdMRE and MMRE values decrease below 10 per cents in best cases, which mean that our proposed model achieves high accuracy. For PRED measures, the best values increase up to 90 per cents which shows the robustness of our proposed model. The standard deviations are around 30s, which means that the proposed model obtain consistent results. As a result, we can conclude that our effort classification model proposed for COCLAM is externally valid and can be applied to datasets from other domains.

Table 6.67. Summary of the further results of COCLAM

Dataset	Misclassification Rate (per cent)	Best Values		
		MdMRE	MMRE	PRED
cocomonasa_v1	0.2 - 15.6	16	21	69.70
desharnais	4 – 13.85	4	9	91.08
cost_data	0 – 12.5	4	4	99
sdr	0 - 19	19	19	78.21
sdr2	6 - 11	1	3	96.04

6.4. Comparison with Other Studies

For embedded software cost estimation, in literature, there is not any study that we can compare our results with. Thus, we think that we can compare our results with COCOMO model since we use three embedded datasets (*coc81_e*, *nasa93_e*, and *both*) that contains COCOMO-based attributes. By making such a comparison, we will also see how our model performs against an independent model. In Table 6.68, we present the COCOMO results and the best effort estimation results obtained with our classification model for each dataset. Best values for each dataset are marked with * for clarity.

Table 6.68. Comparison of our classification model with COCOMO

Datasets	MdMRE		MMRE		PRED(25)	
	Cocomo	COCLAM	Cocomo	COCLAM	Cocomo	COCLAM
coc81_e	31*	34	31*	35	54.95*	52.97
nasa93_e	81	13*	81	13*	38.11	87.12*
both	30	14*	53	39*	46.78	73.51*

According to this table, for *nasa93_e* and *both* datasets, our classification model outperforms COCOMO model with great differences for all measures. For *coc81_e*, COCOMO results are slightly better than those of our model with 2 per cent and 3 per cent difference. Although COCOMO model was calibrated by using *coc81* dataset, the results it gives for *coc81_e* are not pretty good. The reason may be that COCOMO coefficients were calculated by using all of the projects in *coc81* and may not be suitable for a subset of the projects from the same dataset.

While developing our effort classification model, our main aim is to predict effort classes with high accuracy. We showed that, in previous sections, the classification performance of our model is pretty good as it gives misclassification rates changing between 0 per cent and 5.25 per cent for embedded datasets and between 0 per cent and 6 per cent for additional datasets. However, we need to compare our model to those in other studies in order to say that our model really performs well. For comparison, we can use the studies that perform interval estimation since our classification model classifies projects into effort classes each corresponding to an effort interval. In the literature, the most recent study on this topic is Sentas et al.'s study [28]. In this study, hit rates around 70 per cents

are obtained for productivity interval estimation on *coc81* dataset. We should remember that hit rate is the 100-complement of misclassification rate. Thus, we can say that the hit rates for our study are between 90 per cents and 100 per cent which are higher than those obtained in Sentas et al.' study. Furthermore, the intervals in that study are manually pre-defined, whereas they are dynamically created in our model. In Table 6.69, we compare our results with that of Sentas et.al.

Table 6.69. Comparison of our results with Sentas et al.'s

	Hit rate (per cent)	
	Min	Max
Sentas et al.	60.38	79.24
Our model	94.75	100

7. CONCLUSIONS

In this research, we focus on embedded software cost estimation which gained importance after embedded systems has become an important part of our lives. We proposed solutions for both aspects of improving prediction accuracy: understanding the data better and modeling a better predictor.

In data aspect, we investigated *what type of* and *how much* training data should be used for embedded software cost estimation by introducing the notion of *application domain*. We compared *within-application domain* and *cross-application domain* datasets in order to find out what type of training data should be used. For this comparison, a number of machine learning methods were used in different experimental setups. According to the experiments done, we can conclude that cross-domain datasets should be used for training estimators in embedded software cost estimation. In order to find *how much* training data should be used, we performed additional experiments and we observed that there is not a direct relationship between training set size and performance. We can only suggest that trying all possible training set sizes and then selecting the best one to use in experiments.

In model aspect, we investigated if we can obtain higher accuracy by converting cost estimation into a classification problem. For this purpose, we developed a model that classifies software projects in one of the dynamically created effort classes each corresponding to an effort interval. Our proposed model is also able to produce point estimations by using the experimental design we developed. The design utilizes a number of estimators that are trained on the projects that belong to the resulting effort class. According to the experiments done, we can conclude that embedded software cost estimation accuracy increases by effort classification. In order to make our results externally valid, additional experiments are performed with datasets from other domains and again we observed that our model gives accurate results.

When we compare our results with those stated in other studies, we see that our proposed model produces better results in term of both effort class prediction and effort prediction.

7.1. Contributions

The main contribution of this research in academia is that it is one of a kind initial study on embedded software cost estimation. Our experimental design as well as wide range of learning-based models used may inspire and guide other researchers who would be conducting research on this domain. Another academic contribution of our research is that our novel approach of treating cost estimation as a pure classification problem in order to estimate effort intervals.

Software engineering practitioners may also use our results to make important decisions in managing their projects. For example, our comprehensive analysis of data usage for cost estimation in embedded software development domain may guide software engineers to decide which data to use: cross-domain or within-domain. This especially very critical for the companies that are newly-established or that do not have enough historical data. In such a case they can use the cross-domain datasets for their cost estimation studies. Secondly, our proposed model that is based on classification can be used by the project managers in order to make more accurate estimates while bidding for a new project or allocating the resources among different projects.

7.2. Future Work

As a future research direction embedded systems datasets can be expanded to further test the sensitivity of our proposed algorithm for domain specific data. Moreover the data collection process in embedded systems domain may focus on searching for domain specific attributes so that the information content of the attributes becomes richer and as a result prediction performance of the algorithm improves.

Another future research may be using different clustering methods to find effort classes. Also different heuristics can be used for calculating the effort intervals for each cluster.

REFERENCES

1. *Hardware-Software Co-Design*, Design of Embedded Systems Presentation #6, Vienna University of Technology, http://www.ict.tuwien.ac.at/skripten/Design EmbeddedSystems/DES per cent20Fall per cent202006 per cent2006.pdf

2. Hosagrahara, A. and P. Smith, *Measuring Productivity and Quality in Model-Based Design*, MATLAB Digest, March 2006, http://www.mathworks.com/company/ newsletters/digest/2006/mar/measuringprod.html

3. Debardelaben, J. A., V. K. Madisetti and A. J. Gadient, "Incorporating Cost Modeling in Embedded-System Design"*, IEEE Design&Test of Computers*, Vol. 14, No. 3, pp.24-35, July-September 1997.

4. Lee, E. A., *Embedded Software,* Advances in Computers, Vol. 56, Academic Press, London, 2002.

5. Leung, H. and Z. Fan, *Software Cost Estimation*, Handbook of Software Engineering.

6. Nelson, E., *Management Handbook for the Estimation of Computer Programming Costs*, Systems Development Corporation, October 1966.

7. Putnam, L. and W. Myers, *Measures for Excellence*, Yourdon Press Computing Series, 1992.

8. Jones, C., *Applied Software Measurement*, McGraw Hill, 1997.

9. Park, R, *The Central Equations of the PRICE Software Cost Model*, 4th COCOMO Users' Group Meeting, November 1988.

10. Jensen, R., "An Improved Macrolevel Software Development Resource Estimation Model", *Proceedings 5th ISPA Conference*, pp. 88-92, April 1983.

11. Boehm, B., *Software Engineering Economics*, Prentice Hall, 1981.

12. Boehm, B., C. Abts and S. Chulani, "Software development cost estimation approaches – A survey"*, Annals of Software Engineering*, Vol. 10, Issue 1-4, pp. 177-205, 2000.

13. Kitchenham, B. A., E. Mendes and G. H. Travassos, "Cross- vs. Within-Company Cost Estimation Studies: A Systematic Review", I*EEE Transactions on Software Engineering*, Vol. 33, Issue 5, 316-329, 2007.

14. Premraj, R. and T. Zimmerman, "Building Software Cost estimation Models Using Homogenous Data", *First International Symposium on Empirical Software Engineering and Measurement*, ESEM 2007.

15. Ohsugi, N., A. Monden, N. Kikuchi, M. D. Barker, M. Tsunoda, T. Kakimoto and K. Matsumoto, "Is This Cost Estimate Reliable? – The Relationship between Homogeneity of Analogues and Estimation Reliability", *First International Symposium on Empirical Software Engineering and Measurement*, ESEM 2007.

16. Vahid, F. and T. D. Givargis, *Embedded System Design: A Unified Hardware/Software Introduction*, John Wiley & Sons, 2002.

17. J´unior, M. N. O., P. R. M. Macielet, R. S. Barreto and F. F. Carvalho, "Towards a Software Power Cost Analysis Framework Using Colored Petri Net"*, PATMOS 2004*, pp. 362-371.

18. Ragan, D., P. Sandborn and P. Stoaks, "A Detailed Cost Model for Concurrent Use with Hardware/Software Co-Design"*, DAC 2002*, ACM, pp. 269-274.

19. Tiwari, V., S. Malik and A. Wolfe, "Power Analysis of Embedded Software: A First Step towards Software Power Minimization"*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 2, Issue 4, pp. 437-445.

20. Zotos, K., A. Litke, A. Chatzigeorgiou, S. Nikolaidis and G. Stephanides, *Energy Complexity of Software in Embedded Systems*, ACIT-Automation, Control, and Applications, 2005.

21. Draper, N. and H. Smith, *Applied Regression Analysis*, Wiley, 1981.

22. Miyazaki, Y., M. Terakado, K. Ozaki and H. Nozaki, "Robust Regression for developing Software Estimation Models", *Journal of Systems and Software*, Vol.1, pp. 3-16, 1994.

23. Başkeleş, B., B. Turhan and A. Bener, "Software Effort Estimation Using Machine Learning Methods", *ISCIS 2007*.

24. Srinivasan, K. and D. Fisher, "Machine Learning Approaches to Estimating Software Development Effort", *IEEE Transactions on Software Engineering*, Vol. 21, Issue 2, pp. 126-137, 1995.

25. Briand, L.C., V. R. Basili and W.M. Thomas, "A Pattern Recognition Approach for Software Engineering Data Analysis", *IEEE Transactions on Software Engineering*, Vol. 18, Issue 11, pp. 931-942, 1992.

26. Boetticher, G. D., "Using Machine Learning to Predict Project Effort: Empirical Case Studies in Data-Starved Domains", *Model Based Requirements Workshop*, pp. 17–24, 2001.

27. Sentas, P., L. Angelis and I. Stamelos, "Multinominal Logistic Regression Applied on Software Productivity Prediction", *PCI 2003,* 9th Panhellenic Conference in Informatics, Thessaloniki.

28. Sentas, P., L. Angelis, I. Stamelos and G. Bleris, "Software Productivity and Effort Prediction with Ordinal Regression", *Information and Software Technology,* No. 47*,* pp. 17-29, 2005.

29. Angelis, L. and I., Stamelos, "A Simulation Tool for Efficient Analogy Based Cost Estimation", *Empirical Software Engineering*, No. 5, pp. 35-68, 2000.

30. Stamelos, I. and L. Angelis, "Managing Uncertainty in Project Portfolio Cost Estimation", *Information and Software Technology*, Vol. 43, No. 13, pp. 759-768, 2001.

31. Stamelos, I., L. Angelis, P. Dimou and E. Sakellaris, "On the Use of Bayesian Belief Networks for the Prediction of Software Productivity", *Information and Software Technology*, Vol. 45, pp. 51-60, 2003.

32. Jorgensen, M., "An Effort Prediction Interval Approach Based on the Empirical Distribution of Previous Estimation Accuracy", *Information and Software Technology*, Vol. 45, pp. 123-126, 2003.

33. Debardelaben, J.A., V.K. Madisetti and A.J. Gadient, "Incorporating Cost Modeling in Embedded-System Design"*, IEEE Design & Test of Computers*, Vol. 14, Issue 3, pp. 24-35, 1997.

34. *EstimatorPal*, http://software.techrepublic.com.com/download.aspx?docid=236622

35. *Igoodsoft*, http://www.igoodsoft.com/sesdevelopment.asp

36. *SCEP Software Cost Estimation Program*, http://www.retisoft.com/Products.html

37. Boehm, B., B. Clark, E. Horowitz, C. Westland, R. Madachy and R. Selby, "Cost Models for Future Software Life-cycle Processes: COCOMO 2.0", *Annals of Software Engineering Special Volume on Software Process and Product Measurement*, J.D. Arthur and S.M. Henry (Eds.), J.C. Baltzer AG, Science Publishers, Amsterdam, TheNetherlands, Vol 1, pp. 45-60, 1995.

38. Jorgensen, M. and K. H. Teigen, "Uncertainty Intervals versus Interval Uncertainty: An Alternative method for Eliciting Effort Prediction Intervals in Software

Development Projects", *International Conference on Project Management (ProMAC)*, Singapore, pp. 343-352, 2002.

39. Bibi, S., I. Stamelos and L. Angelis, "Software Cost Prediction with predefined Interval Estimates", *Proceedings of the 1st Software Measurement European Forum*, pp. 237-246, 2004.

40. *USC COCOMO II Model Definition Manual*, University of Southern California, 1998.

41. *Cost Estimation Models*, http://yunus.hacettepe.edu.tr/~sencer/cocomo.html

42. Banker, R., R. Kauffman and R. Kumar, "An Empirical Test of Object-Based Output Measurement Metrics in a Computer Aided Software Engineering (CASE) Environment", *Journal of Management Information System*, 1994.

43. Albrecth, A. J. and J. E. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", *IEEE Transactions on Software Engineering*, Vol.9, No. 6, pp. 639-648, 1983.

44. Kauffman, R. and R. Kumar, *Modeling Estimation Expertise in Object Based ICASE Environments*, Stern School of Business Report, New York University, January 1993.

45. Alpaydin, E., *Introduction to Machine Learning*, MIT Press, 2004.

46. Mason, A. K and N. Sweeney, "Parametric Cost Estimating with Limited Sample Sizes", *Proceedings of the Third Annual Artificial Intelligence Symposium*, 1992.

47. Perel, R. J., "Mold Cost Estimator Generator Utilizing Standard Data and Linear Regression", *Proceedings of the Regional Technical Conference of the Society of Plastic Engineers,* GI-G19, 1994.

48. Shepperd, M., C. Schofield and B. Kitchenham, "Effort Estimation Using Analogy", *IEEE*, Proceedings of ICSE-18, 1996.

49. Gray, A. and S. MacDonell, "A Comparison of Techniques for Developing Predictive Models of Software Metrics", *Information and Software Technology,* Vol. 39, 1997.

50. *SoftLab*, Software Research Laboratory, Department of Computer Engineering, Bogazici University, http://softlab.boun.edu.tr

51. Boetticher, G., T. Menzies and T. Ostrand, *PROMISE Repository of Empirical Software Engineering Data*, http://promisedata.org/repository, West Virginia University, Department of Computer Science, 2007.

52. C. Lokan, T. Wright, P. R. Hill and M. Stringer, "Organizational Benchmarking Using the ISBSG Data Repository", *IEEE Software*, Vol. 18, No.5, pp. 26-32, 2001

53. *COCOMO II Cost Estimation Questionnaire*, http://sunset.usc.edu/research/ COCOMOII/Docs/cform22.pdf, 2006.

54. Desharnais, J. M., *Analyse Statistique de la Productivitie des Projets Informatique a Partie de la Technique des Point des Function*, Masters Thesis, University of Montreal, 1989.

55. Smola, A. J. and B. Schölkopf, *A Tutorial on Support Vector Regression*, NeuroCOLT Technical Report, TR-98-030, September, 2003.

56. Gunn, S. R., *Support Vector Machines for Classification and Regression*, Technical Report, Image Speech and Intelligent Systems Research Group, University of Southampton, 1997.

57. Brierley, P., http://www.philbrierley.com/main.html?code/matlab.html&code/codeleft .html

58. Fausett, L., *Fundamentals of Neural Networks,* Prentice Hall, 1994.

59. Alpaydin, E., "Technique for Combining Multiple Learners", *Engineering of Intelligent System '98 Conference*, Vol 2, pp. 6-12, 1998.

60. Lee, A., C. H. Cheng and J. Balakrishnan, "Software Development Cost Estimation: Integrating Neural Network with Cluster Analysis", *Information & Management*, Vol. 34, pp. 1-9, 1998.

61. Gallego, J. J. C., D. Rodriguez, M. A. Sicilia, M. G. Rubio and A. G. Crespo, "Software Project Effort Estimation Based on Multiple Parametric Models Generated through Data Clustering", *Journal of Computer Science and Technology*, Vol. 22, No. 3, pp. 371-378, 2007.

62. Bakar, Z. A., M. M. Deris and A. C. Alhadi, "Performance Analysis of Partitional and Incremental Clustering", *Seminar Nasional Aplikasi Teknologi Informasi,* SNATI 2005.

63. Quinlan, J. R., *C4.5: Programs for Machine Learning*, Morgan Kaufman, 1993.

64. Shalabi, L. A. and Z. Shaaban, "Normalization as a Preprocessing Engine for Data Mining and the Approach of Preference Matrix", *IEEE Proceedings of the International Conference on Dependability of Computer Systems*, DEPCOS-RELCOMEX'06, 2006.

65. *MATLAB*, www.mathworks.com

66. Conte, S. D., H. E. Dunsmore and V. Y. Shen, *Software Engineering Metrics and Models*, Benjamin-Cummings, Menlo Park CA, 1986.

67. Foss, T., E. Stensrud, B. Kitchenham and I. Myrtveit, "A Simulation Study of the Model Evaluation Criteria MMRE", *IEEE Transactions on Software Engineering*, Vol. 29, No. 11, pp 985-995, 2003.

68. Korte, M. and D. Port, “Confidence in Software Cost Estimation Results”, *PROMISE’2008*, pp. 63-70, 2008.

69. Myrtveit, I. and E. Stensrud, “A Controlled Experiment to Assess the Benefits of Estimating with Analogy and Regression Models”, *IEEE Transactions on Software Engineering*, Vol. 25, No. 4, 510-525, 1999.

70. Stensrud, E. and I. Myrtveit, “Human Performance Estimating with Analogy and Regression Models: An Empirical Validation”, *Proceedings of 5th International Metrics Symposium*, Bethesda, MD: IEEE Computer Society, 1998.

71. Bakir, A., B. Turhan and A. Bener, “Software Cost Estimation as a Classification Problem”, *To appear in the Proceedings of ICSOFT’08*, 2008, Portugal.

72. Giorgio Ingargiola, *Building Classification Models: ID3 and C4.5*, Computer and Information Science Department, Temple University, http://www.cis.temple.edu /~ingargio/cis587/readings/id3-c45.html

73. Bakir, A., B. Turhan and A. Bener, “Software Effort Prediction through Dynamic Intervals”, *Submitted to ESEM 2008,* Germany.

74. Turhan, B., A. Bakir and A. Bener, "A Comparative Study for Estimating Software Development Effort Intervals", *Submitted to Knowledge Based Systems Journal*, 2008.