ON THE PREDICTABILITY OF SOFTWARE EFFORTS USING MACHINE LEARNING TECHNIQUES

Wen Zhang, Ye Yang and Qing Wang
Laboratory for Internet Software Technologies, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China

Keywords: Software effort prediction, K-medoids, BPNN, Data imputation.

Abstract: This paper investigates the predictability of software effort using machine learning techniques. We employed unsupervised learning as *k*-medoids clustering with different similarity measures to extract natural clusters of projects from software effort data set, and supervised learning as J48 decision tree, back propagation neural network (BPNN) and naïve Bayes to classify the software projects. We also investigate the impact of imputing missing values of projects on the performances of both unsupervised and supervised learning techniques. Experiments on ISBSG and CSBSG data sets demonstrate that unsupervised learning as *k*-medoids clustering has produced a poor performance in software effort prediction and Kulzinsky coefficient has the best performance in software effort prediction in measuring the similarities of projects. Supervised learning techniques have produced superior performances in software effort prediction. Among the three supervised learning techniques, BPNN produces the best performance. Missing data imputation has improved the performances of both unsupervised and supervised learning techniques.

1 INTRODUCTION

Software effort refers to estimate the human effort needed to develop a software artifact (Finnie et al., 1997). Overestimate of software effort may lead to tight schedule of development and faults may leave in the system after delivery whereas underestimate of effort may lead to delay of deliver of system and complains from customers. The importance of software development effort estimation has motivated the construction of models to predict it as accurate as possible.

Current software effort prediction techniques can be categorized into four types-empirical, regression, theory-based, and machine learning technqiues (Pendharkar et al., 2005). Machine Learning (ML) techniques learn patterns (knowledge) from historical project data and use these patterns for effort prediction, such as Artificial Neural Network (ANN), decision tree, and naïve Bayes. Recent studies (Pendharkar et al., 2005) (Jorgensen, 2004) provide detailed reviews of different studies on predicting software development effort.

The primary concern of this paper is on using machine learning techniques to predict software effort. Despite that COCOMO has provided a viable solution to effort estimation by building analytic model, machine learning techniques such as naïve Bayes and artificial neural network have come up with alternative approaches by making use of knowledge learned from historical projects. Although machine learning techniques though may not be the best solution for effort estimation, we believe they can be used at least by project managers to complement other models. Especially in intensely competitive software market, accurate estimation of software development effort has a decisive effect on success of a software project. Consequently, effort estimation using different techniques, and further risk assessment of budget overrun are of necessity for a trustworthy software project (Yang et al., 2009).

The basic idea of using machine learning techniques for effort prediction is that, historical data set contains many historical projects which are described by features with their values to characterize those projects and, similar values of the features of projects may induce almost the similar project efforts. The task of machine learning methods is to learn the inherent patterns of feature values and their relations with project efforts, which can be used for predicting the effort of new projects.

The rest of this paper was organized as follows. Section 2 introduced evaluation measures of software effort prediction. Section 3 described the data

Zhang W., Yang Y. and Wang Q..
ON THE PREDICTABILITY OF SOFTWARE EFFORTS USING MACHINE LEARNING TECHNIQUES.
DOI: 10.5220/0003408200050014
In *Proceedings of the 6th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE-2011), pages 5-14
ISBN: 978-989-8425-57-7
Copyright © 2011 SCITEPRESS (Science and Technology Publications, Lda.)

sets used in this paper and the preprocessing for experiments. Section 4 conducted clustering software projects in the data sets for effort prediction. Section 5 classified software projects on their efforts using machine learning classifiers. Section 6 presented the related work of this paper and Section 7 concluded this paper.

2 EVALUATION MEASURES

In software engineering, the deviation of predicted effort to real effort is used to measure the accuracy of effort estimators, such as MMRE (Magnitude of Relative Error), PRED(x) (Prediction within x) and AR (Absolute Residual) (Korte and Port, 2008). In machine learning, the performance of classification is often evaluated by accuracy and, F-measure (Steinbach et al., 2000) is usually used to evaluate the performance of unsupervised learning. Essentially, the evaluation measures of effort predictors in software engineering and those in machine learning do not conflict. In this paper, we adopted the measures from machine learning to evaluate the performances of effort predictors.

2.1 Accuracy

Assuming that $D = (D_1, ..., D_i, ..., D_m)$ is a collection of software projects, where D_i is a historical project and it is denoted by n attributes $X_i (1 \leqslant i \leqslant n)$. That is, $D_i = (x_{i1}, ..., x_{ij}, ..., x_{in})^T$. h_i denotes the label of effort for project D_i.

x_{ij} is the value of attribute $X_j (1 \leqslant j \leqslant n)$ on D_j. To evaluate the performance of a classifier in effort prediction, the whole data set was divided into two subsets: one is used for training the classifier and the other one is used for testing. That is, $D = (D_{train} \mid D_{test}) = (D_1, ..., D_k \mid D_{k+1}, ..., D_m)^T$, where k is the predefined number of projects in training set and m is the total number of projects in D. For instance, in 10-fold-cross validation, k should be predefined as $0.9m$ and the remaining $0.1m$ projects are used for testing the trained model. h_i is known for training set but remains unknown for testing set. By machine learning on the training set, a classifier denoted as M is produced. If we define a Boolean function F as Equation 1, then the performance of M is evaluated by accuracy as Equation 2.

$$F(M(D_j), h_j) = \begin{cases} 1, & \text{if } M(D_j) = h_j; \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

$$accuracy = \frac{1}{m-k} \sum_{k<j\leq m} F(M(D_j), h_j) \quad (2)$$

2.2 F-measure

In classification, Y was partitioned into l clusters and l is a predefined number of clusters in the data set. That is, $Y = c_1, ..., c_l$, $c_i = \{D_{i,1}, ..., D_{i,|c_i|}\}$ $(1 \leq i \leq l)$ and $c_i \cap c_j = \phi$. F-measure [14] is employed to evaluate the performance of unsupervised learning (clustering). The formula of F-measure is depicted as Equations 6 with the supports of Equations 3, 4, and 5.

$$P(i,j) = \frac{n_{i,j}}{n_j} \quad (3)$$

$$R(i,j) = \frac{n_{i,j}}{n_i} \quad (4)$$

$$F(i,j) = \frac{2 \times P(i,j) \times R(i,j)}{P(i,j) + R(i,j)} \quad (5)$$

$$F - measure = \sum_i \frac{n_i}{n} max_j F(i,j) \quad (6)$$

Here, n_i is the number of software projects with effort label h_i, n_j is the cardinality of cluster c_j, and $n_{i,j}$ is the number of software projects with effort label h_i in cluster c_j. n is the total number of software projects in Y. $P(i,j)$is the proportion of projects in cluster c_j with effort label h_i; $R_{i,j}$ is the proportion of projects with effort label h_i in cluster c_j; $F(i,j)$ is the F-measure of cluster c_j with respect to projects with effort label h_i. In general, the larger the F-measure is, the better is the clustering result is.

3 THE DATA SETS

We employed two data sets to investigate the predictability of software effort using machine learning techniques. The one is ISBSG (International Software Benchmarking Standard Group) data set (http://www.isbsg.org) and the other one is CSBSG (Chinese Software Benchmarking Standard Group) data set (He et al., 2008).

3.1 ISBSG Data Set

ISBSG data set contains 1238 projects from insurance, government, etc., of 20 different countries and each project was described with 70 attributes. To make the data set suitable for the experiments, we conduct three kinds of preprocessing: data pruning, data discretization and adding dummy variables.

Table 1: The used attributes from ISBSG data set.

Branch	Description	Type
Sizing technique	Count Approach	Nominal
	Adjusted Functional Points	Continuous
Schedule	Project Activity Scope	Nominal
Quality	Minor Defects	Continuous
	Major Defects	Continuous
	Extreme Defects	Continuous
Grouping Attributes	Development Type	Nominal
	Organization Type	Nominal
	Business Area Type	Nominal
	Application Type	Nominal
Architecture	Architecture	Nominal
Documents Techniques	Development Techniques	Nominal
Project Attributes	Development Platform	Nominal
	Language Type	Nominal
	Primary Programming language	Nominal
	1st Hardware	Nominal
	1st Operating System	Nominal
	1st Data Base System	Nominal
	CASE Tool Used	Nominal
	Used Methodology	Nominal
Product Attributes	Intended Market	Nominal

We pruned ISBSG data set into 249 projects with 22 attributes using the criterion that each project must have at least 2/3 attributes whose values are observed and, for each attribute, its values must be observed on at least 2/3 of total projects. We adopt the criterion for data selection in that too many missing values will deteriorate the performances of most machine techniques thus a convincing evaluation of software effort prediction is impossible. Among the 22 attributes, 18 of them are nominal attributes and 4 of them are continuous attributes. Table 1 lists the attributes used in the ISBSG data set.

Data discretization is utilized to transfer the continuous attributes into discrete variables. The values of each continuous attribute are preprocessed into 3 unique partitions. Too many partitions of values of an attribute will cause data redundancy nevertheless too few partitions may not capture the distinction of values of continuous attributes.

For each nominal attribute, dummy variables are added according to its unique values to make all variables having binary values. As a result, all the projects are described using 99 boolean variables with 0-1 and missing values. Only some of machine learning techniques can handle mixed data of nominal and continuous values but, most machine learning techniques can be used to handle Boolean values. In preprocessing, missing values are denoted as “-1” and kept for all projects on corresponding variables. Table 2 shows the value distribution of variables of ISBSG projects after preprocessing. Most values of the variables are zeros due to the transferring from discrete attributes to binary variables.

Table 2: The value distribution of variables of projects in ISBSG data set.

Value	Proportion
1	20% $\sim$ 50%
0	20% $\sim$ 60%
-1	5% $\sim$ 33%

Table 3: The effort classes categorized in ISBSG data set.

Class No	Number of projects	Label
1	64	Low
2	85	Medium
3	100	High

Finally, software effort of those selected 249 projects in the ISBSG data set was categorized into 3 classes. The projects with “normalized work effort” more than 6,000 person hours were categorized into the class with effort label as ”high”, projects with ”normalized work effort” between 2,000 and 6,000 person hours as ”medium” and projects with ”normalized work effort” less than 2,000 person hours as ”low”. Table 3 lists the effort distribution of the selected projects in the ISBSG data set.

Table 4: The used attributes from CSBSG data set.

Branch	Description	Type
Basic information of projects	Count Approach	Nominal
	City of development	Nominal
	Business area	Nominal
	Development type	Nominal
	Application type	Nominal
	Development Platform	Nominal
	IDE	Nominal
	Programming Language	Nominal
	Operation System	Nominal
	Database	Nominal
	Target Market	Nominal
	Architecture	Nominal
	Maximum Number of Concurrent Users	Nominal
	Life-cycle model	Nominal
	CASE Tool	Nominal
Size	Added lines of code	Continuous
	Revised lines of code	Continuous
	Reused lines of code	Continuous
	Number of team members in inception phase	Continuous
	Number of team members in requirement phase	Continuous
	Number of team members in design phase	Continuous
	Number of team members in coding phase	Continuous
	Number of team members in testing phase	Continuous
Schedule	Time limit in planning	Continuous
Quality	Predicted number of Defects in requirements phase	Continuous
	Predicted number of Defects in design phase	Continuous
	Predicted number of Defects in testing phase	Continuous
	Number of defects within one month after deliver	Continuous
Other	Number of requirement changes in requirement phase	Continuous
	Number of requirement changes in design phase	Continuous
	Number of requirement changes in coding phase	Continuous
	Number of requirement changes in testing phase	Continuous

3.2 CSBSG Data Set

CSBSG data set contains 1103 projects from Chinese software industry. It was created in 2006 with its mission to promote Chinese standards of software productivity. CSBSG projects were collected from 140 organizations and 15 regions across China by Chinese association of software industry. Each CSBSG project is described with 179 attributes. The same data preprocessing as those used in ISBSG data set is used on CSBSG data set. In data pruning, 104 projects and 32 attributes (15 nominal attributes and 17 continuous attributes) are extracted from CSBSG data set. Table 4 lists the attributes used in the CSBSG data set.

In data discretization, the values of each continuous attribute are partitioned into 3 unique classes. Dummy variables are added to transfer nominal attributes into Boolean variables. As a result, 261 Boolean variables are produced to describe the 104 projects with missing values denoted as “-1”. The value distribution of variables of CSBSG projects is shown in Table 5 and we can see that CSBSG data set has more missing values than ISBSG data set.

Table 5: The value distribution of variables for describing projects in CSBSG data set.

Value	Proportion
1	$15\% \sim 40\%$
0	$20\% \sim 60\%$
-1	$10\% \sim 33\%$

Finally, the projects in CSBSG data set were categorized into 3 classes according to their real efforts. The projects with “normalized work effort” more than

5,000 person hours were categorized into the class with effort label as “high”, projects with “normalized work effort” between 2,000 and 5,000 person hours as ”medium” and projects with “normalized work effort” less than 2,000 person hours as “low”. Table 6 lists the effort distribution of the selected projects in the CSBSG data set.

Table 6: The effort classes categorized in CSBSG data set.

Class No	Number of projects	Label
1	27	Low
2	31	Medium
3	46	High

4 PREDICTING SOFTWARE PROJECTS WITH UNSUPERVISED LEARNING

4.1 *k*-medoids Clustering and Similarity Measures

Generally, researchers in software engineering hold the assumption that projects with similar characteristics, such as the number of function points, application domain and programming language, are expected to have approximately equivalent efforts (at least they should be in the same effort class). In the standpoint of machine learning, clustering software projects on the basis of a random subset can capture information on the unobserved attributes (Krupka and Tishby, 2008). If we regarded effort as an attribute that also characterize software projects in the data set, then software effort can be deduced by clustering projects using other attributes.

To validate this assumption, *k*-medoids (Theodoridis and Koutroumbas, 2006) is adopted for clustering the projects and three similarity measures are used to measure the similarities of boolean vectors that represent the software projects. *k*-medoids is actually evolved from *k*-means (Theodoridis and Koutroumbas, 2006) and their difference lies in that *k*-medoids assigns existing element in a cluster as cluster center but k-means assigns mean vector of elements in a cluster as the cluster center. We adopt *k*-medoids other than k-means because the mean vector of boolean vectors lacks explainable meaning in practice nevertheless their medoid denotes a real project. The typical *k*-medoids clustering is implemented by partitioning around medoids (PAM) algorithm as depicted in Algorithm 2.1. The computation complexity and the convergence of PAM algorithm refers to

Table 7: Three similarity measure used in *k*-medoids clustering.

Measure	Similarity	Range
Dice	$\frac{A}{2A+B+C}$	$[0,\frac{1}{2}]$
Jaccard	$\frac{A}{A+B+C}$	[0,1]
Kulzinsky	$\frac{A}{B+C}$	∞

(Theodoridis and Koutroumbas, 2006).

Algorithm 1. *The k-medoids clustering implemented by PAM algorithm.*

```
Input:
k, the number of clusters
m, Boolean vectors
Output:
k clusters partitioned from the m Boolean vectors.
Procedure:
1.  Initialize: randomly select k of the m Boolean vectors as the mediods.
2.  Associate each Boolean vector to the closest medoid under predefined similarity measure.
3.  For each mediod d
4.      For each non-medoid Boolean vector b
5.          Swap d and b and compute the total cost of the configuration
6.      End for
7.  End for
8.  Select the configuration with the lowest cost.
9.  Repeat steps 2 to 5 until there is no change in the medoid.
```

The three adopted similarity measures are Dice coefficient, Jaccard coefficient and Kulzinsky coefficient for binary vectors (Gan et al., 2007). Assuming that D_i and D_j are two n-dimensional Boolean vectors and $s_{pq}(D_i,D_j)$ is the number of entries in D_i and D_j whose values are p and q respectively, we define A, B, C and D in Equation 7.

$$A = s_{11}(D_i,D_j), B = s_{01}(D_i,D_j),\\ C = s_{10}(D_i,D_j), D = s_{00}(D_i,D_j) \quad (7)$$

The similarity measures of Dice, Jaccard and Kulzinsky coefficients are listed in Table 7. We regard that D, which means that the characteristic does not exist in both D_i and D_j, might not be an important factor when measuring similarity of two projects because, the proportion of zero in values of variables is very large in both ISBSG and CSBSG data set.

4.2 Clustering Results

PAM algorithm is used to cluster the software projects in ISBSG and CSBSG data sets. The number of clusters is predefined as the number of classes. That is, the

Table 8: *k*-medoids clustering on ISBSG data set.

Similarity Measure	F-measure	
	Without imputation	With imputation
Dice Coefficient	0.3520	0.3937
Jaccard Coefficient	0.3824	0.4371
Kulzinsky Coefficient	0.4091	0.4624

parameter k in PAM algorithm for both ISBSG and CSBSG data sets was set as 3. Without imputation, we regard the missing values in boolean vectors as zeros. We also employed MINI imputation technique (Song and Shepperd, 2007) to impute the missing values before clustering. Due to the unstable clustering results caused by initial selection of cluster centers in PAM algorithm, we repeated each experiment 10 times and ensemble clustering proposed by Zhou et al (Zhou and Tang, 2006) was utilized to produce the final clusters. Table 8 shows the performances of *k*-medoids clustering on ISBSG data set using PAM algorithm with three similarity measures with and without imputation.

We can see from Table 8 that in similarity measure, Kulzinsky coefficient has the best performance among the three measures and Jaccard coefficient has better performance than Dice coefficient. In *k*-medoids clustering without (with) imputation, Kulzinsky coefficient increases the F-measure by 16.29% (17.45%) and Jaccard Coefficient increases the F-measure by 8.6% (5.78%) using Dice coefficient as the baseline.

This outcome illustrates that the number of common entries as in Equation 7 is more important than other indices in similarity measure of software project in effort prediction using clustering. Imputation significantly improves the quality of clustering results. This validates the effectiveness of imputing missing values of projects represented by boolean vectors in *k*-medoids clustering.

To have a detailed look at the clustering results, Table 9 shows the projects in the produced clusters across the classes in Table 3. These clusters were produced by *k*-medoids clustering using Kulzinsky coefficient with imputation (i.e. F-measure is 0.4624). We can see that *k*-medoids clustering actually has not produced high-quality clusters in the ISBSG data set. The results are not good as acceptable in real practice of software effort prediction. For instance, cluster 2 mixes projects in both class 1 and 2 and, most projects in one class scatter on more than one cluster such as the projects in class 2 and class 3.

Table 10 shows the performance of PAM algorithm on CSBSG data set. Table 11 shows the distribution of projects in clusters across classes. Similarly, *k*-medoids clustering has not produced a favorable performance on CSBSG data set. By contrast, the performance of *k*-medoids clustering on CSBSG data set is worse than that on ISBSG data set. Without (with) imputation, the average F-measure on the three coefficients on CSBSG data set is decreased by 7.5% (3.7%) using the average on ISBSG data set as baseline. We explain this outcome as that CSBSG data set has less ones and more missing values (denoted as "-1") in boolean vectors than ISBSG data set, as can be seen in Tables 2 and 5. Based on the analysis, the predictability of software effort using unsupervised learning is not acceptable by software industry.

Table 9: Clustering result using Kulzinsky coefficient with imputation on ISBSG data set.

Similarity	Class 1	Class 2	Class 3	Total
Cluster 1	26	23	28	77
Cluster 2	32	40	27	99
Cluster 3	6	22	45	73
Total	64	85	100	249

5 CLASSIFICATION OF SOFTWARE PROJECTS

5.1 Supervised Learning Techniques

The employed supervised learning techniques are those usually used in effort prediction, including J48 decision tree, BPNN and naïve Bayes. The J48 decision tree classifier follows the following simple algorithm. In order to classify the effort of a software project, it firstly creates a decision tree based on the values of variables in the training data set. Whenever it encounters a set of boolean vectors (training set) it identifies the variable that has the largest information gain (Quinlan, 1993). Among the possible values of this variable, if there is any value for which there is no ambiguity, that is, for which the projects falling within this value having the same label of effort, then we terminate that branch and assign to the terminal node the label of effort.

The back propagation neural network (BPNN) (Rumelhart et al., 1986) is used to classify the soft-

Table 10: *k*-medoids clustering on CSBSG data set.

Similarity Measure	F-measure	
	Without imputation	With imputation
Dice Coefficient	0.3403	0.3772
Jaccard Coefficient	0.3881	0.4114
Kulzinsky Coefficient	0.4065	0.4560

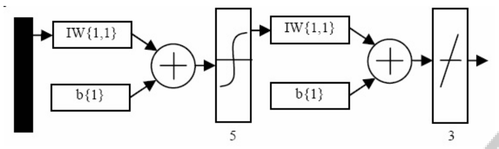

Figure 1: BPNN with 5 nodes in hidden layer and 3 nodes in output layer.

ware projects in both ISBSG and CSBSG data sets as well. BPNN defines two sweeps of the network: first a forward sweep from the input layer to the output layer and second a backward sweep from the output layer to the input layer. The back ward sweep is similar to the forward sweep except that error values are propagated back through the network to determine how the weights of neurons are to be changed during training. The objective of training is to find a set of network weights of neurons that construct a model for prediction with minimum error.

Table 11: Clustering result using Kulzinsky coefficient with imputation on CSBSG data set.

Similarity	Class 1	Class 2	Class 3	Total
Cluster 1	9	11	16	36
Cluster 2	10	10	17	37
Cluster 3	8	10	13	31
Total	27	31	46	104

A three-layer fully connected feed-forward network which consists of an input layer, a hidden layer and an output layer is adopted in the experiments. The "tansigmod" function is used in the hidden layer with 5 nodes and "purelinear" function for the output layer with 3 nodes [17]. The network of BPNN is designed as shown in Figure 2.

Naïve Bayes (Duda et al., 2003) is a well known probabilistic classifier in machine learning. It is based on the Bay's theorem of posteriori probability and assumes that the effect of an attribute value on a given class is independent of the value of the other attributes. This class conditional independence assumption simplifies computation involved in building the classifier so we called the produced classifier "naive". Compared to other traditional prediction models, naïve Bayes provides tools for risk estimation and allows decision-makers to combine historical data with subjective expert estimates (Pendharkar et al., 2005).

The J48 decision tree and naïve Bayes are implemented using Weka (Waikato Environment for Knowledge Analysis) (http://www.cs.waikato.ac.nz/ml/weka/) and, BPNN is implemented using Matlab simulink tool box (http://www.mathworks.com/products/neural-net/). Also, MINI algorithm [12] is used to impute the missing values of Boolean vectors if necessary. Our experiments were carried out with 10-flod cross-validation technique. For each experiment, we divided the whole data set (ISBSG or CSBSG data set) into 10 subsets. 9 of 10 subsets are used for training and the remaining 1 subset was used for testing. We repeat the experiment 10 times and, the performance of the prediction model is measured by the average of 10 accuracies of the 10 repetitions.

5.2 Classification Result

Table 12 shows the performances of the three mentioned classifiers in classifying the projects in ISBSG data set. On average, we can see that BPNN outperforms other classifiers in classifying the software projects based on efforts. J48 decision tree has better performance than naïve Bayes. Using the performance of naïve Bayes as the baseline, BPNN increases the average accuracy by 16.25% (11.71%) and J48 decision tree by 5.6% (2.5%) without (with) imputation.

We explain this outcome that BPNN has the best capacity to eliminate the noise and peculiarities because it adopts back sweep to change the weights of neurons for reducing errors of predictive model. However, the performance of BPNN is not robust as other classifiers (we observe this point from its standard deviation). The adoption of cross-validation technique may reduce overfitting of BPNN to some extent but, it cannot eliminate the drawback of BPNN entirely. The J48 decision tree classifies the projects using learned decision rules. Due to the adoption of information gain [15], those variables having more discriminative power will be fetched out by J48 in earlier branches in constructing decision rules and thus, the noise and peculiarities connotated in the variables with less discriminative power will be ignored automatically (especially in tree pruning).

naïve Bayes has the worst performance among the three classifiers in classifying software efforts. We explain this as that the variables used for describing projects may not be independent of each other. Moreover, naïve Bayes regard all variables as has equivalent weights as each other in the prediction model. The conditional probabilities of all variables have the same weight when predicting the label of an incoming project. However, in fact, some variables of projects have more discriminative power than other variables in deciding the project effort. The noise and peculiarities are often contained in the variables those have little discriminative power and those variables should be given less importance in the prediction model. We conjecture that this fact is also the cause of the poor performance of *k*-medoids in project clustering projects. In the same manner as that in *k*-medoids clustering, MINI technique has significantly improved the performance of project classification by imputing missing values in Boolean vectors.

Table 13 shows the performances of the three classifiers on CSBSG data set. The similar conclusion as on ISBSG data set can be drawn on CSBSG data set. However, the performances of three classifiers on CSBSG data set are worse than those on ISBSG data set. The average of overall accuracies of the three techniques without (with) imputation on CSBSG data set is decreased by 6.95% (6.66%) using that on ISBSG data set as the baseline. We also explain this outcome as the lower quality of CSBSG data set than that of ISBSG data set.

We can see from Tables 12 and 13 that, in both ISBSG and CSBSG data sets, all the three supervised learning techniques have not produced a favorable classification on software efforts using project attributes. The best performance that was produced by BPNN is with the accuracy around 60%. The accuracy as 60% is meaningless for software effort prediction in most cases because, that means that at the probability 0.4, the prediction results fall beyond the range of each effort class. Combined with the results of effort prediction from unsupervised learning, we draw that the predictability of software effort using supervised learning techniques is not acceptable by software industry, either.

6 RELATED WORK

Srinivasan and Fisher (Srinivasan and Fisher, 1995) used decision tree and BPNN to estimate software development effort. COCOMO data with 63 historical projects was used as the training data and Kremer data with 15 projects was used as testing data. They reported that decision tree and BPNN are competitive with traditional COCOMO estimator. However, they pointed out that the performances of machine learning techniques are very sensitive to the data on which they were trained. (Finnie et al., 1997) compared three estimation techniques as BPNN, case-based reasoning and regression models using Function Points as the measure of system size. They reported that neither of case-based reasoning and regression model was favorable in estimating software efforts due to the considerable noise in the data set. BPNN appears capable of providing adequate estimation performance (with MRE as 35%) nevertheless its performance is largely dependent on the quality of training data as well as the suitability of testing data to the trained model. Of all the three methods, a large amount of uncertainty is inherent in their performances. In both (Finnie et al., 1997) and (Srinivasan and Fisher, 1995), a serious problem confronted with effort estimation using machine learning techniques is that huge uncertainty involved in the robustness of these techniques. That is, model sensitivity and data-dependent property of machine learning techniques hinder their admittance by industrial practice in effort prediction. These work as well as (Prietula et al., 1996) motivates this study to investigate the effectiveness of a variety of machine learning techniques on two different data sets.

(Park and Baek, 2008) conducted an empirical validation of a neural network model for software effort estimation. The data set used in their experiments is collected from a Korean IT company and includes 148 IT projects. They compared expert judgment, regression models and BPNN with different input variables in software effort estimation. They reported that neural network using Function Point and other 6 variables (length of project, usage level of system development methodology, number of high/middle/low level manpower and percentage of outsourcing) as input variables outperforms other estimation methods. However, even in the best performance, the average MRE is nearly 60% with standard deviation more than 30%. This result makes it is very hard that the method proposed in their work can be satisfactorily admitted in practice. For this reason, a validation of machine learning methods is necessary in order to shed light on the advancement of software effort estimation. This point also motivates us to investigate the effectiveness of machine learning techniques for software effort estimation and the predictability of software effort using machine techniques.

(Shukla, 2000) proposed a neuron-genetic approach to predict software development effort while

Table 12: Classification of software project efforts on ISBSG data set.

Classifier	Average accuracy ± Standard Deviation	
	Without imputation	With imputation
J48 decision tree	0.5706 ± 0.1118	0.5917 ± 0.1205
BPNN	0.6281 ± 0.1672	0.6448 ± 0.1517
naïve Bayes	0.5403 ± 0.1123	0.5772 ± 0.1030

Table 13: Classification of software project efforts on CSBSG data set.

Classifier	Average accuracy ± Standard Deviation	
	Without imputation	With imputation
J48 decision tree	0.4988 ± 0.1103	0.5341 ± 0.1322
BPNN	0.1650 ± 0.1650	0.6132 ± 0.1501
naïve Bayes	0.5331 ± 0.1221	0.5585 ± 0.0910

the neural network is employed to construct the prediction model and genetic algorithm is used to optimize the weights between nodes in the input layer and the nodes in the output layer. They used the same data sets as that was used in Srinivasan and Fisher (Srinivasan and Fisher, 1995) and reported that the neuron-genetic approach outperforms both decision tree and BPNN. However, they also reported that local minima and over fitting deteriorate the performance of the proposed method in some cases, even make it a poorer predictor than traditional estimator as COCOMO (Beohm, 1981). The focus of our study is not to propose a novel approach to software effort estimation but to extensively review the usefulness of machine learning techniques in software effort estimation. That is, to how much extent the typical machine techniques can accurately estimate the effort of a given project using historical data.

7 CONCLUDING REMARKS

In this paper, we conducted a series experiments to investigate the predictability of software effort using machine learning techniques. With ISBSG and CSBSG data sets, unsupervised learning as k- medoids clustering is used to cluster software projects with respect to efforts and, supervised learning as J48 decision tree, BPNN and naive Bayes are used to classify the projects. Our assumption for this investigation is that the efforts of software projects can be deduced from the values of other attributes in historical data and projects with similar values on attributes other than effort will also have approximately equivalent efforts.

The experimental results demonstrate that neither unsupervised nor supervised learning techniques can provide software effort prediction with a favorable model. Despite of this fact, Kulzinsky coefficient has produced the best performance in similarity measure for unsupervised learning and, BPNN has produced the best performance among the examined supervised learning techniques. Moreover, the MINI imputation can improve data quality and improve effort prediction significantly.

ACKNOWLEDGEMENTS

This work is supported by the National Natural Science Foundation of China under Grant Nos. 6087307261073044, and 60903050; the National Science and Technology Major Project; the National Basic Research Program under Grant No. 2007CB310802; the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry.

REFERENCES

Beohm, B. (1981). *Software Engineering Economics*. Prentice-Hall, New Jersey, USA, 2nd edition.

Duda, R., Hart, P., and Stork, D. (2003). *Pattern Classification*. John Wiley & Sons, 2nd edition.

Finnie, G., Wittig, G., and Desharnais, J. (1997). A comparison of software effort estimation techniques: Using function points with neural networks, case-based reasoning and regression models. *Journal of Systems and Software*, 39:281–289.

Gan, G., Ma, C., and Wu, J. (2007). Data clustering, theory, algorithmsm, and applications. *ASA-SIAM Series on Statistical and Applied Probability*, page 78.

He, M., Li, M., Wang, Q., Yang, Y., and Ye., K. (2008). An investigation of software development productivity in china. In *Proceedings of International Conference on Software Process*, pages 381–394.

Jorgensen, M. (2004). A review of studies on expert estimation of software development effort. *Journal of Systems and Software*, 70:37–60.

Korte, M. and Port, D. (2008). Confidence in software cost estimation results based on mmre and pred. In *Proceedings of PROMISE'08*, pages 63–70.

Krupka, E. and Tishby, N. (2008). Generalization from observed to unoberserved features by clustering. *Journal of Machine Learning Research*, 83:339–370.

Park, H. and Baek, S. (2008). An empirical validation of a neural network model for software effort estimation. *Expert System with Applications*, 35:929–937.

Pendharkar, P., G.Subramanian, and J.Roger (2005). A probabilistic model for predicting software development effort. *IEEE Transactions on Software Engineering*, 31(7):615–624.

Prietula, M., Vicinanza, S., and Mukhopadhyay, T. (1996). Software-effort estimation with a case-based resoner. *Journal of Experimental & Theoritical Artificial Intelligence*, 8:341–363.

Quinlan, J. (1993). *Programs for Machine Learning*. Morgan Kaufmann Publishers, 2nd edition.

Rumelhart, D., Hinton, G., and Williams, J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing, Exploitations in the Microstructure of Cognition*, pages 318–362.

Shukla, K. (2000). Neuro-genetic prediction of software development effort. *Information and Software Technology*, 42:701–713.

Song, Q. and Shepperd, M. (2007). A new imputation method for small software project data sets. *Journal of Systems and Software*, 80:51–62.

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

Steinbach, M., Karypis, G., and Kumar, V. (2000). A comparison of document clustering techniques. In *KDD-2000 Workshop on Text Mining*, pages 109–110.

Theodoridis, S. and Koutroumbas, K. (2006). *Pattern Recognition*. Elsevier, 3rd edition.

Yang, Y., Wang, Q., and Li, M. (2009). Process trustworthiness as a capability indicator for measuring and improving softwaer trustworthiness. In *Proceedings of ICSP 2009*, pages 389–401.

Zhou, Z. and Tang, W. (2006). Clusterer ensemble. *Knowledge-Based Systems*, 19:77–83.