Software Qual J (2010) 18:57–80
DOI 10.1007/s11219-009-9081-z

A new perspective on data homogeneity in software cost estimation: a study in the embedded systems domain

Ayşe Bakır · Burak Turhan · Ayşe B. Bener

Published online: 5 July 2009

Abstract Cost estimation and effort allocation are the key challenges for successful project planning and management in software development. Therefore, both industry and the research community have been working on various models and techniques to accurately predict the cost of projects. Recently, researchers have started debating whether the prediction performance depends on the structure of data rather than the models used. In this article, we focus on a new aspect of data homogeneity, "cross- versus within-application domain", and investigate what kind of training data should be used for software cost estimation in the embedded systems domain. In addition, we try to find out the effect of training dataset size on the prediction performance. Based on our empirical results, we conclude that it is better to use cross-domain data for embedded software cost estimation and the optimum training data size depends on the method used.

Keywords Application domain · Cost estimation · Data homogeneity · Embedded software · Machine learning

1 Introduction

Cost estimation is one of the critical steps of the software development life cycle as underestimation results in approving projects that would exceed their budgets, whereas overestimation results in wasting of resources (Leung and Fan 2001). Modeling accurate and robust software cost estimators is still a key challenge for managing successful

A. Bakır (✉) · A. B. Bener
Department of Computer Engineering, Boğaziçi University, 34342 Bebek, Istanbul, Turkey
e-mail: ayse.bakir@boun.edu.tr

A. B. Bener
e-mail: bener@boun.edu.tr

B. Turhan
Software Engineering Group, Institute for Information Technology, National Research Council of Canada, 1200 Montreal Road, Building M50, Ottawa, ON K1A0R6, Canada
e-mail: Burak.Turhan@nrc-cnrc.gc.ca

software projects. Thus, it preserves its popularity within the research community and many articles continue to be published, each of which introduces a new perspective on cost estimation (Kitchenham et al. 2007; Menzies 2007; Ohsugi et al. 2007; Premraj and Zimmermann 2007).

Recently, the focus of cost estimation studies has shifted from comparing a number of models on a specific dataset to investigating the relationship between the structure of cost data and the prediction accuracy (Kitchenham et al. 2007; Ohsugi et al. 2007; Premraj and Zimmermann 2007). Up to now, this issue has been considered as a comparison of either cross- versus within-company methods or cross- versus within-business domain methods and many studies reached different conclusions. In their article, Kitchenham et al. (2007) perform a systematic review of these studies and report that (1) in 3 of the 7 studies reviewed, cross-company models were not significantly different than within-company models, (2) in 4 of the 7 studies reviewed, within-company models were significantly better than cross-company models, (3) and in none of them, cross-company models were significantly better than within-company models. They further discuss that within-company models based on small datasets are better, since these datasets may be more homogeneous than large cross-company datasets in terms of incorporating similar projects. In contrast to these studies, we treat data homogeneity from an application domain point of view. By application domain, we mean domains such as embedded, real-time, desktop application, etc. and we focus on the embedded domain in this research. Our aim is to find out what type of data (*within-domain* data or *cross-domain* data) should be preferred for embedded software cost estimation. In addition, we aim to find out the effect of training dataset size on prediction performance.

Specifically, we set the limits of our research to the embedded software domain, because, in the last decade, development of multimedia and wireless applications on mobile devices have led to increasing interest in embedded systems (Vahid and Givargis 2002). We can see examples of these systems in every part of our lives; from cellular phones to automobile systems. These systems, which are formerly implemented in hardwired uni-processor architectures, now contain programmable multiprocessors which means increased complexity and cost. For this reason, cost-effective implementations of embedded systems that meet performance, functional, timing, and physical requirements becomes a challenge. This is the main reason why we focus on embedded software in this research.

Our main contribution is to investigate the homogeneity of cost data in terms of application domains, and to focus on the embedded domain. For this purpose, we design three experimental setups to compare models trained on cross-domain data with those trained on within-domain data. While the first experiment provides a baseline by using within-domain data, the second and third experiments deal with cross-domain data with different training data sizes. In each setup, various machine-learning techniques have been applied to make our results independent from the techniques used. Further, in the additional experiments conducted, we investigate the effect of training data size on prediction performance, an open question discussed in other studies. In our experiments, we use different datasets from public repositories so that other researchers can replicate and/or improve our results (Boetticher et al. 2007; SoftLab 2009).

The rest of the article is organized as follows: Section 2 gives some related work from the literature. Section 3 states our main problem and the proposed methodology with the experiments conducted, the datasets, the methods used, and the performance measures. Threats to validity in our experiments are also given in this section. In Section 4, the results obtained are given and in Section 5 evaluation of the results are made. Finally, conclusions and future work are given in Section 6.

2 Background

In general, there are three types of domains as specified by (Lokan et al. 2001):

a. Business domain: the main operation area of a company such as finance, medical, telecommunications, etc.
b. Application domain: the type of application being addressed by the project such as embedded, process control, information system, etc.
c. Organizational domain: the type of organization that submitted the project such as banking, manufacturing, and retail.

The latest trend in cost estimation is to concentrate on the homogeneity of cost data, in terms of these domains, rather than individual cost models built on the data. As an extension of Kitchenham et al.'s (2007) research on cross- versus within-company studies, Premraj and Zimmerman (2007) examine the homogeneity of cost data from the business domain point of view. By business domain, they mean the main operation area of a company such as finance, medical, telecommunications, etc. The idea is that companies can either use business specific data (*within-business* data) or data that belongs to other businesses (*cross-business* data). In their article, they reported that within-business domain data are better for developing estimation models on than cross-business domain data. In contrast to this study, our main aim is to compare cross- and within-domain data and try to find out what kind of data should be preferred for embedded software cost estimation.

Another study that focuses on data homogeneity is Ohsugi et al. (2007). In their hypothesis, they test whether more homogenous analogies for a project produce a more reliable cost estimate. In order to achieve this, they define a metric for homogeneity. As a result, they observe a large variation in reliability between high and low homogeneity level projects. The main disadvantage of this study is that they test their hypothesis on one dataset using one estimation algorithm. Instead, we implement a number of estimators and test them on more datasets in order to interpret the generality of our results independent of the cost model used.

There is a lack of cost models that are specific to application domains, and there are no published studies that address this gap specifically for the embedded domain. In the literature, research on embedded systems mainly concentrates on power cost analysis (Oliveira et al. 2004; Ragan et al. 2002; Tiwari et al. 1994; Zotos et al. 2005). To the best of our knowledge, there is no published study that focuses on the modeling/estimation of software development costs of embedded systems. In this article, our main aim is to fill in this gap from a data homogeneity point of view.

For estimating the cost of embedded software, in practice, parametric models like COCOMO, REVIC, and some commercial tools are used (Boehm 2009; Debardelaben et al. 1997; EstimatorPal 2009; Igoodsoft 2009; SCEP 2009). In particular, the embedded models of COCOMO and REVIC are the most applicable tools for the software costing of embedded systems. COCOMO uses nominal effort equations that are derived from labor effort, and related to the size of the software system. REVIC is the 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, and processors and memory capacity. However, these models are designed to be rather generic and should be calibrated before being used by other companies (Kitchenham et al. 2007).

3 Methodology

We bring a new perspective to data homogeneity and propose an application domain viewpoint, which we call *within-domain* versus *cross-domain*. The domain on which a software project is developed plays a very important role as it affects the whole software development lifecycle. In particular, the embedded domain is promising in today's world where embedded systems are so popular. Thus, we focus on the embedded software domain and investigate what type of data should be used for embedded software cost estimation.

In this context, we have the following research questions:

1. What type of training data should we use for embedded software cost estimation: cross-domain datasets or within-domain (embedded software) datasets?
2. Elsewhere (Kitchenham et al. 2007), it is discussed that small within-company datasets are more homogeneous and large cross-company datasets introduce heterogeneity to the solutions. "What is the effect of training dataset size on the prediction performance?"

In order to answer the research questions above, we compare the estimators that are trained on within-domain data with those trained on cross-domain data by using three different experimental setups:

S1 train and validate the estimators only on the embedded software 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 software dataset, and then validate on the embedded software dataset

S3 train the estimators on the cross-domain dataset by using *all* of the projects and validate on the embedded software dataset

Among these setups, S1 defines a baseline where the estimations are carried out with within-domain datasets. S2 is a cross-domain setup where the training dataset is different from the within-domain dataset, but has the same size as in S1. If we were to use different training set sizes in S1 and S2, then we would have difficulty in determining whether our results are due to using different applications domains or using different training set sizes. In order to see how the results are affected by the training set size, S3 is designed. S3 is the same as S2 except that this time all of the projects in the cross-domain dataset are used as the training dataset.

3.1 Data

In our research, datasets from three main sources are used. The first one is PROMISE Data Repository which is a public repository that contains data about software projects from NASA and different universities located in US (Boetticher et al. 2007). Three of the cost estimation datasets in PROMISE are used in this research: *cocomonasa_v1*, *coc81_1_1*, and *nasa93*. These datasets are collected in COCOMO81 format and include 17 attributes in total (15 effort multipliers, one size attribute, and one actual effort value) except that nasa93 includes seven additional project attributes (Boehm 1981).

The second source is Bogazici University Software Engineering Research Laboratory repository (SoftLab), which contains data about software projects that belongs to various companies in Turkey. Three of the cost estimation datasets in SoftLab are used in this research: *sdr05*, *sdr06*, and *sdr07*. All of these datasets are collected in COCOMO II format and include 22 attributes in total (15 effort multipliers, 5 scale factors, one size attribute, and one actual effort value) (Boehm 1999).

The last resource is International Software Benchmarking Standards Group (ISBSG) repository, which is a non-profit organization that maintains a software project management database from a variety of organizations (Lokan et al. 2001). In this research, we use ISBSG Release 10 that contains 4,106 projects each with around 106 attributes. These attributes can be classified into 15 categories that are rating, sizing, effort, productivity, schedule, quality, grouping attributes, architecture, documents and techniques, project attributes, product attributes, effort attributes, size attributes, size other than FSM, and software age. More details can be found in (Lokan et al. 2001).

In order to form the embedded software datasets to be used in our experiments, from *coc81_1_1* and *nasa93*, by choosing only the projects with embedded development mode, we form two new datasets that we call *coc81_e* and *nasa93_e*. These new datasets contain 28 and 21 projects with 17 COCOMO I attributes. Then, we form a third embedded software dataset from the ISBSG dataset by selecting the projects whose application type is embedded. There are in total 21 such projects. From these projects, the ones whose size and effort attributes are empty are removed and the number of projects decreases to 17. Lastly, unnecessary attributes and the attributes with an empty value for all projects are removed. The remaining attributes that 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. Among these eight attributes, those that are categorical are converted into numerical format by assigning a number (1, 2, 3 …) for each category. This final dataset is called *ISBSG_e*.

In order to form a cross-domain dataset to be used in S2 and S3, firstly, all the datasets except *nasa93*_e and *coc81_e* (*cocomonasa_v1*, *sdr05*, *sdr06*, *sdr07*) are merged into one large dataset, which is called *crossdomain1*. Secondly, one more cross-domain dataset is formed by adding the nonembedded projects in *coc81* and *nasa93* to *crossdomain1*. This second dataset is called as *crossdomain2*. While merging the datasets, only the common attributes, which are 15 in total, are selected from each dataset. The reason for the different number of features is that they are collected in either COCOMO or COCOMO II model format. These two datasets, *crossdomain1* and *crossdomain2*, will be used to train *coc81_e* and *nasa93_e* since they have same attributes. However, for *ISBSG_e*, we have different attributes; so, thirdly, a new cross-domain dataset is formed from the ISBSG dataset by selecting the projects whose application types are not embedded and do not include null values. This new dataset including 104 projects each with 8 attributes is called as *ISBSG*. By taking into account all of the data repositories used, we have three embedded software (within-domain) datasets and three cross-domain datasets for our experiments. An overview of the contents of these datasets is given in Table 1.

3.2 Cost models

There has been various research in the area of software cost and effort estimation, where several different approaches have been used: parametric models, expertise-based techniques, function point analysis, learning-oriented techniques, dynamics-based models, regression-based models, and composite Bayesian techniques for integrating expertise-based and regression-based models (Boehm 1981; Walston and Felix 1977; Albrecht 1979; Putnam 1978). In real life, project managers have to make cost and effort related decisions under uncertainty. Therefore, they need a model that has high accuracy, free of expert judgment, and has flexible attributes. Such a model can be constructed by using

Table 1 An overview of the datasets

Datasets			# of Projects	Total # of Projects
Domain	Name	Content		
Within-domain (embedded software)	coc81_e	Embedded software projects from coc81	28	28
	nasa93_e	Embedded software projects from nasa93	21	21
	ISBSG_e	Embedded software projects from ISBSG	17	17
Cross-domain	crossdomain1	cocomonasa_v1	60	149
		sdr06	24	
		sdr05	25	
		sdr07	40	
	crossdomain2	crossdomain1	149	256
		Remaining projects from coc81	35	
		Remaining projects from nasa93	72	
	ISBSG	Remaining projects from ISBSG	104	104

machine-learning algorithms. The capacity to learn from experience, analytical observation, and other means, results in a system that can continuously self-improve and, thereby, offer increased efficiency and effectiveness (Alpaydin 2004). In such circumstances, learning-based predictor models are expected to be more useful. In the literature, there are a number of studies that focus on applying machine-learning techniques to cost data (Baskeles et al. 2007; Srinivasan and Fisher 1995; Briand et al. 1992; Boetticher 2001). We have chosen six of these methods to be used in our research for estimating the effort value for embedded software projects, since we wanted our results to be independent from a specific model. We have chosen widely used models such as linear regression, support-vector regression (SVR), and multi-layer perceptron (MLP) as well as other models such as kernel smoother (KS), k-nearest neighbors, and voting cost estimation models in the context of embedded system domain.

Linear regression (LR) seeks a linear combination of attributes to estimate cost (Alpaydin 2004). The output is a linear function, which is the weighted sum of the input variables. Despite its simplicity, it has been widely used in other studies, and this is the reason why we used it in our research (Angelis and Stamelos 2000; Mason and Sweeney 1992; Perel 1994; Shepperd et al. 1996).

Another machine-learning algorithm we used is KS. It is similar to linear regression except that the importance of data instances is not the same for all inputs. It gives less weight to distant samples by dividing the data into bins and fitting a kernel function to the data that are in the same bin. The most popular kernel function is the *Gaussian* kernel, which is also the one used in this research (Alpaydin 2004).

We also apply SVR that approximates a solution into a higher dimensional space where the solution is linear (Smola and Schölkopf 2003). We have used Steve Gunn's SVR implementation with a spline kernel (Gunn 1998). Another nonlinear complex model we have included in our research is the MLP with back-propagation, where cost is estimated as nonlinear combinations of input attributes (Fausett 1994). We have used Phil Brierley's neural network with a back-propagation implementation (Brierley 2009).

We have also included an unsupervised method, k-nearest neighbor algorithm (KNN), which gives a baseline for comparing the similarity of the projects. While we use our prior knowledge about the application domain of projects for defining similarity, KNN measures it in terms of the Euclidean distance between input attributes of different projects.

Finally, we have used a voting algorithm in order to combine the results of different estimators. Voting is one of the methods for combining multiple learners (Alpaydin 1998). The estimated cost value of each learner is given a particular weight and this weighted sum of each estimate is taken as the final estimation. In this research, we have set equal weight values for each learner (1/number of learners).

3.3 Experimental design

There are three embedded software (within-domain) and three cross-domain datasets that can be used to compare our three setups (Table 1). Embedded software datasets are *coc81_e*, *nasa93_e*, and *ISBSG_e*, whereas cross-domain datasets are *crossdomain1*, *crossdomain2,* 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 by using *Min-max normalization* (Shalabi and Shaaban 2006).

The datasets used for S1 and the way they are processed are given in Fig. 1.

In S1, since the estimators are both trained and validated on embedded software datasets, 10 × 10 cross-validation is used to create various training and validation sets from the same dataset (Alpaydin 2004). Firstly, the normalized dataset is shuffled 10 times into random order and then divided into 10 bins. Training data are 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 (Alpaydin 2004). Thirdly, the estimators are applied to the training set to learn the models'

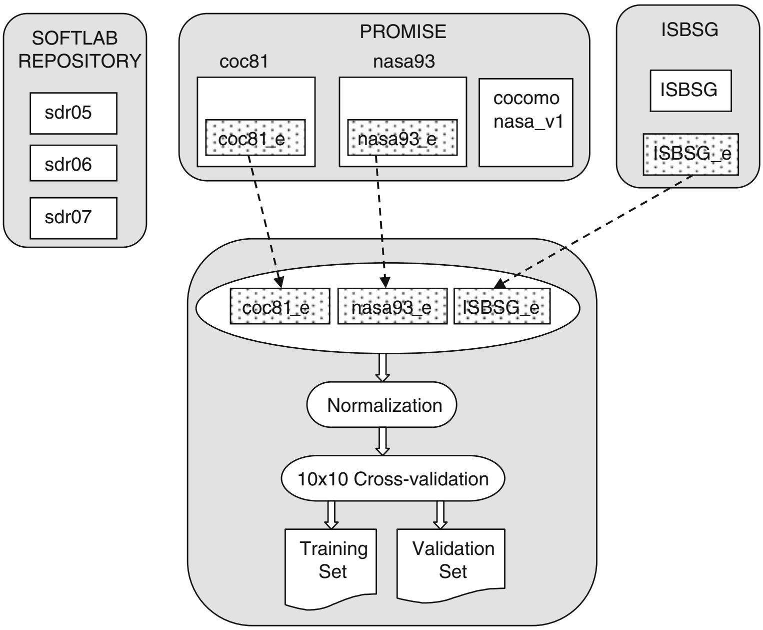

Fig. 1 Data processing in S1

```
M = 10                                      // num of iterations
N = 10                                      // num of bins
DATA = coc81_e,nasa93_e,ISBSG_e             // embedded software 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)
```

Fig. 2 Pseudo code of S1

parameters. Then, the models and the voting method are applied to the validation bin for estimation. 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 in Fig. 2.

The datasets used for S2 and the way they are processed are given in Fig. 3.

In S2, there is no need for cross-validation because there are separate training (cross-domain datasets) and validation sets (embedded software datasets). As we want to use a training set with the same size as the embedded software 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, unlike S1, all of the estimators are first trained on the randomly selected subset of projects from the cross-domain dataset, and then validated on embedded software dataset in order to determine the performance measures. In order to obtain 100 results as in S1, the whole setup is run for 100 times. The pseudo code of S2 is given in Fig. 4.

The datasets used for S3 and the way they are processed are given in Fig. 5.

S3 is almost the same as S2. 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. The reason for this is to check if the dataset size affects the performance of the models on the embedded software dataset. The pseudo code S3 is given in Fig. 6.

3.4 Performance measures

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: mean magnitude of relative error (MMRE), median magnitude of relative error (MdMRE), and prediction at level r (PRED(r)). These are the measures calculated from

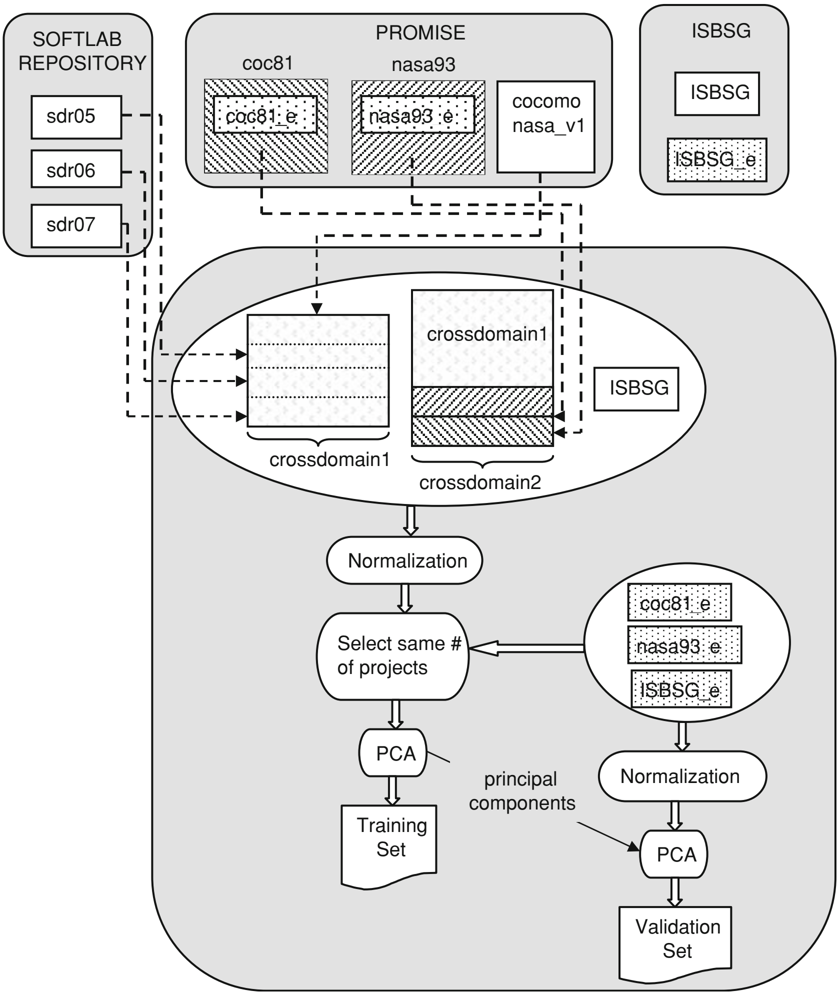

Fig. 3 Data processing in S2

magnitude of relative error (MRE) between the actual and estimated values (Menzies et al. 2006):

$$\text{MRE} = \frac{|\text{predicted} - \text{actual}|}{\text{actual}} \tag{1}$$

Simply, MMRE is the mean of the MRE values and MedianMRE (MdMRE) is the median of the MRE values. A third measure is used to examine the cumulative frequency of MRE for a specific error level, which is Prediction at level r or PRED(r). In this study, we take the desired error level as $r = 25$. That is, if we have n projects and there are m of them whose MRE is smaller than 25%, then PRED (25) is equal to m/n:

```
M = 100
TRA_DATA = crossdomain1, crossdomain2,ISBSG          // training data
VAL_DATA = coc81_e,nasa93_e,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)
```

Fig. 4 Pseudo code of S2

$$\mathrm{PRED}(N) = \frac{100}{T}\sum_{i}^{T}\begin{cases} 1 & \text{if } \mathrm{MRE}_i \leq \frac{N}{100} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

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.

Since we have three competing setups that are used to predict the same dataset, the *t*-test is used for comparing different sets of predictions (Stensrud and Myrtveit 1998). The *t*-test is a test of the null hypothesis 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% significance level ($\alpha = 0.05$) to reject the null hypothesis, then we can be confident that the two sets of predictions are not significantly different.

3.5 Threats to validity

As a threat to internal validity of our results, firstly, we constructed three embedded software datasets that are small in size. To overcome this issue, we used a 10 $\times$ 10 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 (Foss et al. 2003; Korte and Port 2008). Although they are the most widely used evaluation criteria for assessing the performance of different prediction models, there are some studies that question their accuracy. For example, according to Foss et al. (2003), both MMRE and MdMRE are inherently biased and do not always select the best model. Also, Korte and Port (2008) stated that most of the results of these measures are questionable due to large possible variations resulting from population sampling error. However, in order to obtain a

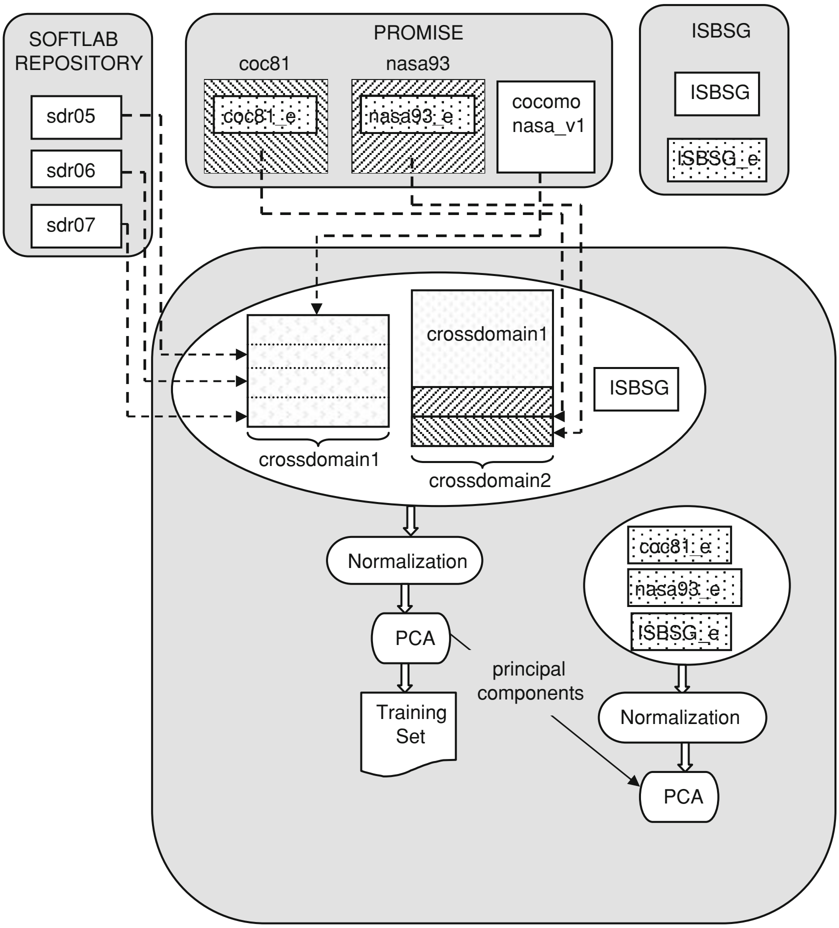

Fig. 5 Data processing in S3

more accurate comparison of the models developed, we give statistical test results (*t*-test) and show the box plots of residuals (i.e. the estimate-actual) for each setup as suggested in (Kitchenham et al. 2001).

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, the ISBSG dataset that contains data about current software development projects from different organizations in the world is used to generalize our results.

4 Results

4.1 Results for research question 1

In our experiments, there are two cross-domain datasets, *crossdomain1* and *crossdomain2*, that can be used for training *coc81_e* and *nasa93_e*. Thus, there are two possible cases for

```
M = 100
TRA_DATA = crossdomain1, crossdomain2,ISBSG            // training data
VAL_DATA = coc81_e,nasa93_e,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
```

Fig. 6 Pseudo code of S3

S2 and S3 that we call S2-crossdomain1, S2-crossdomain2, S3-crossdomain1, and S3-crossdomain2. However, for *ISBSG_e* dataset, there is only one cross-domain dataset, *ISBSG*, thus, there is only one possible case for S2 and S3. Results for each embedded software dataset are given in Tables 2, 3, 4 respectively. Since MdMRE, MMRE, and PRED measures can be misleading as stated in our threats to validity section, Figs. 7, 8, and 9 are included to visualize the residuals for each setup in Tables 2, 3, and 4, respectively. The box plots are interquartile plots, and the line within each box is the median. The length of the box indicates the spread of the distribution and the position of the median (the line) show the skewness of the distribution. In order to say that a method performs well, the box length and tails should be small (Kitchenham et al. 2001).

For *coc81_e*, the results for each setup are given in Table 2. Best values for each method are given in bold. When we look at the results, we see that MdMRE and MMRE values are very high and PRED values are very low. However, our aim is to compare the setups we designed, not the methods with each other. With this in mind, the best values for most of the methods are obtained when S1 is used, which means that the methods perform better when they are trained on the embedded software (within-domain) dataset, *coc81_e*.

For *coc81_e*, the box plots for each method are given in Fig. 7 in order to compare their performances in different setups. In contrast to the results given in Table 2, the best performance is obtained when they are trained on crossdomain2 by using all of the projects, because the box length and tails for S3-crossdomain2 (5) are clearly smaller than the box length and tails for other setups (Kitchenham et al. 2001).

For *nasa93_e*, the results for each setup are given in Table 3. Best values for each method are given in bold. When we look at the results, we see that, again, the best values for most of the methods are obtained when S1 is used, which means that the methods perform better when they are trained on the embedded software (within-domain) dataset, *nasa93_e*.

For *nasa93_e*, the box plots for each method are given in Fig. 8. In contrast to the results given in Table 3, it can be seen that for all methods, the best performance is obtained when they are trained on crossdomain2 by using all of the projects (5).

Table 2 Results for coc81_e dataset

Method	S1 (%)			S2-crossdomain1 (%)			S2-crossdomain2 (%)		
	MdMRE	MMRE	PRED	MdMRE	MMRE	PRED	MdMRE	MMRE	PRED
KS	1,528	1,528	10.89	97	1,278	10.14	**91**	754	11.35
KNN	213	**213**	**18.31**	216	944	12.02	122	658	15.20
LR	718	**718**	**10.89**	436	1,608	7.92	**377**	1,314	10.04
MLP	1,630	1,630	**10.39**	394	1,790	8.20	**246**	**1,224**	10.29
SVR	2,988	**2,988**	**7.42**	931	4,038	5.41	916	4,250	6.01
Voting	933	**933**	9.40	436	1,596	7.74	**326**	1,473	**10.32**

Method	S3-crossdomain1 (%)			S3-crossdomain2 (%)		
	MdMRE	MMRE	PRED	MdMRE	MMRE	PRED
KS	98	1,375	10.32	92	**743**	**11.42**
KNN	124	926	7.07	**87**	464	10.60
LR	670	1,370	3.53	476	1,288	7.07
MLP	411	2,265	7.03	268	1,327	9.97
SVR	**909**	3,511	3.53	942	3,780	3.53
Voting	479	1,530	1.77	433	1,402	6.61

The bold values show the best values for each method and for each measure (MMRE, MdMRE, and PRED). There are 3 bold values for each algorithm

Table 3 Results for nasa93_e dataset

Method	S1 (%)			S2-crossdomain1 (%)			S2-crossdomain2 (%)		
	MdMRE	MMRE	PRED	MdMRE	MMRE	PRED	MdMRE	MMRE	PRED
KS	138	**138**	**32.67**	92	317	12.73	**88**	189	12.02
KNN	193	193	**27.22**	75	456	19	72	212	16.36
LR	116	**116**	24.75	108	675	15.7	97	422	15.60
MLP	175	**175**	**28.21**	112	546	16.36	100	297	15.18
SVR	444	**444**	**16.37**	274	922	12.58	276	864	12.91
Voting	187	**187**	**26.23**	86	533	18.81	69	334	20.03

Method	S3-crossdomain1 (%)			S3-crossdomain2 (%)		
	MdMRE	MMRE	PRED	MdMRE	MMRE	PRED
KS	90	283	12.63	90	179	11.12
KNN	73	1,114	14.14	**59**	**175**	23.57
LR	78	871	**33**	**72**	478	28.28
MLP	125	736	14.09	**91**	368	16.50
SVR	**169**	619	9.42	174	919	14.05
Voting	72	683	22.96	**60**	372	21.97

The bold values show the best values for each method and for each measure (MMRE, MdMRE, and PRED). There are 3 bold values for each algorithm

Table 4 Results for ISBSG_e dataset

Method	S1 (%)			S2 (%)			S3 (%)		
	MdMRE	MMRE	PRED	MdMRE	MMRE	PRED	MdMRE	MMRE	PRED
KS	7,599	7,599	**14.85**	**120**	**1,590**	13.33	123	2,418	11.88
KNN	**187**	**187**	**35.64**	216	1,509	9.14	194	648	11.64
LR	1,878	1,878	6.93	244	2,585	**8.21**	**198**	**1,644**	5.82
MLP	**91**	**91**	**68.3**	299	2,754	8.85	524	3,026	8.50
SVR	9,819	**9,819**	**22.7**	**2,258**	11,483	11.53	2,919	14,248	17.47
Voting	3,239	**3,239**	**10.8**	**623**	3,800	6.69	762	4,230	5.94

The bold values show the best values for each method and for each measure (MMRE, MdMRE, and PRED). There are 3 bold values for each algorithm

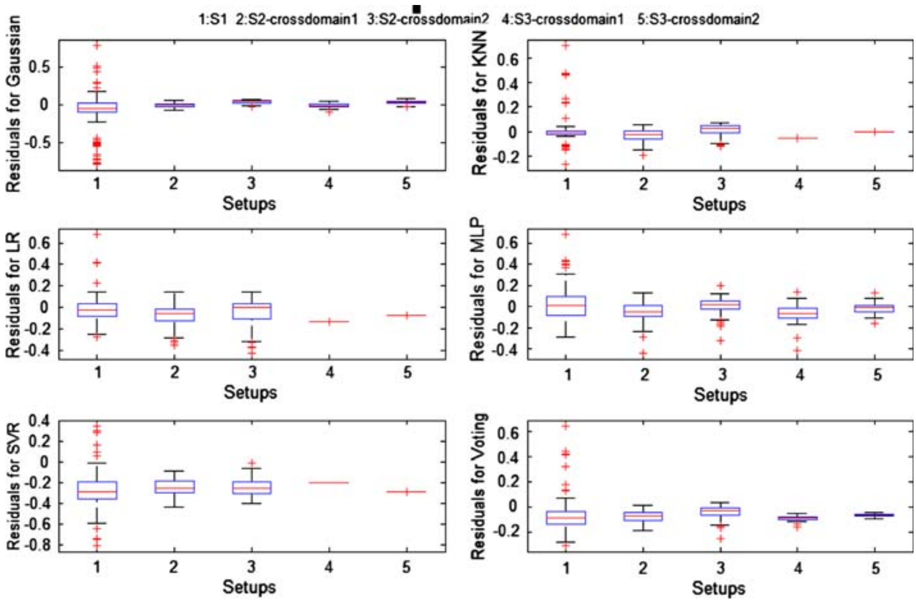

Fig. 7 Boxplots for coc81_e

For *ISBSG_e*, the results for each setup are given in Table 4. The best values for each method are given in bold. We want to remind readers that there are in total three possible cases for this dataset since there is only one cross-domain dataset that is *ISBSG*. When we look at the results, we see that, again, the best values for most of the methods are obtained when S1 is used, which means that the methods perform better when they are trained on the embedded software (within-domain) dataset, *ISBSG_e*.

For *ISBSG_e*, the box plots for each method are given in Fig. 9. In contrast to the results given in Table 4, it can be seen that for all methods except MLP, the best performance is obtained when they are trained by using all of the projects (3). MLP performs the best when it is trained on a subset of projects (2).

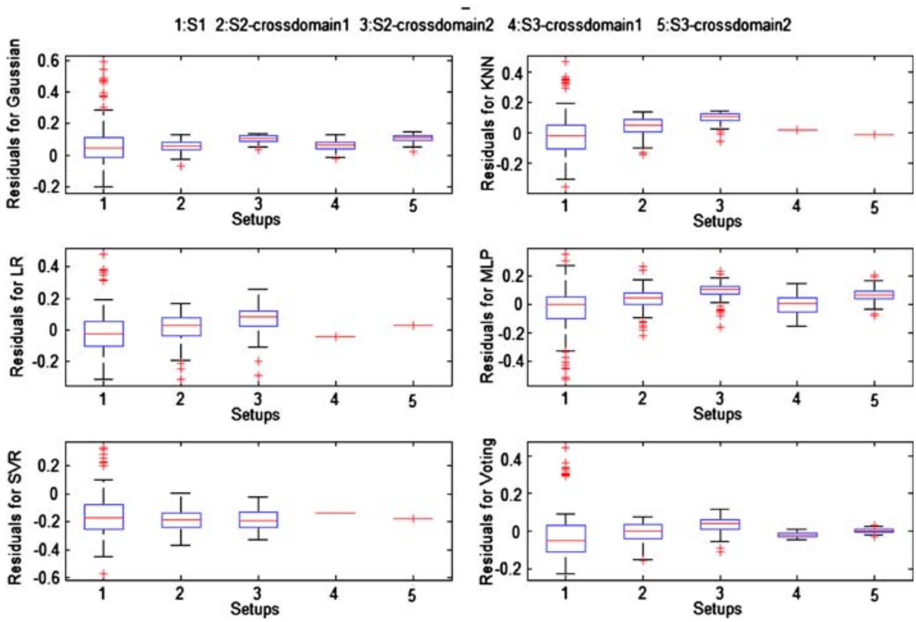

Fig. 8 Boxplots for nasa93_e

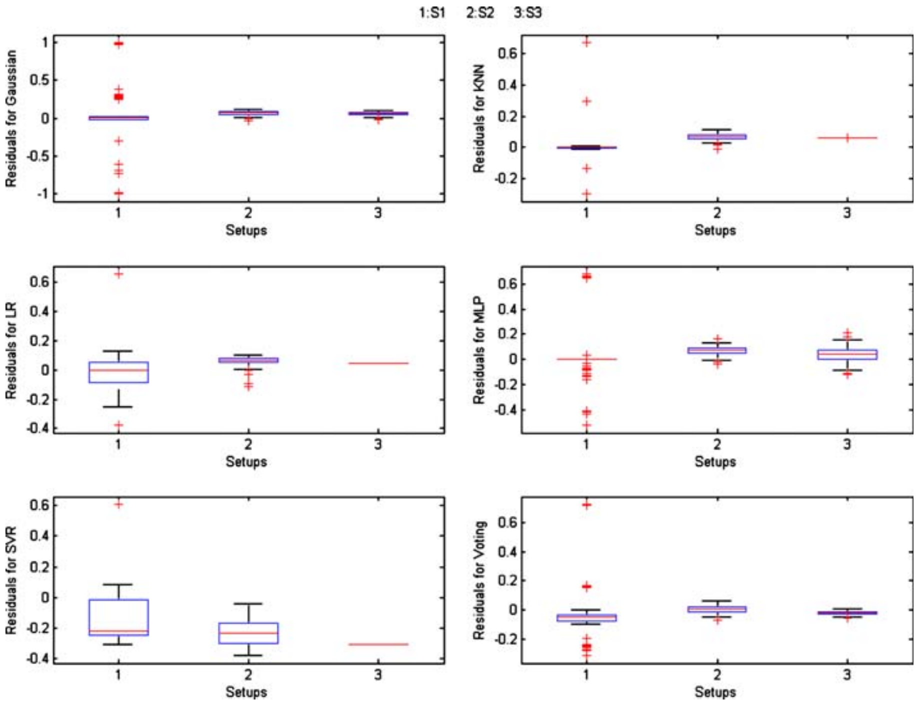

Fig. 9 Boxplots for ISBSG_e

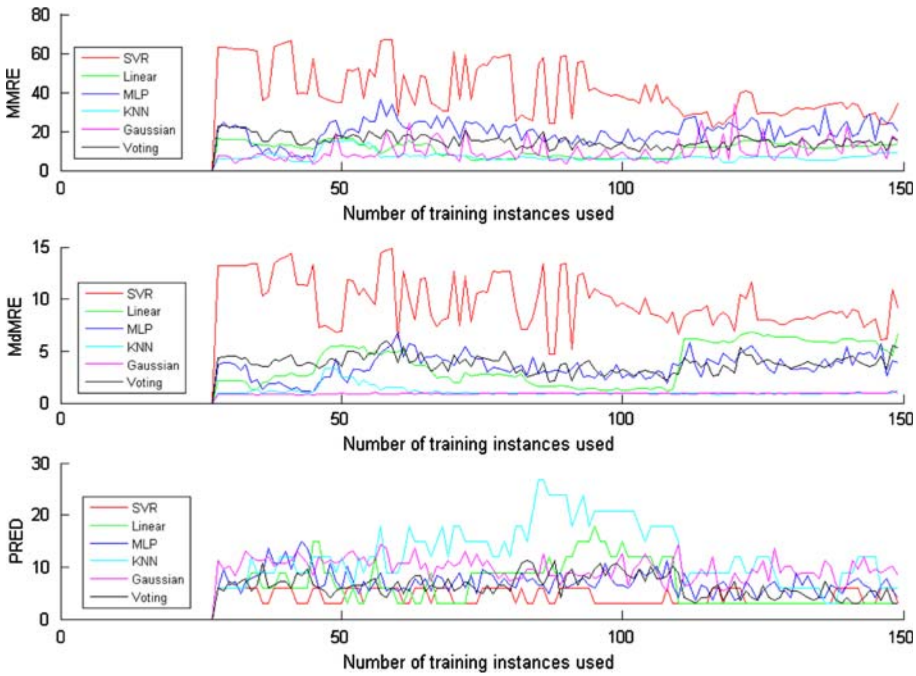

Fig. 10 Performance results for coc81_e

4.2 Results for research question 2

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 software datasets. In this experiment, we begin training the embedded software 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 rest of the article.

For *coc81_e*, we perform the performance experiment by training the methods using the *crossdomain1* dataset, and then validating using the *coc81_e* dataset. We begin with 28 projects in the training set and continue by increasing the number one by one until all of the 149 projects in *crossdomain1* are used. The results obtained are given in Fig. 10. When we look at the figure, we can see that there is no general tendency such as effort estimation performance gets better or worse as the training set size increases.

For *nasa93_e*, we perform the performance experiment by training the methods using the *crossdomain1* dataset and then validating using the *nasa93_e* dataset. We begin with 21 projects in the training set and continue by increasing the number one by one until all of the 149 projects in *crossdomain1* are used. The results obtained are given in Fig. 11. When we look at the figure, again, we cannot see a direct relationship between training set size and performance.

For *ISBSG_e*, we perform the performance experiment by training the methods using the *ISBSG* dataset, and then validating using the *ISBSG_e* dataset. We begin with 17

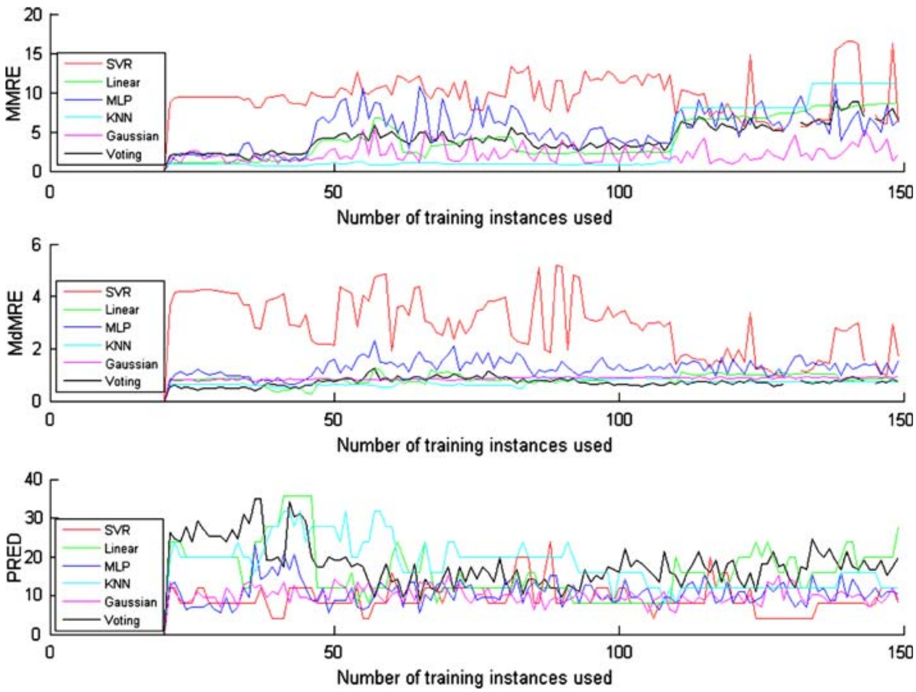

Fig. 11 Performance results for nasa93_e

projects in the training set, and continue by increasing the number one by one until all of the 104 projects in *crossdomain1* are used. The results obtained are given in Fig. 12. When we look at the figure, again, there is no general tendency about the methods' performances.

5 Evaluation

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 what type of training data (within-domain or cross-domain) should be used 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 methods are trained on the embedded software datasets (S1). However, when we look at the box plots of residuals for each method, we see that the best performances are obtained when the methods are trained on *crossdomain2* or *ISBSG* by using all of the projects (S3-crossdomain2 or S3 for *ISBSG*). Thus, in order to come up with a neutral conclusion, we performed *t*-tests between different sets of results of each algorithm. While performing the *t*-tests, we used a 0.05 significance level ($\propto = 0.05$).

According to the *t*-tests performed, the best performing setups obtained for each estimator are given for *coc81_e* dataset in Table 5. When we look at the results, S3-crossdomain2 performs best for most of the methods. This means that when the estimators are trained by using all of the projects in *crossdomain2*, the best performances are obtained for the *coc81_e* dataset.

The best performing setups for *nasa93_e* dataset are given in Table 6. When we look at the results, S3-crossdomain2 again performs best for most of the methods. This means that

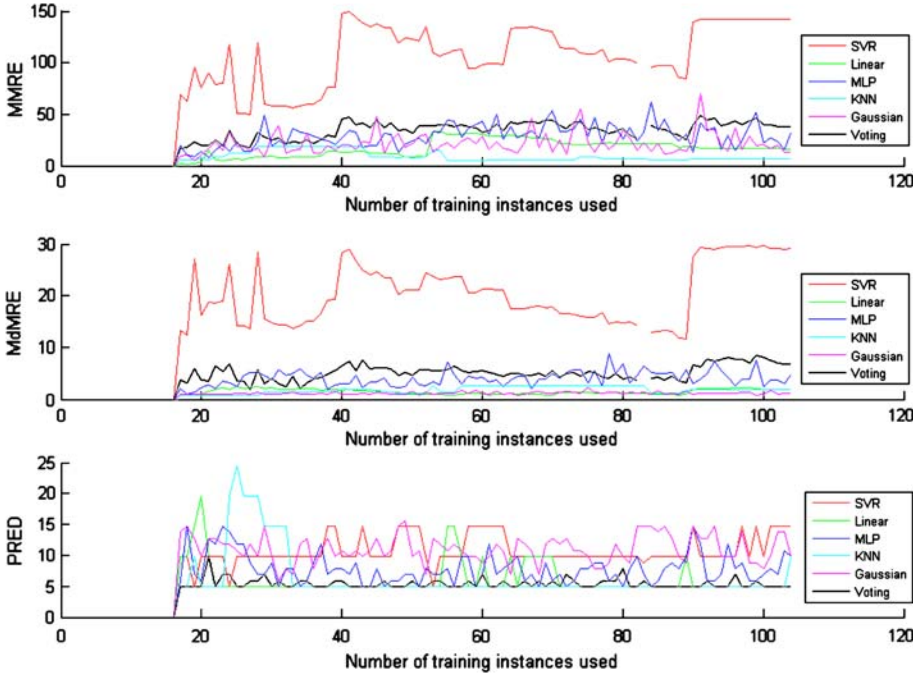

Fig. 12 Performance results for ISBSG_e

Table 5 *t*-test results for coc81_e dataset

Method	MdMRE	MMRE	PRED (25)
SVR	S3-crossdomain1	S3-crossdomain1	S2-crossdomain2
GS	S3-crossdomain2	S3-crossdomain2	S3-crossdomain2
LR	S2-crossdomain2	S1	S2-crossdomain2
MLP	S3-crossdomain2	S3-crossdomain2	S3-crossdomain2
KNN	S3-crossdomain2	S1	S2-crossdomain2
Voting	S2-crossdomain2	S1	S2-crossdomain2

Table 6 *t*-test results for nasa93_e dataset

Estimator	MdMRE	MMRE	PRED (25)
SVR	S3-crossdomain1	S1	S3-crossdomain2
GS	S3-crossdomain1	S3-crossdomain2	S1
LR	S3-crossdomain2	S1	S3-crossdomain2
MLP	S3-crossdomain2	S1	S1
KNN	S3-crossdomain2	S3-crossdomain2	S3-crossdomain2
Voting	S3-crossdomain2	S1	S3-crossdomain2

Table 7 *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 the estimators are trained by using all of the projects in *crossdomain2*, the best performances are obtained for the *nasa93_e* dataset.

The best performing setups for the *ISBSG_e* dataset are given in Table 7. We should note that the *ISBSG_e* dataset can use only the *ISBSG* dataset as training data, because, the attributes in *crossdomain1* and *crossdomain2* are different; so, there are three possible cases: S1 S2, and S3. 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*, the best performances are obtained for the *ISBSG_e* dataset.

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

On the other hand, for two of the embedded software datasets used (*coc81_e* and *nasa93_e*), 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. According to 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 no direct relationship between training set size and the performances of the estimators. We can only suggest that all possible training set sizes should be tested, and then the best one should be selected for use in experiments.

Although finding the best performing algorithm is not the main focus of this research, we would like to give some insights on the comparative performances of the algorithms we used. When we look at the six algorithms used, k-nearest neighbor (KNN) outperforms the others, because KNN learns the effort values only from similar projects. KS and MLP are the next best ones, where Kernel smother assigns larger weight values to the similar projects and MLP dynamically adjusts its weights according to the delta between the attributes of input project and the training project. Linear regression comes next in terms of prediction performance, since it tries to fit a linear model to the multi-variate project samples. Voting algorithm, on the other hand, does not perform well compared to the previous models. The reason may be that we use equal weights for each model and, thus, all models including the bad-performing ones have the same effect on the result. Finally, the support-vector regression (SVR) model is the worst performing one among the six models used. SVR tries to approximate the solution into a higher dimensional space, where the solution is linear. Cost data is generally high-dimensional; the solution becomes highly complex resulting in over fitting.

We should carefully note that the ordering of these methods may not necessarily reflect their accuracies in practice. Our sole intention is to span as large a number of methods as possible while investigating within versus cross- application domain issues. Determining the best method is out of the scope of this article and is still an open issue.

5.1 Answers to research questions and practical implications

In this article, we report our study about training data domain on software cost modeling, and we especially focus on the cross- versus within-application domain. We have analyzed embedded software systems due to the increasing attention in this domain. We have designed experiments to answer our research questions. Below, we provide the analysis on the rationale of our experimental results for each research question:

1. What type of training data should we use for embedded software cost estimation: cross-domain datasets or within-domain (embedded software) datasets?—In our experiments, we observed that all estimators perform better when they are trained on the cross-domain datasets than they do on the embedded software datasets. Thus, we can conclude that cross-domain datasets should be used for training estimators in embedded software cost estimation.
2. What is the effect of training dataset size on the prediction performance?—In our experiments, we observed that as training set size increases, performance gets better for two of four cross-domain datasets. However, we could not observe this relationship consistently through all datasets. Thus, we suggest determining the correct training dataset size after validating on possible set sizes. A possible reason why this remains an open issue is the fact that the choice of the learning algorithm and data quality have undeniable effects on the size of the training set.

Our results also have practical implications for decision making in the software engineering industry. Below, we list the implications of our research for software practitioners:

1. Our comprehensive analysis of data usage for cost estimation in embedded software development domain can help managers to decide which data to use: cross-domain or within-domain. This is very critical for 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, at least for embedded systems.
2. Project managers can benefit from the learning-based methods we have used, in order to make more accurate estimates while bidding for (1) a new project, (2) allocating the resources among different projects, as well as among different stages of software development lifecycle.
3. A widely used approach is to employ analogy based cost models. However, this assumes the availability of project data that are *similar* to the project at hand, which can be difficult to obtain, i.e. there may be no similar projects in house and obtaining data from other companies may be limited due to confidentiality. On the other hand, our proposed framework suggests that it is not necessary to take care of particular development characteristics of a project, i.e. its similarity with other projects, while constructing cost models. On the contrary, we observe that rather than using projects from a similar application domain, it is better to use projects from a wider spectrum.[1]

[1] This observation is consistent with other cost models such as COCOMO, where model parameters are determined from a diverse set of software projects. Though COCOMO is a generic model with predefined parameters, these can be fine tuned with local data.

6 Conclusion and future work

There have been many studies that compare within-company cost estimation models to cross-company models and try to find an answer to the question of when companies should rely on cross-company models. However, in this study, we focus on a different aspect of data homogeneity, that is the application domain, and investigate what type of training data should be used for embedded software cost estimation. Further, we investigate the effects of training data size on prediction performance.

We carry out our experiments on public datasets in order to enable other researchers to replicate our experiments. We have used three different experimental setups with a number of learning-based methods. The first setup is to apply the estimators in a within-domain setup. The second one uses a cross-domain training dataset with the same as in the first setup. The last one is the same as the second one except that the training dataset size grows larger. According to the experiments, we can conclude that cross-domain datasets should be used for training estimators in embedded software cost estimation.

In order to find the effect of training data size, we performed additional experiments and we observed that there is no direct relationship between training set size and performance. The optimum training data size depends on the method used, thus, we can only suggest that all possible training set sizes should be tested and then the best one should be selected for use in experiments.

Our main research contribution is to investigate the homogeneity of cost data in terms of application domain. This issue has not been studied before in software cost estimation literature. The second contribution of our study is that, we investigate the effect of training data size on prediction performance, an open question discussed in other studies. We performed experiments to answer this question, yet we conclude that this still remains as an open issue due to variations in data quality and the choice of prediction algorithm. However, the current experiment results can guide project managers in making a decision on how much data is enough for training the algorithm. Finally, we benefit from various machine-learning techniques for software cost estimation and provide a performance comparison. Also, our experimental design may inspire and guide other researchers who would be conducting research on this domain. In our experiments, we use different datasets from public repositories so that other researchers can replicate, refute and/or improve our results.

As a future research direction, 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.

Acknowledgments This research is supported in part by Boğaziçi University research fund under grant number BAP 06HA104 and by Tubitak EEEAG 108E014.

References

Albrecht, A. J. (1979). Measuring application development productivity. In *Proceedings of the joint SHARE, GUIDE, and IBM application development symposium*, Monterey, CL, October 14–17 (pp. 83–92). IBM Corporation.

Alpaydin, E. (1998). Techniques for combining multiple learners. *Proceedings of Engineering of Intelligent Systems, 2*, 6–12.

Alpaydin, E. (2004). *Introduction to machine learning*. Cambridge: MIT.

Angelis, L., & Stamelos, I. (2000). A simulation tool for efficient analogy based cost estimation. *Journal of Empirical Software Engineering, 5*(1), 35–68. doi:10.1023/A:1009897800559.

Baskeles, B., Turhan, B., & Bener, A. (2007). Software effort estimation using machine learning methods. *ISCIS, 2007*, 1–6.

Boehm, B. W. (1981). *Software engineering economics. Advances in computer science and technology series*. Borough: Prentice Hall PTR.

Boehm, B. W. (1999). *COCOMO II and COQUALMO data collection questionnaire*. University of Southern California, Version 2.2.

Boehm, B. W. (2009). *COCOMO II model definition manual*. University of Southern California, Version 1.4. http://sunset.usc.edu/research/.

Boetticher, G. D. (2001). Using machine learning to predict project effort: Empirical case studies in data-starved domains. 1st International workshop on model-based requirements engineering, pp. 17–24.

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

Briand, L. C., Basili, V. R., & Thomas, W. M. (1992). A pattern recognition approach for software engineering data analysis. *IEEE Transactions on Software Engineering, 18*(11), 931–942. doi: 10.1109/32.177363.

Brierley, P. (2009). http://www.philbrierley.com/main.html?code/matlab.html&code/codeleft.html.

Debardelaben, J. A., Madisetti, V. K., & Gadient, A. J. (1997). Incorporating cost modeling in embedded-system design. *IEEE Design & Test of Computers, 14*(3), 24–35. doi:10.1109/54.605989.

EstimatorPal. (2009). http://software.techrepublic.com.com/download.aspx?docid=236622.

Fausett, L. (1994). *Fundamentals of neural networks*. Borough: Prentice Hall.

Foss, T., Stensrud, E., Kitchenham, B., & Myrtveit, I. (2003). A simulation study of the model evaluation criteria MMRE. *IEEE Transactions on Software Engineering, 29*(11), 985–995. doi: 10.1109/TSE.2003.1245300.

Gunn, S. R. (1998). Support vector machines for classification and regression. Faculty of Engineering, Science and Mathematics, School of Electronics and Computer Science, Tech. Rep., May 1998 (online). Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.9736.

Igoodsoft. (2009). http://www.igoodsoft.com/sesdevelopment.asp.

Kitchenham, B. A., Mendes, E., & Travassos, G. H. (2007). Cross- vs. within-company cost estimation studies: A systematic review. *IEEE Transactions on Software Engineering, 33*(5), 316–329. doi: 10.1109/TSE.2007.1001.

Kitchenham, B. A., Pickard, L. M., MacDonell, S. G., & Shepperd, M. J. (2001). What accuracy statistics really measure. *IEEE Proceedings-Software, 148*(3), 81–85. doi:10.1049/ip-sen:20010506.

Korte, M., & Port, D. (2008). Confidence in software cost estimation results. *PROMISE, 2008*, 63–70. doi: 10.1145/1370788.1370804.

Leung, H., & Fan, Z. (2001). Software cost estimation. *Handbook of software engineering and knowledge engineering*. ftp://cs.pitt.edu/chang/handbook/42b.pdf.

Lokan, C., Wright, T., Hill, P. R., & Stringer, M. (2001). Organizational benchmarking using the ISBSG data repository. *IEEE Software, 18*(5), 26–32. doi:10.1109/52.951491.

Mason, A. K. & Sweeney, N. (1992). Parametric cost estimating with limited sample sizes. In *Proceedings of the 3rd annual artificial intelligence symposium*.

Menzies, T. (2007). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering, 33*(1), 2–13. doi:10.1109/TSE.2007.256941.

Menzies, T., Chen, Z., Hihn, J., & Lum, K. (2006). Selecting best practices for effort estimation. *IEEE Transactions on Software Engineering, 32*(11), 883–895. doi:10.1109/TSE.2006.114.

Ohsugi, N., Monden, A., Kikuchi, N., Barker, M. D., Tsunoda, M., Kakimoto, T., & Matsumoto, K. (2007). Is this cost estimate reliable?—The relationship between homogeneity of analogues and estimation reliability. In *1st International symposium on empirical software engineering and measurement*, ESEM 2007.

Oliveira, M. N., Martins, P. R. M., Barreto, R. S., & Carvalho, F. F. (2004). Towards a software power cost analysis framework using colored petri net. PATMOS 2004: International workshop on power and timing modeling, optimization and simulation, Santorini, Greece, Vol. 3254, pp. 362–371.

Perel, R. J. (1994). Mold cost estimator generator utilizing standard data and linear regression. In *Proceedings of the regional technical conference of the society of plastic engineers*, pp. GI–G19.

Premraj, R., & Zimmermann, T. (2007). Building software cost models using homogenous data. In *ESEM '07: Proceedings of the 1st empirical software engineering and measurement*, Madrid, Spain, September 2007, IEEE, pp. 393–400.

Putnam, L. H. (1978). A general empirical solution to the macro software sizing and estimating problem. *IEEE Transactions on Software Engineering, 4*(4), 345–361. doi:10.1109/TSE.1978.231521.

Ragan, D., Sandborn, P., & Stoaks, P. (2002). A detailed cost model for concurrent use with hardware/software co-design. DAC 2002, ACM, pp. 269–274.

SCEP. (2009). Software cost estimation program. http://www.retisoft.com/Products.html.

Shalabi, L. A., & Shaaban, Z. (2006). Normalization as a preprocessing engine for data mining and the approach of preference matrix. In *IEEE proceedings of the international conference on dependability of computer systems* (DEPCOS-RELCOMEX'06).

Shepperd, M., Schofield, C., & Kitchenham, B. (1996). Effort estimation using analogy. *18th International conference on software engineering (ICSE'96)*, p. 170.

Smola, A. J., & Schölkopf, B. (2003). A tutorial on support vector regression. *NeuroCOLT Technical Report*. http://eprints.pascal-network.org/archive/00002057/01/SmoSch03b.pdf.

SoftLab. (2009). Software Research Laboratory. Department of Computer Engineering, Bogazici University. http://softlab.boun.edu.tr.

Srinivasan, K., & Fisher, D. (1995). Machine learning approaches to estimating software development effort. *IEEE Transactions on Software Engineering, 21*(2), 126–137. doi:10.1109/32.345828.

Stensrud, E., & Myrtveit, I. (1998). Human performance estimating with analogy and regression models: An empirical validation. In *Proceedings of 5th international metrics symposium*. Bethesda, MD: IEEE Computer Society.

Tiwari, V., Malik, S., & Wolfe, A. (1994). Power analysis of embedded software: A first step towards software power minimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2*(4), 437–445. doi: 10.1109/92.335012.

Vahid, F., & Givargis, T. D. (2002). *Embedded system design: A unified hardware/software introduction*. New York: Wiley.

Walston, C. E., & Felix, C. P. (1977). A method of programming measurement and estimation. *IBM Systems Journal, 16*(1), 54–73.

Zotos, K., Litke, A., Chatzigeorgiou, A., Nikolaidis, S., Stephanides, G., & Giannakides (Greece), G. (2005). Energy complexity of software in embedded systems. *From Proceeding* (483) ACIT—Automation, Control, and Applications.

Author Biographies

Ayşe Bakır She received her MSc degree in Computer Engineering from Bogazici University and her BSc degree in Computer Engineering from Gebze Institute of Technology in 2006. Her research interests include software quality modeling and software cost estimation.

Burak Turhan He is a research assistant and pursuing a PhD degree in the Department of Computer Engineering at Boğaziçi University. He received his BS and MS degrees from the same department. His research interests include all aspects of software quality and are focused on software defect prediction models. He is a student member of IEEE, IEEE Computer Society, and ACM SIGSOFT.

Ayşe B. Bener She is an assistant professor and a full time faculty member in the Department of Computer Engineering at Boğaziçi University. Her research interests are software defect prediction, process improvement and software economics. Bener has a PhD in information systems from the London School of Economics. She is a member of the IEEE, the IEEE Computer Society, and the ACM.