Soft Comput
DOI 10.1007/s00500-014-1576-2

FOCUS

Three empirical studies on predicting software maintainability using ensemble methods

Mahmoud O. Elish · Hamoud Aljamaan · Irfan Ahmad

Abstract More accurate prediction of software maintenance effort contributes to better management and control of software maintenance. Several research studies have recently investigated the use of computational intelligence models for software maintainability prediction. The performance of these models, however, may vary from dataset to dataset. Consequently, ensemble methods have become increasingly popular as they take advantage of the capabilities of their constituent computational intelligence models toward a dataset to come up with more accurate or at least competitive prediction accuracy compared to individual models. This paper investigates and empirically evaluates different homogenous and heterogeneous ensemble methods in predicting software maintenance effort and change proneness. Three major empirical studies were designed and conducted taken into consideration different design such as the types of the investigated ensembles methods, types of prediction problems, used datasets, and other experimental setup. Overall empirical evidence obtained from the three studies confirms that some ensemble methods provide more accurate or at least competitive prediction accuracy compared to individual models across datasets, and thus they are more reliable.

Keywords Computational intelligence · Ensemble techniques · Homogenous ensemble · Heterogeneous ensemble · Software maintenance · Prediction · Empirical studies

Communicated by I. R. Ruiz.

M. O. Elish (✉) · H. Aljamaan · I. Ahmad
Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia
e-mail: elish@kfupm.edu.sa

H. Aljamaan
e-mail: hjamaan@kfupm.edu.sa

I. Ahmad
e-mail: irfanics@kfupm.edu.sa

1 Introduction

Software maintenance has been one of the most difficult and costly tasks in the software development lifecycle (Li and Henry 1993; Zhou and Leung 2007). Accurate prediction of software maintainability can be useful to support and guide (De Lucia et al. 2005): software-related decision-making; maintenance process efficiency; comparing productivity and costs among different projects; resource and staff allocation, and so on. As a result, future maintenance effort can be kept under control. Recent research studies have investigated the use of computational intelligence models for software maintainability prediction (Elish and Elish 2009; Koten and Gray 2006; Zhou and Leung 2007). These models have different prediction capabilities and none of them has proved to be the best under all conditions. Performance of these models may vary from dataset to dataset. Ensemble methods take advantage of the capabilities of their constituent computational intelligence models (base learners) toward a dataset to come up with more accurate or, at least, competitive prediction accuracy as compared to the individual models. They have high potential in providing reliable predictions. Therefore, there is a need for empirical evidences on the effectiveness of ensemble methods and the extent to which these ensembles enhance the accuracy, or in some cases deteriorate the prediction accuracy.

In this research, we conducted three empirical studies on predicting software maintainability using ensemble methods. These studies differ in terms of types of the investigated ensembles methods (homogenous and heterogeneous), types of prediction problems (maintenance effort and change-

Published online: 08 January 2015

proneness), used datasets, and other experimental setup. The objective was to investigate and empirically evaluate different ensemble methods with respect to prediction accuracy, and to compare them among themselves and against individual models. This work is a significant extension of the preliminary work reported in Aljamaan et al. (2013) where some experiments were carried out to investigate the use of one ensemble method for software maintenance effort prediction.

This paper reports the details of the three conducted empirical studies and their results. The first study aimed to evaluate and compare three heterogeneous ensemble methods in predicting software maintenance effort. The purpose of the second study was to evaluate and compare two homogeneous ensemble methods in predicting object-oriented class change proneness. The third study was conducted to evaluate and compare three heterogeneous ensemble methods in predicting object-oriented class change proneness. According to the best knowledge of the authors, there is no other work in the published literature which reports such a comprehensive study of ensemble models for software maintenance effort and change proneness prediction in terms of the use of different datasets, variety of individual computational models used, and the different approaches to ensemble.

The rest of this paper is organized as follows. Section 2 reviews the related work. In Sect. 3, we provide an overview of the ensemble methods of computational intelligence models. We describe the three empirical studies that were conducted and we provide the analysis of the results in Sects. 4, 5, and 6; one empirical study per section. In Sect. 7, we present the conclusions and suggest directions for future work.

2 Related work

Several research studies have investigated the relationship between object-oriented metrics and the maintainability of object-oriented software systems, and they found significant correlations between them (Al-Dallal 2013; Bandi et al. 2003; Briand et al. 2001; Fioravanti and Nesi 2001; Li and Henry 1993; Misra 2005). These metrics can thus be used as good predictors of software maintainability. Furthermore, recent studies have investigated the use of computational intelligence models for software maintainability prediction. These models were constructed using object-oriented metrics as input variables. Such models include TreeNet (Elish and Elish 2009), multivariate adaptive regression splines (Zhou and Leung 2007), Naïve bayes (Koten and Gray 2006), artificial neural network (Thwin and Quah 2005; Zhou and Leung 2007), regression tree (Koten and Gray 2006; Zhou and Leung 2007), and support vector regression (Zhou and Leung 2007), and Mamdani fuzzy inference engine (Ahmed and Al-Jamimi 2013).

Thwin and Quah (2005) predicted the software maintainability as the number of lines changed per class. Their experimental results found that general regression neural network predict maintainability more accurately than Ward network model. Koten and Gray (2006) evaluated and compared the naïve bayes classifier with commonly used regression-based models. Their results suggest that the naïve bayes model can predict maintainability more accurately than the regression-based models for one system, and almost as accurately as the best regression-based model for the other system. Zhou and Leung (2007) explored the employment of multiple adaptive regression splines (MARS) in building software maintainability prediction models. MARS was evaluated and compared against multivariate linear regression models, artificial neural network models, regression tree models, and support vector models. Their results suggest that, for one system, MARS can predict maintainability more accurately than the other four typical modeling techniques. Then, Elish and Elish (2009) extended the work done by Zhou and Leung (2007) to investigate the capability of TreeNet technique in software maintainability prediction. Their results indicate that TreeNet can yield improved, or at least competitive, prediction accuracy over previous maintainability prediction models.

Recently, ensemble methods have received much attention and have demonstrated promising capabilities in improving the accuracy over single models (Braga et al. 2007; Sollich 1996). Ensemble methods have been used in the area of software engineering prediction problems. For example, they have been used in software reliability prediction (Zheng 2009), software project effort estimation (Braga et al. 2007; Elish et al. 2013), and software fault prediction (Aljamaan and Elish 2009; Khoshgoftaar et al. 2003). In addition, they have been used in many real applications such as face recognition (Gutta and Wechsler 1996; Huang et al. 2000), OCR (Mao 1998), seismic signal classification (Shimshoni and Intrator 1998) and protein structural class prediction (Bittencourt et al. 2005). To the best of our knowledge, ensemble methods have not been explored in predicting software maintainability expect our preliminary work reported in Aljamaan et al. (2013). In that work (Aljamaan et al. 2013), we proposed and empirically evaluated one ensemble method of computational intelligence models for predicting software maintenance effort. The results confirm that the proposed ensemble method provides more accurate prediction compared to individual models, and thus it is more reliable.

This paper is a significant extension of the preliminary work reported in Aljamaan et al. (2013), and it differs from the above related works in several aspects. This paper investigates and compares different homogeneous and heterogeneous ensemble methods in software maintainability prediction problems. We considered maintenance effort prediction (regression problem) and also change-proneness prediction (classification problem). Furthermore, different combination

Table 1 Comparison of the three empirical studies conducted in this research

Study	Problem type	Datasets	Ensemble methods	Base learner(s)	Combination rule(s)
Empirical study I (Sect. 4)	Regression (maintenance effort)	UIMS (39) and QUES (71)	Heterogeneous	MLP, RBF, SVM, M5P	Linear (averaging)
			Heterogeneous	MLP, RBF, SVM, M5P	Linear (weighted averaging)
			Heterogeneous	MLP, RBF, SVM, M5P	Linear (best in training)
Empirical study II (Sect. 5)	Classification (change proneness)	VSSPLUGIN (36) and PeerSim (60)	Homogeneous (bagging)	MLP, RBF, SVM, DT	Linear (averaging)
			Homogeneous (Boosting)	MLP, RBF, SVM, DT	Linear (averaging)
Empirical study III (Sect. 6)	Classification (change proneness)	VSSPLUGIN (36) and PeerSim (60)	Heterogeneous	SVM, MLP, logistic regression, genetic programming, K-means	Linear (best in training)
			Heterogeneous	SVM, MLP, logistic regression, genetic programming, K-means	Linear (majority voting)
			Heterogeneous	SVM, MLP, logistic regression, genetic programming, K-means	Non-linear (DTF)

rules (linear and non-linear) for the ensemble methods were investigated.

3 Ensembles of computational intelligence models

An ensemble of computational intelligence models uses the outputs of all its individual constituent prediction models (base learners), each being assigned a certain priority level, and provide the final output with the help of an arbitrator (combination rule) (Optiz and Maclin 1999). There are homogenous (single-model) ensembles and heterogeneous (multi-model) ensembles. In homogenous ensembles, the individual base learners are of the same type (for example, all of them could be radial basis function network), but each with randomly generated training set. Examples of homogenous ensembles include bagging (Breiman 1996) and boosting (Freund 1995). In heterogeneous ensembles, there are different individual base learners.

The ensemble methods can be further classified, according to the design of their arbitrator, into linear ensembles and nonlinear ensembles (Kiran and Ravi 2008). In linear ensembles, the arbitrator combines the outputs of the base learners in a linear fashion such as averaging, weighted averaging, etc. In nonlinear ensembles, no assumptions are made about the input that is given to the ensemble (Kiran and Ravi 2008). The output of the individual base learners are fed into an arbitrator, which is a nonlinear prediction model such as neural network which when trained, assigns the weights accordingly.

In this research, we conducted three empirical studies. In each study, we developed different ensemble methods, and then evaluated and compared their prediction performance in a software maintainability prediction problem. Table 1 provides a summary comparison of the three conducted empirical studies. The details of these empirical studies, their results and analysis are provided in the following sections.

4 Empirical study I

The goal of this empirical study is to evaluate and compare three heterogeneous ensemble methods (i.e., heterogeneous ensembles with three different linear combination rules) in predicting software maintenance effort.

4.1 Ensemble methods

4.1.1 Average-based ensemble

Average-based (AVG) ensemble is the simplest ensemble method, where each constituent model in the ensemble has the same weight. For each observation in the dataset, the output (predicted) values of the individual prediction models are taken as inputs to the arbitrator that outputs the average of these values. Figure 1 provides a formal description of the AVG ensemble method.

4.1.2 Weighted-based ensemble

In weighted-based (WT) ensemble, individual output values by the prediction models in the ensemble are given weights based upon a certain criterion. In this study, the criterion is mean magnitude of relative error (MMRE); the lower the

Choose dataset with N observations
Choose M individual prediction models
Set K for K folds cross validation
For each $k \in K$ fold
  For each $n \in N$ observation in the testing set for fold(k)
    For each $m \in M$ model
      Apply model m on observation n
      Store the result as R(m)
    End for
    $$EnsembleOutput = \frac{\sum_{m=1}^{M} R(m)}{M}$$
  End for
End for

Fig. 1 AVG ensemble

Choose dataset with N observations
Choose M individual prediction models
Set K for K folds cross validation
For each $k \in K$ fold
  For each $m \in M$ model
    Apply model m on the training set for fold(k)
    Calculate training error E, based on a certain criterion
    Store error E
  End for
  Rank all M models based on their training error E
  For each $n \in N$ observation in the testing set for fold(k)
    For each $m \in M$ model
      Apply model m on observation n
      Multiply model m output by its rank
      Store the result as WR(m)
    End for
    $$EnsembleOutput = \frac{\sum_{m=1}^{M} WR(m)}{\sum_{i=1}^{M} i}$$
  End for
End for

Fig. 2 WT ensemble

MMRE the higher the weight. Figure 2 provides a formal description of the WT ensemble method.

4.1.3 Best-in-training-based ensemble

Best-in-training-based (BT) ensemble takes the advantage of the fact that individual prediction models have different errors across the used dataset partitions. The idea behind this ensemble method is to take across the dataset partitions, the best model in training based upon a certain criterion in that partition. In this study, the criterion is MMRE. Figure 3 provides a formal description of the BT ensemble method.

4.2 Base learners

In this section, we briefly describe the individual computational intelligence models that were used as base learners for the ensemble methods in this empirical study. These models were built using WEKA machine learning toolkit (Misra 2005), and their parameters were initialized using the default values.

Choose dataset with N observations
Choose M individual prediction models
Set K for K folds cross validation
For each $k \in K$ fold
  For each $m \in M$ model
    Apply model m on the training set for fold(k)
    Calculate training error E, based on a certain criterion
    Store error E
  End for
  Select the best model $b \in M$, based on training error E
  For each $n \in N$ observation in the testing set for fold(k)
    EnsembleOutput = the result of applying model b on observation n
  End for
End for

Fig. 3 BT ensemble

4.2.1 Multilayer perceptron

Multilayer perceptron (MLP) (Haykin 1999) are feed-forward networks that consist of an input layer, one or more hidden layers of nonlinearly activating nodes and an output layer. Each node in one layer connects with a certain weight to every other node in the following layer. MLP uses back-propagation algorithm as the standard learning algorithm for any supervised-learning.

The parameters of this model were initialized as follows. Back-propagation algorithm was used for training. Sigmoid was used as an activation function. Number of hidden layers was 5. Learning rate was 0.3 with momentum 0.2. Network was set to reset with a lower learning rate. Number of epochs to train through was 500. Validation threshold was 20.

4.2.2 Radial basis function network

Radial basis function network (RBF) (Poggio and Girosi 1990) is an artificial neural network that uses radial basis functions as activation functions to provide a flexible way to generalize linear regression function. Commonly used types of radial basis functions include Gaussian, multi-quadric, and poly-harmonic spline. RBF models with Gaussian basis functions possess desirable mathematical properties of universal approximation and best approximation. A typical RBF model consists of three layers: an input layer, a hidden layer with a non-linear RBF activation function and a linear output layer.

The parameters of this model were initialized as follows. A normalized Gaussian radial basis function network was used. Random seed to pass on to K-means clustering algorithm was 1. Number of clusters for K-means clustering algorithm to generate was 2, with minimum standard deviation for clusters set to 0.1.

4.2.3 Support vector machines

Support vector machines (SVMs) were proposed by Vapnik (1995) based on the structured risk minimization (SRM) principle. SVMs are a group of supervised learning methods that can be applied to classification or regression problems. SVMs aim to minimize the empirical error and maximize the geometric margin. SVM model is defined by these parameters: complexity parameter C, extent to which deviations are tolerated ε, and kernel.

The parameters of this model were initialized as follows. The cost parameter C was set to 1, with polynomial as SVM-reg kernel. The most popular (RegSMOImproved) algorithm Shevade et al. (2000) was used for parameter learning.

4.2.4 M5 model tree

M5 model tree (M5P) (Quinlan 1992; Witten and Frank 2005) is an algorithm for generating M5 model trees that predicts numeric values for a given instance. To build a model tree, the M5 algorithm starts with a set of training instances. The tree is built using a divide-and-conquer method. At a node, starting with the root node, the instance set that reaches it is either associated with a leaf or a test condition is chosen that splits the instances into subsets based on the test outcome. In M5, the test that maximizes the error reduction is used. Once the tree has been built, a linear model is constructed at each node. The linear model is a regression equation.

The parameters of this model were initialized as follows. M5 algorithm was used for generating M5 model trees (Quinlan 1992; Wang and Witten 1997). Pruned M5 model trees were built, with three instances as the minimum number of instances allowed at a leaf node.

4.3 Datasets

We used two popular object-oriented software maintainability datasets published by Li and Henry (1993): UIMS and QUES datasets. These datasets are publicly available which makes our study verifiable, repeatable, and reputable (Bradley 1997). The UIMS dataset contains class-level metrics data collected from 39 classes of a user interface management system, whereas the QUES dataset contains the same metrics collected from 71 classes of a quality evaluation system. Both systems were implemented in Ada. Both datasets consist of 11 class-level metrics: ten independent variables and one dependent variable.

The independent (input) variables are five Chidambar and Kemerer metrics (Chidamber and Kemerer 1994): WMC, DIT, NOC, RFC, and LCOM; four Li and Henry metrics (Li and Henry 1993): MPC, DAC, NOM, SIZE2; and one traditional lines of code metric (SIZE1). Table 2 provides brief description for each metric.

Table 2 Independent variables in the datasets for empirical study I

Metric	Description
WMC	Count of methods implemented within a class
DIT	Level for a class within its class hierarchy
NOC	Number of immediate subclasses of a class
RFC	Count of methods implemented within a class plus the number of methods accessible to an object class due to inheritance
LCOM	The average percentage of methods in a class using each data field in the class subtracted from 100 %
MPC	The number of messages sent out from a class
DAC	The number of instances of another class declared within a class
NOM	The number of methods in a class
SIZE1	The number of lines of code excluding comments
SIZE2	The total count of the number of data attributes and the number of local methods in a class

The dependent (output) variable is a maintenance effort proxy measure, which is the actual number of lines in the code that were changed per class during a 3-year maintenance period. A line change could be an addition or a deletion. A change in the content of a line is counted as a deletion and an addition (Li and Henry 1993).

Previous studies (Elish and Elish 2009; Koten and Gray 2006; Zhou and Leung 2007), on both datasets, indicate that both datasets have different characteristics, and therefore, considered heterogeneous and a separate maintenance effort prediction model is built for each dataset.

4.4 Performance evaluation measures

We used de facto standard and commonly used accuracy evaluation measures that are based on magnitude of relative error (MRE) (Conte et al. 1986). These measures are mean magnitude of relative error (MMRE), standard deviation magnitude of relative error (StdMRE), and prediction at level q (Pred(q)). MMRE over a dataset of n observations is calculated as follows:

$$\text{MMRE} = \frac{1}{n}\sum_{i=1}^{n}\text{MRE}_i$$

where MRE_i is a normalized measure of the discrepancy between the actual value (x_i) and the predicated value ($\hat{x}_i$) of observation i. It is calculated as follows:

$$\text{MRE}_i = \frac{\left|x_i - \hat{x}_i\right|}{x_i}.$$

In addition to MMRE, we used StdMRE since it is less sensitive to the extreme values compared to MMRE. We also used Pred(q), which is a measure of the percentage of obser-

Table 3 Prediction accuracy results: UIMS dataset

	Individual models				Ensemble methods		
	MLP	RBF	SVM	M5P	AVG	WT	BT
MMRE	1.39	3.23	1.64	1.67	1.46	1.21	**0.97**
StdMRE	2.40	4.43	2.38	2.75	2.08	1.78	**1.61**
Pred(0.3)	23.33	15	20	23.33	23.33	23.33	**25**

vations whose MRE is less than or equal to q. It is calculated as follows:

$$\text{Pred}(q) = \frac{k}{n},$$

where k is the number of observations whose MRE is less than or equal to a specified level q, and n is the total number of observations in the dataset. An acceptable value for level q is 0.3, as indicated in the literature (Conte et al. 1986; Koten and Gray 2006; Zhou and Leung 2007). We therefore adopted that value.

4.5 Results and analysis

We used a tenfold cross validation (Kohavi 1995) (i.e., k-fold cross validation, with k set to 10). In tenfold cross validation; a dataset is randomly partitioned into tenfolds of equal size. For ten times, ninefolds are picked to train the models and the remaining fold is used to test them, each time leaving out a different fold.

Table 3 provides the results obtained from applying the individual computational intelligence models on UIMS dataset, as well as the results achieved by the ensemble methods. Among the individual models, the MLP model achieved the best result in general, whereas the RBF model was the worst. Among the ensemble methods, the BT ensemble method achieved the best result (bold).

Figure 4 shows the box plot of MRE values for each model on UIMS dataset, where the middle of each box represents the MMRE for each model. As can be seen, the BT ensemble method has the narrowest box and the smallest whiskers (i.e., the lines above and below from the box). Moreover, its box and whiskers are lower than those of the individual models, which clearly indicate that the BT ensemble method outperforms the individual models. Moreover, all the ensemble methods were generally better than the individual models. Figure 5 shows a histogram of the achieved Pred(0.30) value by each model. Clearly, each of the three ensemble methods (AVG, WT, and BT) achieved a Pred(0.30) value that is more than or equal to the achieved value by any of the individual models (MLP, RBF, SVM, and M5P).

Table 4 provides the results obtained from applying the individual computational intelligence models on QUES dataset, as well as the results achieved by the ensemble methods under investigation. Among the individual models, the SVM model achieved the best result, whereas the RBF model was the worst. Among the ensemble methods, the BT ensemble method achieved the best result (bold).

Figure 6 shows the box plot of MRE values for each model on QUES dataset, where the middle of each box represents the MMRE for each model. It can be observed that the BT ensemble method has the narrowest box and the smallest whiskers. Its box and whiskers are also lower than those of the individual models, which clearly indicate that the BT ensemble model outperforms the individual models in this dataset too. Figure 7 shows a histogram of the achieved Pred(0.30) value by each model. The BT ensemble method achieved the

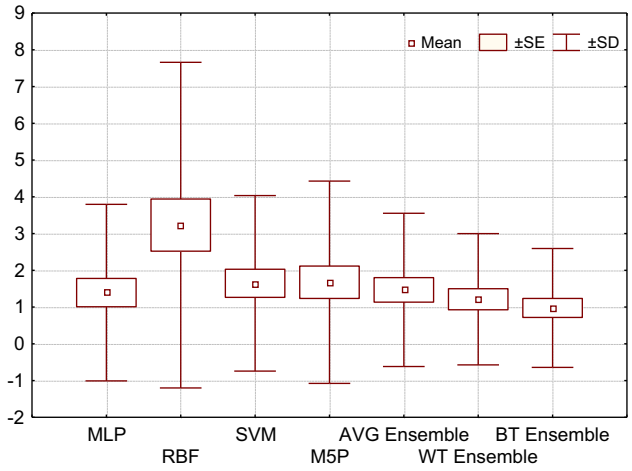

Fig. 4 Box plots of MRE for each model: UIMS dataset

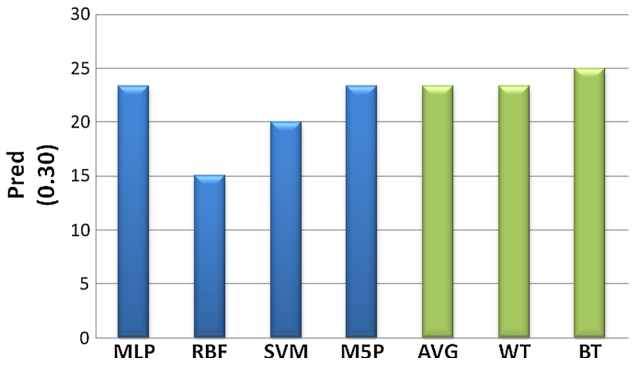

Fig. 5 Pred(0.30) for each model: UIMS dataset

Table 4 Prediction accuracy results: QUES dataset

	Individual models				Ensemble methods		
	MLP	RBF	SVM	M5P	AVG	WT	BT
MMRE	0.71	0.96	0.44	0.54	0.58	0.49	**0.41**
StdMRE	0.65	1.52	0.39	0.56	0.69	0.51	**0.32**
Pred(0.3)	40	36.66	56.66	51.66	53.33	53.33	**60**

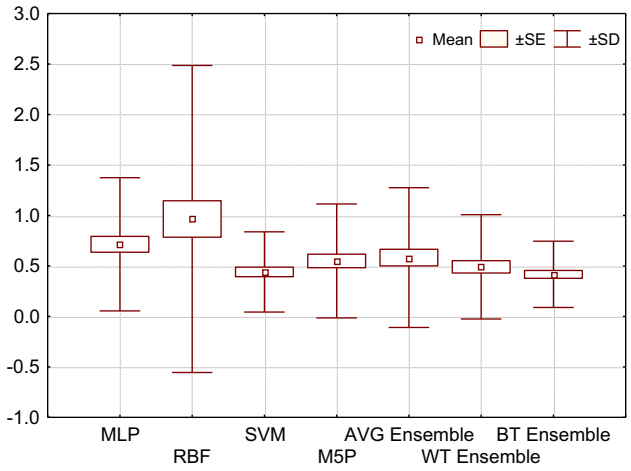

Fig. 6 Box plots of MRE for each model: QUES dataset

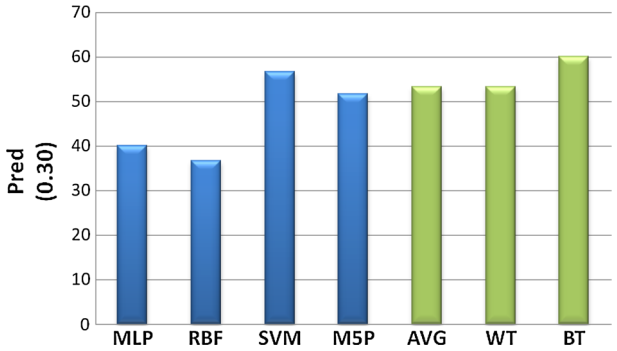

Fig. 7 Pred(0.30) for each model: QUES dataset

highest Pred(0.30) value, i.e., 60 %. Furthermore, each of the three ensemble methods (AVG, WT, and BT) achieved a Pred(0.30) value that is more than the achieved value by each of the individual models except the SVM model. However, the Pred(0.30) values of the AVG and WT ensemble methods were slightly less than the Pred(0.30) value of the SVM model.

When considering the results from both datasets, there are a number of interesting observations. First, the results support that the performance of the individual prediction models may vary from dataset to dataset; the MLP model was the best in the UIMS dataset while the SVM model was the best in the QUES dataset. Second, the BT ensemble method outperformed all other ensemble and individual models in both datasets. Third, among the ensemble methods, the BT method was the best followed by the WT method and then the AVG method. Finally, ensemble methods generally achieved more accuracy or at least competitive prediction accuracy compared to individual models.

model generation
Let n be the number of instances in the training data
For each of t iterations:
    Sample n instances with replacement from training data
    Apply the learning algorithm to the sample
    Store the resulting model
classification
For each of the t models:
    Predict class of instance using model
Return class that has been predicted most often

Fig. 8 Bagging ensemble

5 Empirical study II

The goal of this empirical study is to evaluate and compare two homogeneous ensemble methods in predicting class change proneness.

5.1 Ensemble methods

5.1.1 Bagging ensemble

Bagging, short for bootstrap aggregating, is an ensemble technique proposed by Breiman (1996) to improve the accuracy of classification models by combining classifications of same type (i.e., based on the same base classifier) of randomly generated training sets. Bagging assigns equal weight to models created, thus helps in reducing the variance associated with classification, which in turn improves the classification process. Bagging technique has produced good results whenever the learning algorithm is unstable (Breiman 1996). Figure 8 states the bagging algorithm (Witten and Frank 2005):

Bagging technique requires three parameters: (1) classifier, the base classifier to apply bagging on; (2) bagSizePercent, size of each bag, as a percentage of the training set size; and (3) numIterations, number of instances of the base classifiers to be created, i.e., the ensemble size. In this study, we prefer to use the term ensemble size for clarity purpose.

5.1.2 Boosting ensemble

Boosting is an ensemble technique proposed by Freund (1995) to build a classifier ensemble incrementally, by adding one classifier at a time. The training set used for each member of the ensemble is chosen based on the performance of the earlier classifiers in the ensemble. Figure 9 states the boosting algorithm (Witten and Frank 2005):

Boosting technique requires three parameters: (1) classifier: the base classifier to apply boosting on; (2) resampling/reweighting: which approach is used (resampling or reweighting); and (3) numIterations: number of instances of the base classifiers to be created, i.e., the ensemble size. In

```
model generation
Assign equal weight to each training instance.
    For each of t iterations:
    Apply learning algorithm to weighted dataset and
    store resulting model.
    Compute error e of model on weighted dataset and
    store error.
    If e equal to zero, or e greater or equal to 0.5:
        Terminate model generation.
    For each instance in dataset:
        If instance classified correctly by model:
            Multiply weight of instance by e / (1 – e).
  Normalize weight of all instances.
classification
Assign weight of zero to all classes.
For each of the t (or less) models:
  Add –log(e / (1 – e)) to weight of class predicted by model.
Return class with highest weight.
```

Fig. 9 Boosting ensemble

this study, we prefer to use the term ensemble size for clarity purpose.

There are a family of boosting algorithms (Freund and Schapire 1996). In this study, we used AdaBoost algorithm proposed by Freund and Schapire (1995). AdaBoost was proposed to improve the performance of other learning algorithms. There are two approaches implemented in AdaBoost: resampling and reweighting. In resampling, the fixed training sample size and training examples are resampled according to a probability distribution used in each iteration. In reweighting, all training examples, with weights assigned to each example, are used in each iteration to train the base classifier. In this study, we used the resampling approach, because it has been reported to yield better accuracy (Banfield et al. 2007; Zhang et al. 2008).

5.2 Base learners

Four base learners (classifiers) were used for the bagging and boosting ensemble methods. Three of them, which are MLP, RBF and SVM are described in Sect. 4.2. The fourth model is decision tree (DT), which is created typically using C4.5 algorithm developed by Quinlan (1993). C4.5 creates decision tree whose structure consists of leaves using a top-down, divide-and-conquer approach. We used C4.5 algorithm to generate decision tree through WEKA machine learning toolkit (Misra 2005), and its parameters were initialized using the default values as follows. Confidence factor used for pruning was 25 %. Minimum number of instance per leaf was 2.

5.3 Datasets

We used two recent object-oriented class change-proneness datasets collected by Elish and Al-Khiaty (2013): VSSPLUGIN and PeerSim datasets. The VSSPLUGIN dataset contains class-level metrics data collected from the 36 classes of the first release of the system, whereas the PeerSim dataset contains the same metrics collected from the 60 classes of the first release of the system. Both systems were implemented in Java.

Table 5 Independent variables in the datasets for empirical study II

Metric	Description
WMC	Count of methods implemented within a class
DIT	Level for a class within its class hierarchy
NOC	Number of immediate subclasses of a class
RFC	Count of methods implemented within a class plus the number of methods accessible to an object class due to inheritance
LCOM	The average percentage of methods in a class using each data field in the class subtracted from 100 %
CBO	The number of classes to which a class is coupled

Both datasets consist of seven class-level metrics: six independent variables and one dependent variable. The independent (input) variables are the Chidambar and Kemerer metrics (Chidamber and Kemerer 1994): WMC, DIT, NOC, RFC, LCOM, and CBO. Table 5 provides brief description for each metric. The dependent (output) variable is a Boolean variable, which indicates whether or not the corresponding class has changed during the software evolution.

5.4 Performance evaluation measures

Two popular and common performance metrics were used to assess and compare the prediction models. The first one is correct classification rate (CCR), which is the ratio of cases that were correctly predicted to the total number of cases. It is calculated as follows:

$$\mathrm{CCR} = \frac{\mathrm{TP} + \mathrm{TN}}{N},$$

where TP is the number of true positive cases, TN is the number of true negative cases, and N in the total number of cases. The second metric is area under curve (AUC), which is calculated based on the receiver operating characteristic (ROC) curve that plots the true positive rate versus the false positive rate at various threshold settings. It is calculated as follows (Bradley 1997):

$$\mathrm{AUC} = \sum_i \left\{ (1 - \beta_i \cdot \Delta\alpha) + \frac{1}{2}[\Delta(1-\beta) \cdot \Delta\alpha] \right\}$$

$$1 - \beta = \mathrm{TruePositiveRate} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

$$\alpha = \mathrm{FalsePositiveRate} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}},$$

Table 6 Classification performance results: VSSPLUGIN dataset

Model	Individual		Bagging ensemble		Boosting ensemble	
	CCR	AUC	CCR	AUC	CCR	AUC
MLP	55.00	0.54	66.67	0.60	61.11	0.50
RBF	47.22	0.44	63.89	0.55	55.56	0.53
SVM	63.89	0.48	61.11	0.41	58.33	0.56
DT	**66.67**	**0.55**	**72.22**	**0.71**	**63.89**	**0.60**

Bold values in this table represent best prediction performance values

Table 7 Classification performance results: PeerSim dataset

Model	Individual		Bagging ensemble		Boosting ensemble	
	CCR	AUC	CCR	AUC	CCR	AUC
MLP	55.00	0.54	58.33	0.60	60.00	0.50
RBF	66.67	**0.59**	61.67	0.61	**66.67**	**0.64**
SVM	**68.33**	0.55	**68.33**	0.59	60.00	0.58
DT	55.00	0.57	65.00	**0.65**	63.33	0.60

Bold values in this table represent best prediction performance values

where FP is the number of false positive cases, and FN is the number of false negative cases. The higher the AUC, the better the model.

5.5 Results and analysis

A leave-one-out cross-validation procedure was used in this experiment. In this procedure, one observation is removed from the dataset, and then each model is built with the remaining $n - 1$ observations and evaluated in predicting the value of the observation that was removed. The process is repeated each time removing a different observation. It was observed that when ensemble size was set to 25 and more, bagging and boosting did not produce significant different results over smaller ensemble sizes, i.e., most results are stable (Aljamaan and Elish 2009). The size of ensembles was therefore set to 25 in this study.

Tables 6 and 7 show the CCR and AUC values that were achieved by each of the four individual classifiers and their bagging and boosting ensembles when applied to VSSPLUGIN and PeerSim datasets, respectively. By comparing the individual classifiers, BT model was the best performing classifier on VSSPLUGIN dataset while both RBF and SVM models achieved the best competitive accuracy on PeerSim dataset.

Figure 10 shows the impact of the bagging and boosting ensemble methods on the classification accuracy of the individual models when applied to VSSPLUGIN dataset. Bagging ensembles increased the accuracy for MLP, RBF, and DT, while there was a minor decrease in accuracy for SVM. Boosting ensembles had an increase of accuracy for MLP and RBF, while it decreased the accuracy for SVM and DT.

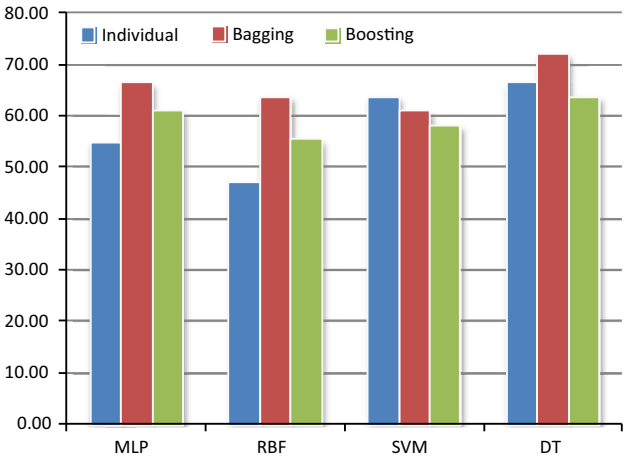

Fig. 10 CCR for each model: VSSPLUGIN dataset

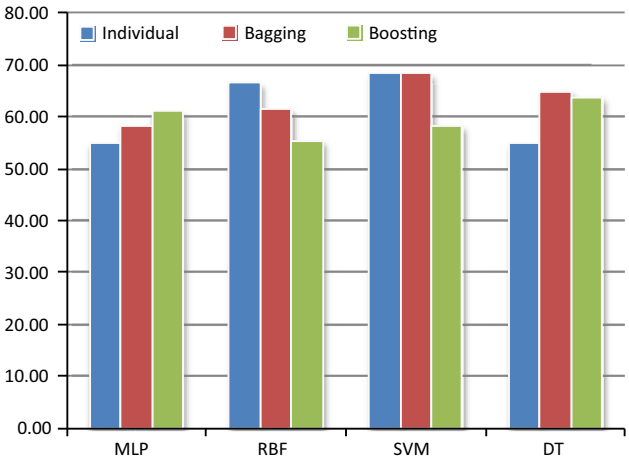

Fig. 11 CCR for each model: PeerSim dataset

It can be also observed that bagging ensembles resulted in better accuracy than the corresponding boosting ensembles.

Figure 11 shows the impact of the bagging and boosting ensemble methods on the classification accuracy of the individual models when applied to PeerSim dataset. Bagging ensembles increased the accuracy or at least produced the same accuracy for MLP, SVM and DT, while there was a minor decrease in accuracy for RBF. Boosting ensembles had an increase of accuracy for MLP and DT, while it decreased the accuracy in RBF and SVM.

The results from both datasets suggest that bagging and boosting ensemble methods have positive impact on the classification accuracy when MLP model is used as base classifier, but they have negative impact when SVM model is used as base classifier. In case of RBF model, the impact was positive in one dataset and negative in the other dataset. In case of DT model, the impact of boosting ensembles was positive in one dataset and negative in the other dataset, but the impact of bagging ensembles was positive on both datasets.

```
Choose dataset with N observations
Choose M individual prediction models
Set K for K folds cross validation
For each k ∈ K fold
    For each m ∈ M model
        Apply model m on the training set for fold(k)
        Calculate training error E, based on a certain criterion
        Store error E
    End for
    Select the best model b ∈ M, based on training error E
    For each n ∈ N observation in the testing set for fold(k)
        EnsembleOutput = the result of applying model b on observation n
    End for
End for
```

Fig. 12 Best-in-training ensemble

6 Empirical study III

The goal of this empirical study is to evaluate and compare three heterogeneous ensemble methods in predicting class change proneness. Two of the ensembles have linear combination rules, whereas the third one has non-linear combination rules.

6.1 Ensemble methods

6.1.1 Best-in-training ensemble

The idea behind this ensemble is to take across the dataset partitions, the best model (base classifier) in training based upon a certain criterion in that partition. In our case, the criterion is the classification accuracy. Figure 12 provides a formal description of the ensemble.

6.1.2 Majority voting ensemble

For majority voting, we take the output of each base learner (classifier) for the test set and the ensemble output is the category which is predicted by majority of the base learners. Figure 13 provides a formal description of the ensemble.

6.1.3 Non-linear ensemble

In the non-linear ensemble, we train the model by training a classifier whose input is the prediction outputs (for the training set) of base learners and the classifier uses them to learn the actual output (for the training set). Finally, the trained ensemble uses the test set output of base learners to make a final prediction on the test set. Decision tree forest (DTF) was used as a classifier for the non-linear ensemble. DTF is an implementation of random forest developed by Breiman (2001). It is a collection of decision trees where the prediction of each tree is combined to make an overall prediction. DTF has high prediction/classification accuracy and is highly resistant to over fitting. We used DTREG tool for implementation without any parameter optimization. Figure 14 provides a formal description of the non-linear ensemble.

6.2 Base learners

Five base learners (classifiers) were used for the ensemble methods in this empirical study. Two of them, which are MLP and SVM are described in Sect. 4.2. The other three are described next.

6.2.1 Logistic regression

Logistic regression is a well-known and widely used regression model. It is used when the target variable is categorical (for classification task) as opposed to continuous (for prediction task). We used DTREG tool implementation of logistic regression.

6.2.2 K-Means

K-Means is one of the oldest models and was developed by Hartigan and Wong (1979). The core idea of the model is the clustering algorithm where the algorithm clusters the data points and assign same cluster IDs to the clusters belonging to same class (*K* being the number of classes). A target is assigned the class whose cluster center is nearest to the target.

Fig. 13 Majority voting ensemble

```
Choose dataset with N observations
Choose M individual prediction models
Set K for K folds cross validation
For each k ∈ K fold
    For each m ∈ M model
        Apply model m on the training set for fold(k)
        Do the prediction on the test set for fold(k)
    End for
    For each n ∈ N observation in the testing set for fold(k)
        Count the number of models predicting a category
        EnsembleOutput = the category which has the maximum count for the observation n
    End for
End for
```

Fig. 14 Non-linear ensemble

```
Choose dataset with N observations
Choose M individual prediction models
Set K for K folds cross validation
For each k ∈ K fold
    For each m ∈ M model
        Train model m on the training set for fold(k)
          For each n ∈ N observation in the training set for fold(k)
              Predict the category of observation n by applying the trained model m
        End for
    End for
    Train the ensemble with inputs as category predicted for the observation n ∈ N in the training
      set for fold(k) by each model m ∈ M
    For each n ∈ N observation in the testing set for fold(k)
        EnsembleOutput = the prediction made by the trained ensemble
    End for
End for
```

6.2.3 Gene expression programming (GEP)

Gene Expression Programming was developed by Ferreira (2001). It is a special type of genetic algorithm where the individual chromosomes are initially encoded as linear strings but later gets transformed into non-linear representation with variable sizes and shape. It performs symbolic regression (where the form of the function to fit is not specified beforehand) to fit the data. We used DTREG tool for implementation of GEP without any parameter optimization.

6.3 Datasets

In this study, we used the same two datasets (VSSPLUGIN and PeerSim) that we used for the second empirical study, which are described in Sect. 5.3.

6.4 Performance evaluation measures

Correct classification rate (CCR) and area under curve (AUC) were used as performance evaluation measures. They were already described in Sect. 5.4.

6.5 Results and analysis

We partition each dataset into four disjoint randomly selected test sets such that each test set contains 25 % of the data and the remaining 75 % of the data was assignment as the training set for that particular test set. Each training set was used to training the base classifiers and evaluation was carried out on the test set associated with that particular training set. Performance evaluation measures were recorded for each experiment. Finally, we report the overall results by aggregating the performance over the four set of experiments carried out on a dataset.

Table 8 reports the classification performance results achieved by the individual classifiers as well as the three ensemble methods. Among the individual classifiers, Genetic Programming performed best for VSSPLUGIN dataset whereas SVM classifier performed best for PeerSim dataset. Among the ensemble methods, the non-linear ensemble performed best for VSSPLUGIN dataset, but the best-in-training ensemble performed best for PeerSim dataset. Moreover, the performance of the non-linear ensemble was competitive for PeerSim dataset.

Table 8 Classification performance results

Classifier	VSSPLUGIN dataset		PeerSim dataset	
	CCR	AUC	CCR	AUC
SVM	66.67	0.36	63.33	0.64
MLP	50.00	0.45	60.00	0.52
Logistic regression	55.56	0.54	55.00	0.50
Genetic programming	**77.78**	**0.71**	60.00	0.55
K-Means	58.33	0.50	58.33	0.55
Ensemble (best in training)	61.11	0.58	**68.33**	**0.67**
Ensemble (majority voting)	63.89		60.00	
Ensemble (nonlinear-DTF)	66.67	0.59	**68.33**	0.63

Bold values in this table represent best prediction performance values

Furthermore, in case of VSSPLUGIN dataset, Genetic Programming classifier was the best and outperformed all ensembles. However, the non-linear ensemble outperformed all other individual classifiers. In case of PeerSim dataset, the best-in-training ensemble and also the non-linear ensemble outperformed all the individual classifiers. These results are also supported by Figs. 15 and 16, which show the ROC curves for all classifiers. The top most curves represent the best performing classifier.

These results may be explained by the fact that some classifiers train very well on the train set, but do not perform well for the test set (the problem of over-fitting) and due to this the effectiveness of the ensemble is reduced because they essentially rely on the training performance of individual classifiers. One way to address this issue as a future work is to select classifiers that are not too prone to over-fitting. Another way to address this problem is to have a separate

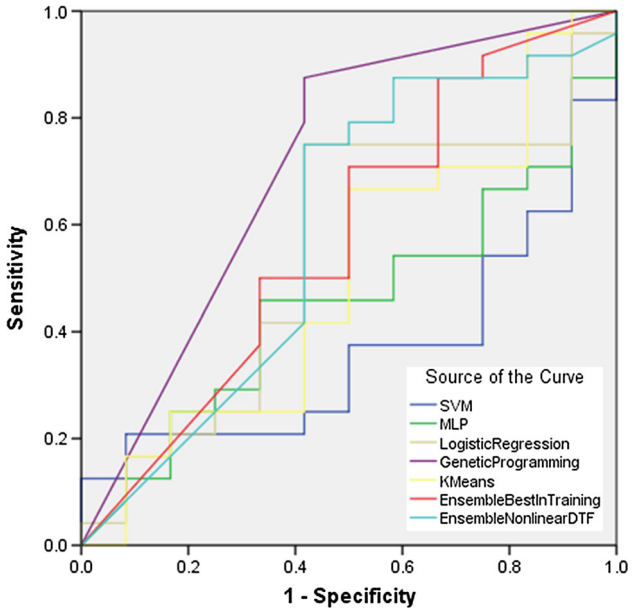

Fig. 15 ROC curves for the classifiers: VSSPLUGIN dataset

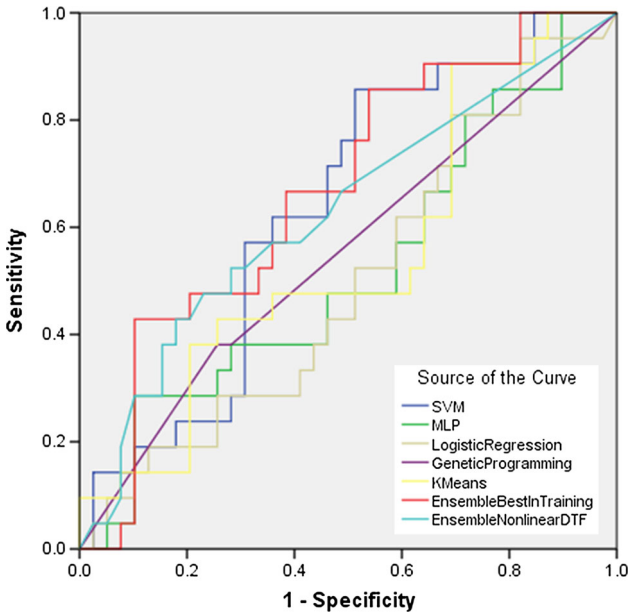

Fig. 16 ROC curves for the classifiers: PeerSim dataset

validation set (taken out from training set) and use the classifiers performance on validation set instead of the training set for the ensembles.

7 Conclusion

This paper has reported a comprehensive study of ensemble models for predicting software maintainability. We conducted three empirical studies on predicting software maintainability using ensemble methods. The first study aimed to evaluate and compare three heterogeneous ensemble methods with different linear combination rules in predicting software maintenance effort. Several interesting findings were obtained from that study. The results support the indication that the performance of the individual prediction models may vary from dataset to dataset; the MLP model was the best in one dataset while the SVM model was the best in the other dataset. The BT ensemble method outperformed all other ensemble and individual models in both datasets. Moreover, among the ensemble methods, the BT method was the best followed by the WT method and then the AVG method. We observed that the ensemble methods generally achieved more accurate or at least competitive prediction accuracy compared to individual models.

The purpose of the second empirical study was to evaluate and compare two homogeneous ensemble methods in predicting class change proneness. The results from that study suggest that bagging and boosting ensemble methods have positive impact on the classification accuracy when MLP model is used as base classifier, but they have negative impact when SVM model is used as base classifier. In case of RBF model, the impact was positive in one dataset and negative in the other dataset. In case of DT model, the impact of boosting ensembles was positive in one dataset and negative in the other dataset, but the impact of bagging ensembles was positive on both datasets.

The third empirical study was conducted to evaluate and compare three heterogeneous ensemble methods in predicting class change proneness. Linear as well as non-linear combination rules were used for the ensembles. From that study we observed that among the individual classifiers, genetic programming performed best for one dataset, whereas SVM classifier performed best for the other dataset. Among the ensemble methods, the non-linear ensemble performed best for one dataset, but the best-in-training ensemble performed best for the other dataset. Moreover, the performance of the non-linear ensemble was also competitive for the other dataset.

This paper contributes novel empirical evidences on the effectiveness of ensemble methods in predicting software maintainability. Overall empirical evidence obtained from the three empirical studies confirms that some ensemble methods provide more accurate or at least competitive prediction accuracy compared to individual models across datasets, and thus they are more reliable. There are possible directions for future work, which include: investigating more nonlinear ensemble methods and comparing their performance with linear ensemble methods; considering other ensemble constituent models; applying ensemble methods to other software engineering prediction problems such as fault prediction. Both theoretical (Hansen and Salamon 1990; Krogh and Vedelsby 1995) and empirical research stud-

ies (Hashem et al. 1994; Opitz and Shavlik 1996a, b) have demonstrated that a good ensemble is one where the individual prediction models in the ensemble are both accurate and make their errors on different parts of the input space. Therefore, one important direction of future work is to investigate different sets of ensemble constituent models.

Acknowledgments The authors wish to acknowledge King Fahd University of Petroleum and Minerals (KFUPM) for utilizing the various facilities in carrying out this research.

References

Ahmed M, Al-Jamimi H (2013) Machine learning approaches for predicting software maintainability: a fuzzy-based transparent model. IET Softw 7(6):317–326

Al-Dallal J (2013) Object-oriented class maintainability prediction using internal quality attributes. Inf Softw Technol 55:2028–2048

Aljamaan H, Elish M (2009) An empirical study of bagging and boosting ensembles for identifying faulty classes in object-oriented software. In: IEEE symposium on computational intelligence and data mining, pp 187–194

Aljamaan H, Elish M, Ahmad I (2013) An ensemble of computational intelligence models for software maintenance effort prediction. In: 12th International work conference on artificial neural networks (IWANN 2013), part I, LNCS 7902, pp 592–603

Bandi R, Vaishnavi V, Turk D (2003) Predicting maintenance performance using object-oriented design complexity metrics. IEEE Trans Softw Eng 29(1):77–87

Banfield R, Hall L, Bowyer K, Kegelmeyer W (2007) A comparison of decision tree ensemble creation techniques. IEEE Trans Pattern Anal Mach Intell 29(1):173–180

Bittencourt V, Abreu M, Souto M, Canuto A (2005) An empirical comparison of individual machine learning techniques and ensemble approaches in protein structural class prediction. In: International joint conference on neural networks, pp 527–531

Bradley A (1997) The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recognit 30(7):1145–1159

Braga P, Oliveira A, Ribeiro G, Meira S (2007) Bagging predictors for estimation of software project effort. In: International joint conference on neural networks, pp 1595–1600

Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Briand L, Bunse C, Daly J (2001) A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs. IEEE Trans Softw Eng 27(6):513–530

Chidamber S, Kemerer C (1994) A metrics suite for object oriented design. IEEE Trans Softw Eng 20(6):476–493

Conte S, Dunsmore H, Shen V (1986) Software engineering metrics and models. Benjamin/Cummings, Menlo Park

De Lucia A, Pompella E, Stefanucci S (2005) Assessing effort estimation models for corrective maintenance through empirical studies. Inf Softw Technol 47(1):3–15

DTREG, Predictive modeling software by Phillip Sherrod. http://www.dtreg.com. Accessed 5 Jan 2014

Elish M, Al-Khiaty M (2013) A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software. J Softw Evol Process 25(5):407–437

Elish M, Elish K (2009) Application of TreeNet in predicting object-oriented software maintainability: a comparative study. In: 13th European conference on software maintenance and reengineering (CSMR '09), pp 69–78

Elish M, Helmy T, Hussain M (2013) Empirical study of homogeneous and heterogeneous ensemble models for software development effort estimation. Math Probl Eng 2013:1–21. doi:10.1155/2013/312067

Ferreira C (2001) Gene expression programming: a new adaptive algorithm for solving problems. Complex Syst 13(2):87–129

Fioravanti F, Nesi P (2001) Estimation and prediction metrics for adaptive maintenance effort of object-oriented systems. IEEE Trans Softw Eng 27(12):1062–1084

Freund Y (1995) Boosting a weak learning algorithm by majority. Inf Comput 121(2):256–285

Freund Y, Schapire RE (1995) A decision-theoretic generalization of on-line learning and an application to boosting. In: European conference on computational learning theory, pp 23–37

Freund Y, Schapire RE (1996) Experiments with a new boosting algorithm. In: Thirteenth international conference on machine learning, Italy, pp 148–156

Gutta S, Wechsler H (1996) Face recognition using hybrid classifier systems. In: IEEE international conference on neural networks, pp 1017–1022

Hansen L, Salamon P (1990) Neural network ensembles. IEEE Trans Pattern Anal Mach Intell 12(10):993–1001

Hartigan J, Wong M (1979) Algorithm AS 136: a K-means clustering algorithm. J R Stat Soc Ser C (Appl Stat) 28(1):100–108

Hashem S, Schmeiser B, Yih Y (1994) Optimal linear combinations of neural networks. Neural Netw 3:1507–1512

Haykin S (1999) Neural networks: a comprehensive foundation. Prentice Hall, New Jersey

Huang FJ, Zhou Z, Zhang H-J, Chen T (2000) Pose invariant face recognition. In: Proceedings of the 4th IEEE international conference on automatic face and gesture recognition, France, pp 245–250

Khoshgoftaar T, Geleyn E, Nguyen L (2003) Empirical case studies of combining software quality classification models. In: Third international conference on quality software, p 40

Kiran N, Ravi V (2008) Software reliability prediction by soft computing techniques. J Syst Softw 81(4):576–583

Kohavi R (1995) A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of the 14th international joint conference on artificial intelligence (IJCAI), pp 1137–1143

Koten C, Gray A (2006) An application of Bayesian network for predicting object-oriented software maintainability. Inf Softw Technol 48(1):59–67

Krogh A, Vedelsby J (1995) Neural network ensembles, cross validation, and active learning. Adv Neural Inf Process Syst 7:231–238

Li W, Henry S (1993) Object-oriented metrics that predict maintainability. J Syst Softw 23(2):111–122

Mao J (1998) A case study on bagging, boosting and basic ensembles of neural networks for OCR. In: Proceedings of IEEE international joint conference on neural networks, pp 1828–1833

Misra S (2005) Modeling design/coding factors that drive maintainability of software systems. Softw Qual Control 13(3):297–320

Opitz D, Shavlik J (1996) Actively searching for an effective neural-network ensemble. Connect Sci 8(3/4):337–353

Opitz D, Shavlik J (1996) Generating accurate and diverse members of a neural-network ensemble. Adv Neural Inf Process Syst 8:535–541

Optiz D, Maclin R (1999) Popular ensemble methods: an empirical study. J Artif Intell Res 11:169–198

Poggio T, Girosi F (1990) Networks for approximation and learning. Proc IEEE 78(9):1481–1497

Quinlan J (1993) C4.5: programs for machine learning. Morgan Kaufmann Publishers, San Francisco

Quinlan R (1992) Learning with continuous classes. In: 5th Australian joint conference on artificial intelligence, Singapore, pp 343–348

Shevade S, Keerthi S, Bhattacharyya C, Murthy K (2000) Improvements to the SMO algorithm for SVM regression. IEEE Trans Neural Netw 11(5):1188–1193

Shimshoni Y, Intrator N (1998) Classification of seismic signals by integrating ensembles of neural networks. IEEE Trans Signal Process 46(5):1194–1201

Sollich P (1996) Learning with ensembles: how over-fitting can be useful. Adv Neural Inf Process Syst 8:190–196

Thwin M, Quah T (2005) Application of neural networks for software quality prediction using object-oriented metrics. J Syst Softw 76(2):147–156

Vapnik V (1995) The nature of statistical learning theory. Springer, New York

Wang Y, Witten IH (1997) Induction of model trees for predicting continuous classes. In: Poster papers of the 9th European conference on machine learning

Witten I, Frank E (2005) Data mining: practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, San Francisco

Zhang C, Zhang J, Zhang G (2008) An efficient modified boosting method for solving classification problems. J Comput Appl Math 214:381–392

Zheng J (2009) Predicting software reliability with neural network ensembles. Expert Syst App 36(2):2116–2122

Zhou Y, Leung H (2007) Predicting object-oriented software maintainability using multivariate adaptive regression splines. J Syst Softw 80(8):1349–1361