Toward Accurate Software Effort Prediction Using Multiple Classifier Systems

Bhekisipho Twala and June Verner

Abstract Averaging is a standard technique in applied machine learning for combining multiple classifiers to achieve greater accuracy. Such accuracy could be useful in software effort estimation which is an important part of software process management. To investigate the use of ensemble multiple classifiers learning in terms of predicting software effort. The use of ensemble multiple classier combination is demonstrated and evaluated against individual classifiers using 10 industrial datasets in terms of the smoothed error rate. Experimental results show that multiple classifier combination can improve software effort prediction with boosting, bagging and feature selection achieving higher accuracy rates. Accordingly, good performance is consistently derived from static parallel systems while dynamic classifier selection systems exhibit poor accuracy rates. Most of the base classifiers are highly competitive with each other. The success of each method appears to depend on the underlying characteristics of each of the ten industrial datasets.

Keywords Multiple classifiers · Machine learning · Software effort · Predictive accuracy

B. Twala (✉)
Department of Electrical and Electronic Engineering Science, University of Johannesburg, P.O. Box 524, Auckland Park, Johannesburg 2006, South Africa
e-mail: btwala@uj.ac.za

J. Verner
Computer Science and Engineering, University of New South Wales UNSW, Sydney, NSW 2052, Australia
e-mail: june.verner@gmail.com

W. Pedrycz et al. (eds.), *Computational Intelligence and Quantitative Software Engineering*, Studies in Computational Intelligence 617,
DOI 10.1007/978-3-319-25964-2_7

1 Introduction

Software effort estimation is an important area for software development. If the software development effort is under estimated tight time schedules will result leading to the possibility of inadequate testing and poor quality software. In contrast, if the software development effort is overestimated over allocation of man power and resources may result. Thus, accurate software effort estimation is an important part of the software management process in terms of productivity and quality. Many software effort estimation models have been proposed [2, 6, 18] and unbiased effort prediction is an important contributor to effective software project management. It is also generally accepted that the highest accuracy results that a classifier system can achieve depend on the quality of data and the appropriate selection of a learning algorithm for the data [7, 27, 35]. One of the central tasks of classifiers is determining whether a particular instance belongs to a specified class, given a description of that instance. The wealth and complexity of industrial data lends itself well to the application of classifiers for prediction or classification of software projects according to factors that influence software effort rates.

Machine learning (ML) deals with the problem of building computer programs that improve their performance at some tasks through experience and has proven to be of great value in a variety of applications including software development effort estimation (the process of predicting the effort required to develop a software system). In recent years, several machine learning approaches have been applied in software systems development and deployment in order to establish more sound predictive models for software quality [41]. Averaging is a standard technique in applied and theoretical ML for combining multiple classifiers in order to achieve great accuracy. In fact, in recent years, there has been an explosion of papers in the ML and statistical pattern recognition (SPR) communities discussing how to combine models or model predictions in order to improve predictive accuracy.

Research in both ML and SPR communities has shown that combining (ensemble) individual classifiers is an effective technique for improving predictive accuracy. In other words, developing an effective decision combination function is critical to the success of a multiple classifier system (MCS). Such a function should take advantage of the strengths of individual classifiers while avoiding their weaknesses, and improve classification correctness. The performance of multiple classifier systems not only depends on the power of the individual classifiers in the system but is also influenced by the independence between classifiers.

It has long been recognized that software effort estimation is a key consideration for good software cost estimation. However, effort prediction in terms of using multiple classifier (machine) learning or ensembles has attracted some attention in areas such as pattern recognition [24, 25], information security [8], credit risk [37], engineering [36], and so on, but yet received little attention in the software engineering community. Work by Wettschereck [26] provides a solid start to the use of multiple classifier learning by proposing a hybrid strategy that combines the nearest-hyper-rectangle and *k*-nearest neighbour algorithms in terms of improving classification accuracy.

Follow up research work by Braga et al. [3] and Kultur et al. [23] shows how bagging may improve software effort predictive accuracy in comparison with the use of a single classifier; although the results from both studies are inconclusive. Khoshgoftaar et al. [19] propose a hybrid software quality prediction model that combines rule-based and case-based learning which outperforms the best individual rule-based model. Kocaguneli et al. [21] suggest that ensembles are not able to improve predictive accuracy of single learning classifiers, contradicting the findings of Khoshgoftaar et al.'s research. However, the Kocaguneli et al.'s [21] research work lacks any statistical justification. In their most recent research work, Kocaguneli et al. [22] show ensemble methods significantly outperforming single classifiers with error rates significantly less than are shown by their earlier work. The ranking of the best ensemble methods were also shown to be stable by Kocaguneli et al. [22]. Twala and Cartwright [37] showed that the ensemble approach can also be used to improve software effort predictive accuracy in the presence of missing values.

The performance of several multiple classifier systems are evaluated in terms of their ability to predict software effort using 10 industrial datasets in this research. Initially single classifiers are constructed using five base methods for classifier construction. These are then used to provide benchmarks against which various multiple classifier systems are assessed. To the best of our knowledge this is the first study where such a combination of methods in terms of classifier learning and ensemble learning approaches have been used to create different ensemble multiple classifier systems across ten industrial datasets. A classifier ensemble is generated by training multiple learners for the same task and then combining their predictions as demonstrated in Sect. 3 of the paper. There are different ways in which ensembles can be generated, and the resulting output combined for the classification of new instances. Popular approaches for creating ensembles include changing the instances used for training through techniques such as bagging [4], boosting [13], stacked generalization or stacking [40], changing the features used in training [15], and introducing randomness in the classifier itself [10].

Bagging is a combination of bootstrapping and averaging used to decrease the variance part of prediction errors; boosting is one of the most well-known techniques for solving classification problems; stacking combines various machine learning methods using a stacking generalization technique; randomization is based on bagging models built using a random tree strategy in which classification trees are grown on a random subset of descriptors; feature selection aims for an optimal set as a whole rather than a combination of stand-alone high performance attributes.

The rest of this paper is organised as follows. Section 2 briefly provides details of the five classifiers used in this paper; this is followed by a description of different types of multiple classifier system architectures. Section 4 empirically explores the robustness and accuracy of five multiple classifier systems when used with ten industrial datasets in terms of the smoothed error rate. This section also presents empirical results from the application of the ensemble procedures. Section 5 provides our conclusions and future research directions.

2 Classifiers

In supervised learning, for multivariate data, a classification function $y = f(x)$ from training examples of the form $\{(x_1,y_1),\ldots,(x_m,y_m)\}$, predicts one (or more) output attribute(s) or dependent variable(s) given the values of the input attributes of the form $(x, f(x))$. The x_i values are vectors of the form $\{x_{i1},\ldots,x_{in}\}$ whose components can be numerically ordered, nominal or categorical, or ordinal. The y values are drawn from a discrete set of classes $\{1, \ldots, K\}$ in the case of *classification*. Depending on the usage, the prediction can be "definite" or probabilistic over possible values. Given a set of training examples and any given prior probabilities and misclassification costs, a learning algorithm outputs a *classifier*. The classifier is an hypothesis about the true classification function that is learned from, or fitted to, *training data*. The classifier is then tested on *test data*.

The five base methods for classifier construction considered in our study are presented below.

2.1 Logistic Discrimination

Logistic discrimination analysis (LgDA) Cox [9] is related to logistic regression. The dependent variable can only take the values 0 and 1, say, given two classes. This technique is partially parametric, as the probability density functions for the classes are not modelled but rather the ratios between them are used as described below.

Let $y \in \{0, 1\}$ be the dependent or response variable and let $x = x_{i1}, \ldots, x_{ip}$ be the predictor variables vector. A linear predictor ζ_i is given by $\beta_0 + \beta'_X$ where β_0 is the constant and β' is the vector of regression coefficients $(\beta_1, \ldots, \beta_p))$ to be estimated from the data. They are directly interpretable as log-odds ratios or in terms of $exp(\beta')$, as odds ratios.

The a posteriori class probabilities are computed by the logistic distribution. These terms are often referred to as "predictions" for the given characteristic vector x. Therefore, a new element is classified as 0 if $\pi_0 \leq c$ and as 1 if $\pi_0 > c$, where c is the cut-off point score and π_0 is the predictor. Typically, the error rate is lowest for cut-off point = 0.5 [30]. In fact, the slope of the cumulative logistic probability function has been shown to be steepest in the region where, say, $\pi_i = 0.5$. Thus, if $\pi_i > 0.5$, the unknown instance is classified as "1" and if $\pi_i \leq 0.5$, the unknown instance is classified as "0". The generalisation of the LgDA approach to the case of three or more classes is known as the multinomial logit model and the derivation is similar to that of the logistic discrimination model. The reader is referred to Hosmer and Lameshow [16] for more details.

2.2 k-Nearest Neighbour

One of the most accepted algorithms in ML is the *k*-nearest neighbour (*k*-NN), which is sometimes referred to as instance-based learning or memory-based reasoning [1]. *k*-NN methods have been used for classification tasks. The method essentially works by assigning to an unclassified sample point the classification of the nearest of a set of previously classified points. The entire training set (a set of data used to discover potentially predictive relationships in different areas of information science) is stored in the memory. Consider a set of *n* pairs is $(x_1,C_1),\ldots,(x_n,C_n)$, where x_i's take values in the metric space X upon which is defined a metric d, and the C_i's take values in the set $\{1, 2, \ldots, K\}$. A new measurement x is observed, and it is desired to estimate C by utilising the information contained in the set of correctly classified points. $x'_n \in \{x_1,\ldots,x_n\}$ is called a nearest neighbour to x if $\min d(x_i, x) = d(x'_n, x)$ $i = 1, 2,\ldots, n$. The nearest neighbour rule decides that x belongs to the category C'_n of its nearest neighbour x'_n. A mistake is made if $C'_n \neq C$. Notice that only classification of the nearest neighbour is utilised by this, simplest, nearest neighbours rule. The remaining $n - 1$ classifications C_i are ignored.

To classify a new instance, the Euclidean distance (possibly weighted) is computed between the instance and each stored training instance and the new instance is assigned the class of the nearest neighbouring instance. More generally, these *k*-nearest neighbours (*k*-NNs) are computed, and the new instance is assigned the class that is most frequent amongst the *k* neighbours. IBL's have three defining general characteristics: a similarity function (how close together the two instances are), a "typical instance" selection function (which instances to keep as examples), and a classification function (deciding how a new case relates to the learned cases). The lack of a formal framework for choosing the size of neighbourhood "k" can be problematic. To determine the distance between a pair of instances we apply the Euclidean distance metric. In our experiments, *k* is set to five. Three to five neighbours have been shown to make a good prediction [38].

2.3 Artificial Neural Network

Artificial neural networks (ANNs) use nonparametric approaches (i.e. no assumptions about the data are made). ANNs are represented by connections between a very large number of simple computing processors or elements (neurons). ANNs have been used for a variety of classification and regression problems. There are many types of ANNs, but for the purposes of this study we concentrate on single unit and multi-layer perceptrons [29] which utilizes a supervised learning technique known as backpropagation.

The backpropagation learning algorithm performs a hill-climbing search procedure on the weight space described above or a (noisy or stochastic) gradient descent numerical method whereby an error function is minimised. At each iteration, each weight is adjusted proportionally to its effect on the error. One cycles through the training set and on each example changes each weight proportionally to its effect on lowering the error. One may compute the error gradient using the chain rule and the information propagates backwards through the network through the interconnections, which accounts for the procedure's name.

There are two stages associated with the backpropagation method: training and classification. The ANN is trained by supplying it with a large number of learned (input data pattern) whose corresponding classifications (target values or desired output) are known. During training, the final sum-of-squares error over the validation data for the network is calculated. The selection of the optimum number of hidden nodes is made on the basis of this error value. The question of how to choose the structure of the network is beyond the scope of this thesis and is a current research issue in neural networks. Once the network is trained, a new object is classified by sending its attribute values to the input nodes of the network, applying the weights to those values, and computing the values of the output units or output unit activations. The assigned class is that with the largest output unit activation.

2.4 Decision Trees

Decision tree (DT) classifiers have four major objectives. According to Safavian and Landgrebe [31], these are: (1) to classify correctly as much of the training sample as possible; (2) generalise beyond the training sample so that unseen samples could be classified with as high accuracy as possible; (3) be easy to update as more training samples become available (i.e., be incremental); (4) and have as simple a structure as possible. Objective (1) is actually highly debatable as this might not be the case and to some extent conflicts with objective (2). Also, not all tree classifiers are concerned with objective (3). DTs are non-parametric and a useful means of representing the logic embodied in software routines. A DT [5, 28] takes as input a case or example described by a set of attribute values, and outputs a Boolean or multi-valued "decision". For the purpose of this paper, we shall stick to the Boolean case.

One property that sets DTs apart from all other classifiers is their invariance to monotone transformations of the predictor variables. For example, replacing any subset of the predictor variables $\{x_j\}$ by (possible different) arbitrary strictly monotone functions of them $\{x_j \leftarrow m_j(x_j)\}$, gives rise to the same tree model. Thus, there is no issue with having to experiment with different possible transformations $m_j(x_j)$ for each individual predictor x_j to try to find the best. This invariance provides immunity to the presence of extreme values ("outliers" or noise) in the predictor variable space [5].

2.5 Naïve Bayes Classifer

The NBC is perhaps the simplest and most widely studied probabilistic learning method. It learns from the training data the conditional probability of each attribute A_i given the class label C [11]. The NBC can handle an arbitrary number of independent attributes whether continuous or categorical. The strong major assumption is that all attributes A_i are independent given the value of the class C. Classification is therefore done applying Bayes rule to compute the probability of, say, C given $A_1, \ldots, A_n$ and then predicting the class with the highest posterior probability. The probability of a class value C_i given an instance $X = \{A_1, \ldots, A_n\}$ for n observations is given by:

$$\begin{aligned} p(C_i|X) &= p(X|C_i) \cdot p(C_i)/p(X) \\ &\propto p(A_1, \ldots, A_n|C_i) \cdot p(C_i) \\ &= \prod_{j=1}^{n} p(A_j|C_i) \cdot p(C_i) \end{aligned}$$

The assumption of conditional independence of a collection of random variables is very important for the above result. It would be impossible to estimate all the parameters without such an assumption. This is a fairly strong assumption that is often not applicable. However, bias in estimating probabilities may not make a difference in practice—it is the order of the probabilities, not the exact values that determine the probabilities. When the strong attribute independence assumption is violated, the performance of the NBC might be poor.

3 Multiple Classifier System Architectures

Multiple classifier systems can be classified into one of three architectural types [12]: (1) static parallel (SP); (2) multi-stage (MS); and (3) dynamic classifier selection (DCS). The outputs from each classifier are combined to deliver a final classification decision. A large number of combination functions are available. These include: voting methods (simple majority vote, weighted majority vote, the product or sum of model outputs also known as the product rule, the minimum rule, the maximum rule); rank based methods (borda-count); and probabilistic methods (Bayesian methods).

3.1 Static Parallel

SP is probably the most popular architecture and it is where two or more classifiers are developed independently in parallel [42]. The outputs from each classifier are then combined to deliver a final classification decision (where the decision is selected from a set of possible class labels). A large number of combination functions are available. These include majority voting, weighted majority voting, the product or sum of model outputs, the minimum rule, the maximum rule and Bayesian methods. In practice most combination strategies are reported to yield very similar levels of performance. However, a simple majority vote or weighted majority vote are often favoured due to the simplicity of their application and their applicability to situations where the raw outputs from each classifier may not all be interpretable in the same way.

3.2 Multi-stage

The second type of architectures is MS, where the classifiers are constructed iteratively. At each iteration (and at previous stages), the parameter estimation process is dependent upon the classification properties of the classifier(s) developed. Some MS approaches generate models that are applied in parallel using the same type of combination rules used for SP methods. For example, most forms of boosting generate a set of weak classifiers that are combined to create stronger ones [33]. Adaboost [13] is one of the most well-known algorithms that uses a MS architecture.

3.3 Dynamic Classifer Selection

For DCS, different classifiers are developed or applied to different regions within the problem domain. While one classifier may be shown to outperform all others based on global measures of performance, it may not entirely dominate all other classifiers. Weaker competitors will sometimes outperform the overall best across some regions [20]. DCS problems are normally approached from a global and local accuracy perspective [24, 25]. With a DCS global approach classifiers are constructed using all observations within the development sample. Classifier performance is then assessed over each region on interest (I am not sure what this term means) and the best classifier is chosen for each region. With DCS local, regions of interest are determined first, and then separate classifiers are developed for each region.

3.4 Classifier Ensemble

A generalised classifier ensemble algorithm is summarised in the following steps [34].

1. Partition original dataset into n training datasets, TR_1, TR_2, TR_n.
2. Construct n individual models (M_1, M_2, M_n) with the different training datasets TR_1, TR_2, …, TR_n to obtain n individual classifiers (ensemble members) generated by different algorithms, thus different.
3. Select m de-correlated classifiers from n classifiers using de-correlation maximization algorithm.
4. Using Step 3, obtain m classifier output values (misclassification error rates) of unknown instance.
5. Transform output value to reliability degrees of positive class and negative class, given the imbalance of some datasets.
6. Fuse the multiple classifiers into aggregate output in terms of majority voting.

4 Experimental Design

In order to test the suitability of multiple classifiers for predicting software effort, w performed experiments on ten industrial datasets in terms of the smoothed misclassification error rate. The smoothed error rate is used due to its variance reduction benefit. Instead of summing terms that are either zero or one as in the error-count estimator, the smoothed estimator uses a continuum of values between zero and one in the terms that are summed. The resulting estimator has a smaller variance than the error-count estimate. Each dataset, used in the experiments defines a different learning problem as summarized in Table 1. Most of the datasets are available at predictor models in software engineerinig (PROMISE) [32] with the exception of ISBSG and Company X which is not available for public use due to non-disclosure agreement.

For the simulation study, the five base methods of classifier construction were chosen. Each method utilizes a different form of parametric estimation/learning; between them they generate different models forms: linear models, density estimation, trees and networks; and they are all practically applicable within software engineering environments, with known examples of their application within the engineering management industry. To begin, single classifiers were constructed using each method. These were used to provide benchmarks against which various

Table 1 Industrial datasets problem

Dataset	Instances	Attributes		Mean development effort
		Numerical	Categorical	
Test equipment	16	17	4	236
Kemerer	18	4	2	261
Test equipment	16	17	4	379
Bank	18	2	7	1470
Test equipment	16	17	4	550
Data science institute	26	5	0	2528
Moser	32	1	1	2874
Desharnais	77	3	6	4834
Experience	95	1	5	1443
ISBSG-version 7	166	2	7	1668
China	499	16	2	3921
Company X	10,434	4	18	41,643

multiple classifier systems were assessed. To select an appropriate number of ensemble members, the de-correlation maximization method [17] was utilized. 10-fold cross validation is used for all the experiments.

For all the classifiers, the implementation in WEKA data mining software package library [39] is used, with the default parameters used for each classifier. These models were built in WEKA by performing five-fold cross validation.

Analyses of variance are used to examine the main effect and their respective interactions. This was done using a 3-way repeated measures design (where each effect was tested against its interaction with datasets). The fixed effect factors are multiple classifier methods; the ensemble learning approaches used to build the multiple classifier systems and the multiple classifier architectures. The random effect is the ten datasets. Friedman ranking test [14] was also used to check if the difference in performances between the multiple classifiers (ensembles) and the individual classifiers were significantly different in terms of the smoothed error rate.

To measure the performance of classifiers, the training set/test set methodology is employed. For each run, each dataset is split randomly into 80 % training set and 20 % testing or validation set. The performance of each classifier is then assessed on the smoothed error rate.

Although, an operational definition of accurate prediction is hard to come by predictive accuracy is mostly operationally defined as the prediction with the minimum misclassification costs (the proportion of misclassified instances). The need for minimizing costs, rather than the proportion of misclassified instances, arises when some predictions that fail are more catastrophic than others, or when some predictions that fail occur more frequently than others. Minimizing costs, however, does correspond to minimizing the proportion of misclassified instances when priors (i.e. the probability estimates drawn from the training data that one would make for each possible target value prior to knowing anything about the predictor values) are taken to be proportional to the class sizes and when misclassification costs are taken to be equal for every class [5]. This is the approach we follow in the paper.

5 Experimental Results

The results across all the ten datasets are summarized in Figs. 1, 2, 3 and 4 (and Tables 2, 3 and 4) in terms of smoothed error rate against the baseline classifiers (BASE) and their respective ensemble multiple classifiers (i.e. ENS1, ENS2, ENS3, ENS4, ENS5). The components of the ensembles are ENS1 (ANN, DT, NBC,

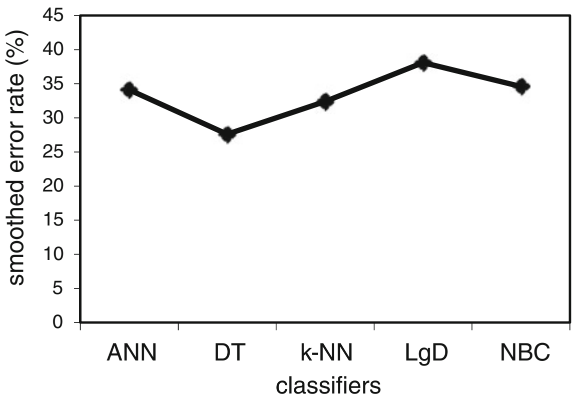

Fig. 1 Overall means for base classifiers

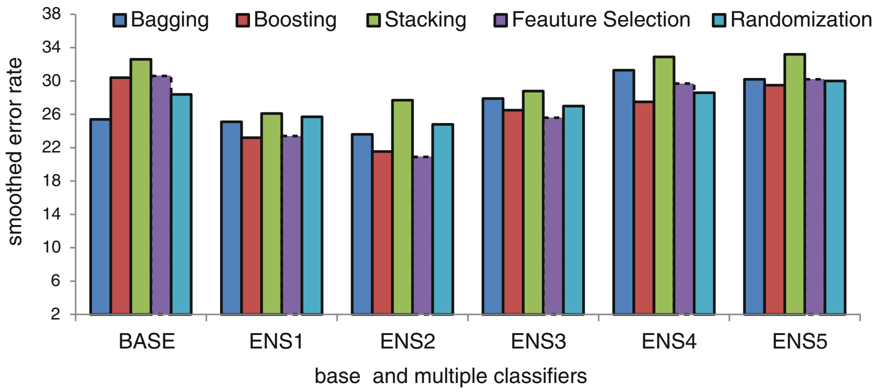

Fig. 2 Ensemble multiple classifiers (static parallel)

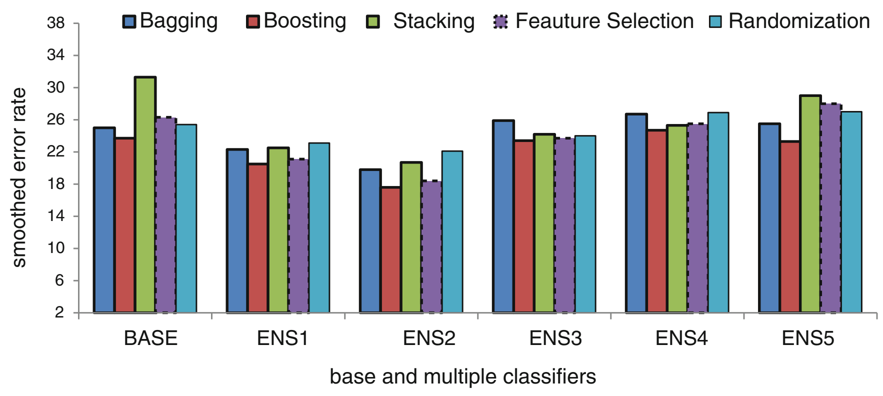

Fig. 3 Ensemble multiple classifiers (multi-stage)

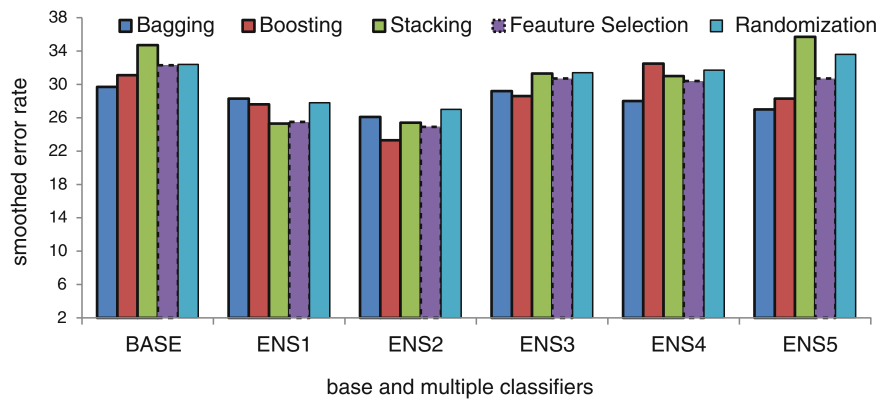

Fig. 4 Ensemble multiple classifiers (dynamic classifier selection)

Table 2 Overall means (individual classifiers and multiple clasifier systems)

Classifier/ensemble multiple classifiers	Average generalization performance (%)
ANN	34.1 ± **3.63**
DT	27.6 ± **3.99**
NBC	35.6 ± **3.27**
k-NN	32.4 ± **3.51**
LgD	38.1 ± **3.90**
ENS1	20.4 ± **2.95**
ENS2	17.9 ± **1.75**
ENS3	22.0 ± **3.23**
ENS4	23.7 ± **3.27**
ENS5	25.6 ± **3.61**

Table 3 Overall means (ensemble learning approaches)

Learning approaches	Average generalization performance (%)
Bagging	16.5 ± 2.82
Boosting	19.4 ± 3.32
Feature selection	20.3 ± 3.51
Stacking	26.9 ± 4.73
Randomization	22.1 ± 3.38

Table 4 Overall means (multiple classifier architectures)

Multiple classifier architectures	Average generalization performance (%)
Dynamic classifier selection	24.2 ± 3.64
Multi stage	21.9 ± 3.41
Static parallel	17.4 ± 2.95

k-NN, LgD) ENS2 (ANN, DT, NBC, LgD) ENS 3(ANN, DT, k-NN, LgD) ENS4 (ANN, NBC, k-NN, LgD) and ENS5 (DT, NBC, k-NN, LgD).

For the baseline classifiers, DT achieves the lowest smoothed error rate (27.6 %), followed by k-NN (32.4 %), ANN (34.1 %) and NBC (35.6 %), respectively. The worst performance for predicting software effort is by LgD with a smoothed error rate of 38.1 %. The differences in performance between the base individual classifiers are significant at the 5 % level (with the exception of ANN against NBC).

From Table 2 the multiple classifier performances (ENS1, ENS2, ENS3, ENS4, ENS5) is significantly better when compared to the individual classifiers (ANN, DT, NBC, *k*-NN, LgD) at the 95 % level of significance ($F_{statistic} = 13.141 > F_{critical\ value(10,\ 75)} = 2.056$).

When comparing the smoothed error rates in Table 3, bagging outperforms all the other sampling methods whenever it is used to construct the ensemble multiple classifier systems in software effort prediction. Bagging exhibits a smoothed error rate of 16.5 %, followed by boosting (19.4 %) and feature selection (20.3 %). However, there appears to be no significant difference in performance between boosting and feature selection at the 5 % level. Poor performance is observed when stacking is used, achieving a smoothed error rate of 26.9 %.

From Table 4, the results show that static parallel ensemble multiple classifier systems performs better in terms of predicting software effort when compared with either dynamic classifier selection or multi-stage systems. The difference in performance between the three systems is significantly different at the 5 % level.

All the static parallel systems (Fig. 2) show some potential to significantly outperform the baseline. However, stacking and bagging are the weakest, with only ensembles using ANN, LgD and DT showing major improvement over the other multiple classifier architectures.

Multi-stage systems provide statistically significant benefits over baseline models. The clear winners are feature selection and boosting, which provide large and significant improvements over the baseline and other multiple classifier systems for all methods considered, with best performance when applied to NBC (Fig. 3).

DCSs that look to segment the population into a number of sub-regions are consistently poor performers, with all the experiments yielding results that are inferior to the single best classifier. However, the performance of most static parallel and multi-stage combination strategies provide statistically significant improvements compared to DCSs (Fig. 4).

6 Conclusion

Machine learning has proved to be promising for automating software development effort. The rise of big data is most likely the largest catalyst. There are other factors as well that have made machine learning algorithms faster and easier to run which has been of great benefit to engineers. Not only does it enable the replication of results it provides some of the much needed automation capability in terms of engineering analysis and automated process planning. This covers design of software development processes in a wide range of domains. Multiple classifier learning could be involved here in areas such as learning process plans, learning error recovery strategies, learning and models for physical processes, and so on.

In this chapter we have proposed a strategy that uses machine learning techniques to improve software effort predictive accuracy. In summary, it has been found that a combination of multiple classifiers can enhance the classification and prediction accuracy of software effort to a great extent. Based on the experiments and findings on this paper, it can be concluded that multiple classifier combination can play an important role in the accurate and unbiased prediction of software effort by making full use of the abundant and detailed information in software projects and integrating the benefits of different classifiers. Thus, we can conclude that practitioners and researchers may use bagging and boosting for constructing models to predict software effort especially when measuring the quality of systems in software development. In fact, multiple classifier learning provides a new style of software development. But there are still many issues for further study, for example, developing models that would identify trends in effort revisions, selection of larger datasets, selection of member classifier, optimization of feature sets and determination of combination strategy. We intend to present our future findings in the next research journal paper.

Acknowledgments The project was sponsored by the Department of Electrical and Electronic Engineering Science at the University Of Johannesburg, South Africa. The authors would like to thank their colleagues for their valuable comments and suggestions to improve the paper.

References

1. Aha, D.W., Kibbler, D., Albert, M.K.: Instance-based learning algorithms. Mach. Learn. **6** (37), 37–66 (1991)
2. Basha, S., Dhavechelvan, P.: Analyisis of empirical software effort estimation models. Int. J. Comput. Sci. Inf. Secur. **7**(3), 68–77 (2010)
3. Braga, P.L., Oliveira, A., Ribeiro, G., Meira, S.: Bagging predictors for estimation of software project effort. In: International Joint Conference on Neural networks, Orlando, pp. 1595–1600 (2007)
4. Breiman, L.: Bagging predictors. Mach. Learn. **26**(2), 123–140 (1996)
5. Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and regression trees. Wadsworth (1984)
6. Briand, L.C., Wieczorek, I.: Resource estimation in software engineering. In: Marcinak, J. J. (ed.) Encyclopedia of Software Engineering, pp. 1160–1196. Wiley, New York (2002)
7. Brodley, C.E., Friedl, M.A.: Identifying mislabeled training data. J Artif. Intell. Res. **11**, 131–167 (1999)
8. Corona, I., Giacinto, G., Roli, F.: Intrusion detection in computer systems using multiple classifier systems. In: Okun, O., Valentini, G. (eds.) Supervised and Unsupervised Ensemble Methods and Their Applications, vol 126, pp. 91–114. Springer, Berlin (2008)
9. Cox, D.R.: Some procedures associated with the logistic qualitative response curve. In: David, F.N. (ed.) Research Papers in Statistics: Festschrift for J. Neyman, pp. 55–71. Wiley, New York (1966)
10. Dietterich, T.: An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. Mach. Learn. **40**(2), 139–158 (2000)
11. Duda, R.O., Hart, P.E.: Pattern Classification, 2nd edn. Wiley, New York (1973)
12. Finlay, S.M.: Multiple classifier architectures and their application to credit risk assessment. Working Paper 2008/012, Department of Management Science, Lancaster University, UK (2008)
13. Freund, Y., Schapire, R.: A decision theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. **55**, 119–139 (1996)
14. Friedman, M.: The use of ranks to avoid the assumption of normality implicit in the analysis of variance. J. Am. Stat. Assoc **32**(200), 675–701 (1937)
15. Ho, T.K.: Random decision forests. In: Proceedings of the 3rd International Conference on Document Analysis and Recognition, pp. 278–282 (1995)
16. Hosmer, D.W., Lameshow, S.: Applied Logistic Regression. Wiley, New York (1989)
17. Jolliffe, I.: Principal Component Analysis. Springer, Berlin (1986)
18. Jørgensen, M.: A review of studies on expert estimation of software development effort. J. Syst. Softw. **70**(1–2), 37–60 (2004)
19. Khoshgoftaar, T.M., Xiao, Y., Gao, K.: Software quality assessment using a multi-strategy classifier. Inf Sci (2010, in press)
20. Kittler, J., Hatef, M., Duin, R.P.W., Matas, J.: On combining classifiers. IEEE Trans. Pattern Anal. Mach. Intell. **20**(3), 226–239 (1998)
21. Kocaguneli, E., Bener, A., Kultur, Y.: Combining multiple learners induced on multiple datasets on software effort prediction. In: International Symposium on Software Reliability Engineering, Mysuri, India, p. 6 (2009)
22. Kocaguneli, E., Menzies, T., Keung, J.: On value of ensemble effort estimation. IEEE Trans. Softw. Eng. **38**(06), 1403–1416 (2012)
23. Kultur, Y., Turhan, B., Bener, A.: Ensemble of neural networks with associative memery (ENNA) for estimating software development costs. Knowl. Based Syst. **22**, 395–402 (2009)
24. Kuncheva, L.I.: Swithcing between selection and fusion in combining classifers: an experiment. IEEE Trans. Syst. Man Cybern. Part B Cybern. **32**(2), 146–156 (2002)
25. Kuncheva, L.: A theoretical study in six classifier fusion strategies. IEEE Trans. Pattern Anal. Mach. Intell. **24**(2), 281–286 (2002)

27. Pechenizkiy, M., Tsymbal, A., Puuronen, S., Pechenizkiy, O.: Class noise and supervised learning in medical domains: the effect of feature extraction. In: Proceedings of the 19th IEEE Symposium on Computer-Based Medical Systems, pp. 708–713 (2006)
28. Quinlan, J.R.: C4.5: Programs for Machine Learning. Los Altos, California. Morgan Kauffman Publishers INC, Burlington (1993)
29. Ripley, B.D.: Pattern Recognition and Neural Networks. Cambridge University Press, Cambridge and Wiley, New York (1992)
30. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. In: Rumelhart, D.E., McClelland, J.L. (eds.) Parallel Distributed Processing, vol. 1, pp. 318–362. MIT Press, Cambridge (1986)
31. Safavian, S.R., Landgrebe, D.: A survey of decision tree classifier methodology. IEEE Trans. Syst. Man Cybern. **21**, 660–674 (1991)
32. Sayyad, J.S., Menzies, T.J.: The PROMISE repository of software engineering databases. School of Information Technology and Engineering, University of Ottawa, Canada (2005). http://promise.site.uottawa.ca/SERepository. Accessed on 01 Dec 2014
33. Schapire, R., Freund, Y., Bartlett, P., Lee, W.: Boosting the margin: a new explanation for the effectiveness of voting methods. In: Proceedings of International Conference on Machine Learning, Morgan Kaufmann, San Francisco pp. 322–330 (1997)
34. Twala, B.: multiple classifier application to credit risk assessment. Expert Syst. Appl. **37**(4), 3236–3336 (2010)
35. Twala, B.: Effective techniques for dealing with incomplete data using decision trees. Published PhD thesis, Open University, Milton Keynes, UK (2005)
36. Twala, B.: software faults prediction using multiple classifiers. In: IEEE International Conference on Computer Research and Development (ICCRD2011), Shanghai, China, 11–13 Mar 2011
37. Twala, B., Cartwright, M.: Ensemble missing data methods in software effort prediction. Intell. Data Anal. **14**, 299–331 (2010)
38. Venables, W., Ripley, B.: Modern Applied Statistics with S-Plus. Springer, Berlin (1997)
26. Wettschereck, D.: A hybrid nearest neighbour and nearest hyperrectangle algorithm. In: Bergadano, F., Raedt, L.D. (eds.) Proceedings of European Conference on Machine Learning, pp 323–335 (1994)
39. Witten, I., Frank, E.: Data Mining Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kauffman, Burlington (2005)
40. Wolpert, D.: Stacked generalization. Neural Netw. **5**(2), 241–259 (1992)
41. Zhang, D., Tsai, J.J.P.: Advances in Machine Learning Applications in Software Engineering (2007)
42. Zhu, H., Beling, P.A., Overstreet, G.A.: A study in the combination of two consumer credit scores. J. Oper. Res. Soc. **52**, 2543–2559 (2001)

Author Biography

Bhekisipho Twala is a Professor in Artificial Intelligence and Statistical Sciences and the Head of the Electrical and Electronic Engineering Science Department at the University of Johannesburg. Before then he was a Principal Research Scientist at the Council of Science and Industrial Research (CSIR) within the Modelling and Digital Science unit. His current work involves promoting and conducting research in artificial intelligence within the electrical and electronic engineering fields and developing novel and innovative solutions to key research problems in this field. He earned his B.A. in Economics and Statistics from the University of Swaziland in 1993; his MSc in Computational Statistics from Southampton University (UK) in 1995; and his Ph.D. in Machine Learning and Statistics from the Open University (UK) in 2005. Prof. Twala was a post-doctoral researcher and Bournemouth University (UK) and later at Brunel University in the UK, mainly focussing on empirical software engineering research. His broad research interests include multivariate statistics, classification methods, knowledge discovery and reasoning with uncertainty, sensor data fusion and inference, and the interface between statistics and computing. He has particular interests in applications in finance, medicine, psychology, software engineering and most recently in robotics and has published over 70 scientific papers. Prof. Twala has a wide ranging work experience to organisations ranging from banks, through universities, to governments. He is currently an associate editor of the Intelligent Data Analysis journal, Journal of Computers, International Journal of Advanced Information Science and Technology, International Journal of Big Data Intelligence, Journal of Image and Data Fusion, Journal of Information Processing Systems, and a fellow of the Royal Statistical Society. Other professional memberships include the Association of Computing Machinery (ACM); the Chartered Institute of Transport (CIT), South Africa and a senior member of the Institute of Electrical and Electronics Engineers (IEEE).