Artif Intell Rev (2014) 42:695–703
DOI 10.1007/s10462-012-9348-9

# A comparative study of fault density prediction in aspect-oriented systems using MLP, RBF, KNN, RT, DENFIS and SVR models

**Mahmoud O. Elish**

Published online: 5 July 2012

**Abstract** This paper investigates and empirically evaluates and compares six popular computational intelligence models in the context of fault density prediction in aspect-oriented systems. These models are multi-layer perceptron (MLP), radial basis function (RBF), k-nearest neighbor (KNN), regression tree (RT), dynamic evolving neuro-fuzzy inference system (DENFIS), and support vector regression (SVR). The models were trained and tested, using leave-one-out procedure, on a dataset that consists of twelve aspect-level metrics (explanatory variables) that measure different structural properties of an aspect. It was observed that the DENFIS, SVR, and RT models were more accurate in predicting fault density compared to the MLP, RBF, and KNN models. The MLP model was the worst model, and all the other models were significantly better than it.

**Keywords** Computational intelligence · Fault density prediction · Aspect-oriented software

## 1 Introduction

Software quality is mainly compromised by faults as they may cause failures, increase maintenance cost, and decrease customer satisfaction. Effective fault prediction can therefore enable software developers to focus quality assurance activities and allocate effort and resources more effectively and efficiently. This will eventually lead to a substantial improvement in software quality.

Computational intelligence models have been applied to many prediction problems in different fields including software fault prediction, and shown promising performance. They have been applied to fault prediction in object-oriented systems, for example (Ceylan et al. 2006; Elish and Elish 2008; Guo et al. 2004; Gyimothy et al. 2005; Quah and Thwin 2003).

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

Recently, aspect-oriented software development is increasingly becoming recognized in both academia and industry as useful approach to improve the modularization of software artifacts (Filman et al. 2004). Application of computational intelligence models in fault prediction in the emerging aspect-oriented paradigm has not been investigated. The objective of this paper is to empirically evaluate and compare six popular computational intelligence models in the context of fault density prediction in aspect-oriented systems.

The rest of this paper is organized as follows. Section 2 reviews related works. Section 3 describes the computational intelligence models under investigation. Section 4 discusses the comparative study and its results. Section 5 concludes the paper.

## 2 Related work

Some research studies have investigated different statistical and computational intelligence models for fault prediction in object-oriented systems at both module-level (functions/methods) and class-level. At the module-level, studies have been mainly focused in predicting fault-proneness of modules, i.e., identifying faulty modules. For example, Guo et al. (2004) applied random forest to predict fault-proneness of modules; Khoshgoftaar et al. (2002) applied regression trees (RTs) with classification rules; Elish and Elish (2008) applied support vector machines; and Ceylan et al. (2006) applied three machine learning models, i.e., decision tree, multi-layer perceptron (MLP) and radial basis functions (RBFs) for the same purpose.

For fault prediction at the class-level, Basili et al. (1996), El-Emam et al. (2001), and Fioravanti and Nesi (2001), Briand et al. (2000) used logistic regression to identify faulty classes. Gyimothy et al. (2005) used logistic regression, decision trees and neural networks to identify faulty classes, and used linear regression to predict number of faults in a class. Yu et al. (2002) used linear regression analysis and linear discriminant analysis to predict number of faults in a class. Quah and Thwin (2003) used two neural network models to predict number of faults in a class.

Despite the availability of research studies on fault prediction in object-oriented systems, there is no similar work related to aspect-oriented systems. This research paper, however, investigates the problem of fault prediction in the relatively new and emerging aspect-oriented paradigm.

## 3 Computational intelligence models

In this study, six computational intelligence models were empirically evaluated and compared in predicting the fault density of aspects in aspect-oriented systems. These models, which are briefly described next, are popular and commonly applied models in software engineering prediction problems.

### 3.1 Multi-layer perceptron

A MLP (Bishop 1995; Duda et al. 2001) is one type of neural network that is trained using backpropagation algorithm. It consists of multiple layers of computational units that are connected in a feed-forward way. This forms a directed connection from lower units to a unit in a subsequent layer. The basic structure of MLP consists of an input layer, one or more hidden layers and one output layer. The output from a unit is used as input to units in the subsequent layer. The connection between units in subsequent layers has an associated weight which is learned using backpropagation algorithm. The hidden and output units are based on sigmoid units. A sigmoid unit calculates a linear combination of its input and then applies the sigmoid function on the result. The sigmoid function, for net input $x$ is defined as:

$$sigmoid(x) = \frac{1}{(1 + e^{-x})}$$

### 3.2 Radial basis function

The RBF Network (Buhmann and Albowitz 2003; Orr 1996) is another type of neural network. It consists of three layers: the input, hidden and output layer. It differs from MLP in the way that the hidden layer units perform calculations. In RBF Network, inputs from the input layer are mapped to each of the hidden units. The hidden units use radial functions for activation. A Gaussian function is useful in finding the activation at a hidden unit. The Gaussian function is defined as:

$$h(x) = \exp\left(\frac{-(x - c)^2}{r^2}\right)$$

where $c$ is the center of bell-shaped Gaussian and $r$ is the width.

### 3.3 K-nearest neighbor

The K-nearest neighbor (KNN) (Han and Kamber 2001; Hardle 1989; Korhonen and Kangas 1997) is a non-parametric classification technique. It can be used for both classification and regression problems. In case of classification, when given an unknown instance ($q$), a KNN classifier searches the pattern space for the k training instances that are closest to the $q$. These $k$ training instances are the $k$ “nearest neighbors” of the $q$. Then the class of $q$ can be determined by simple majority voting or by distance weighted voting from its KNN. In case of regression, we also find the $k$ instances in the dataset that are closest to $q$. Then we must use their values to predict the value of $q$. While there may be many approaches, the simplest approach is to take the mean of the neighbors’ values. So, KNN has two steps; the first is the determination of the nearest neighbors and the second is the determination of the class (in classification) or real value (in regression) using those neighbors.

### 3.4 Regression tree

The RT (Duda et al. 2001; Quinlan 1986; Wang and Witten 1997) is a regression-based decision tree. It builds a tree to predict numeric values for a given instance. For a given instance the tree is traversed from top to bottom till a leaf node is reached. At each node of the tree a decision is made to follow a particular branch based on a test condition on the attribute associated with that node. Each leaf node has a linear regression model associated with it to obtain the predicted output.

### 3.5 Dynamic evolving neuro-fuzzy inference system

Dynamic evolving neuro-fuzzy inference system (DENFIS) was introduced by Kasabov and Song (2002): “It evolves through incremental, hybrid (supervised/unsupervised), learning, and accommodates new input data, including new features, new classes, etc., through local element tuning. New fuzzy rules are created and updated during the operation of the system.

At each time moment, the output of DENFIS is calculated through a fuzzy inference system based on m-most activated fuzzy rules which are dynamically chosen from a fuzzy rule set. A set of fuzzy rules can be inserted into DENFIS before or during its learning process. Fuzzy rules can also be extracted during or after the learning process".

### 3.6 Support vector regression

Support vector machines (SVM) are kernel based learning algorithm introduced by Vapnik (1995) using the Structural Risk Minimization (SRM) principle which minimizes the generalization error, i.e., true error on unseen examples. Originally, SVM has been developed to solve classification problems. However, with the introduction of Vapnik's $\varepsilon$-insensitive loss function, SVM has been extended to solve regression estimation problems. This is known as support vector regression (SVR). The SVR maps the input data $x$ into a higher dimensional feature space by nonlinear mapping to solve a linear regression problem in this feature space. This transformation can be done using a kernel function. The most common existing kernel functions are linear kernel, polynomial kernel, Gaussian (RBF) kernel, and sigmoid (MLP) kernel. In this study, we used Gaussian RBF kernel because it yields better prediction performance (Smola 1998). The general form of a SVR function can be written as:

$$f(\mathbf{x}) = \mathbf{w} \cdot \phi(\mathbf{x}) + b$$

where $\mathbf{w} \in \Re^n$, $b \in \Re$, $\cdot$denotes the dot product in $\Re^n$, and $\phi$ is a non-linear transformation from $\Re^m$ to the high-dimensional space $\Re^n$ (i.e., n > m). Due to space constraints, detailed descriptions of SVR can be found in Burges (1998), Cristianini and Shawe-Taylor (2000), Gun (1998), Smola (1998).

## 4 Comparative study

This section discusses the comparative study that was conducted to evaluate and compare the six computational intelligence models under investigation in the context of fault density prediction in aspect-oriented systems.

### 4.1 Dataset description

The dataset used in this study was collected from an open source aspect-oriented software system called Glassbox.[1] This system, which is written in AspectJ, is an automated troubleshooting and monitoring agent for Java applications. The explanatory (input) variables in the dataset are twelve aspect-level metrics that measure different structural properties of an aspect such size, coupling and inheritance. Table 1 describes the explanatory variables and Table 2 provides their descriptive statistics. It can be observed that aspects vary in size in terms of the number of attributes, methods, advices, pointcuts, introductions, etc. Furthermore, there is a good utilization of inheritance, and aspects' afferent couplings are more than their efferent couplings.

The dependent (output) variable in the dataset is the aspect's faulty density. The number of faults per aspect was calculated by using the concurrent versions system (CVS) repository of the Glassbox system. The log data (i.e. revision history) for each aspect was obtained by using the 'log' subcommand of 'cvs' command, which was then searched for fault fixes. The

[1] http://sourceforge.net/projects/glassbox/.

**Table 1** Explanatory (input) variables in the dataset

| Metric | Description |
|---|---|
| NA | Number of attributes defined in an aspect |
| NM | Number of methods defined in an aspect |
| NAD | Number of advices in an aspect |
| NP | Number of pointcuts in an aspect |
| NI | Number of introductions in an aspect |
| LOC | Number of lines of code in an aspect |
| DIT | Level of an aspect in its inheritance hierarchy |
| NOC | Number of immediate sub-aspects of an aspect |
| ACTI | Number of classes and aspects that are affected by an aspect through introductions |
| ACTP | Number of classes and aspects that are affected by an aspect through pointcuts |
| ECTI | Number of aspects that affect an aspect through introductions |
| ECTP | Number of aspects that affect an aspect through pointcuts |

**Table 2** Descriptive statistics

| Variables | Min | Max | Avg | SD |
|---|---|---|---|---|
| NA | 0 | 9 | 2.29 | 2.79 |
| NM | 0 | 31 | 5.63 | 9.08 |
| NAD | 0 | 12 | 3.00 | 3.34 |
| NP | 0 | 29 | 4.50 | 5.96 |
| NI | 0 | 13 | 1.96 | 3.76 |
| LOC | 3 | 283 | 86.71 | 83.13 |
| DIT | 0 | 5 | 1.67 | 1.69 |
| NOC | 0 | 10 | 0.63 | 2.10 |
| ACTI | 0 | 381 | 20.96 | 80.43 |
| ACTP | 0 | 500 | 31.17 | 104.61 |
| ECTI | 0 | 28 | 2.21 | 5.52 |
| ECTP | 0 | 3 | 2.25 | 0.90 |

number of faults per aspect was then divided by the total number of lines of code in the aspect to compute its fault density.

## 4.2 Models construction

The DENFIS model for fault density prediction was built using NeuCom© Student version 0.919. The other models were built using the open source WEKA machine learning toolkit. The parameters for each of the fault density prediction models were initialized with the default settings as follows:

- *MLP*: a three layered, fully connected, feedforward MLP was used as network architecture. MLP was trained using backpropagation algorithm with 16 hidden nodes. All nodes in the network used the sigmoid transfer function. The learning rate was initially 0.3 and the momentum term was set at 0.2. The algorithm was halted when there had been no significant reduction in training error for 500 epochs with a tolerance value to convergence of 0.01.

- *RBF*: k-means clustering algorithm was used to determine the RBF center c and width $\sigma$. The value of k was set at 2.
- *KNN*: the number of observations ($k$) in the set of closest neighbor was set at 3.
- *RT*: the confidence factor used for pruning the tree was set at 25 % and the minimum number of instances per leaf was set at 2.
- *DENFIS*: the distance threshold (Dthr) was set at 0.1, m-of-n mode was set at 3, and the learning epochs value was set at 2.
- *SVR*: the regularization parameter (C) was set at 1; the kernel function used was Gaussian (RBF); and the bandwidth ($\gamma$) of the kernel function was set at 0.01.

### 4.3 Cross validation

A leave-one-out cross-validation procedure was performed to obtain a realistic estimate of the predictive power of the fault density prediction models. This procedure was used because (Myrtveit et al. 2005; Witten and Frank 2005): (i) it is closer to a real world situation than k-cross validation from a practitioner's point of view (ii) it is deterministic (no sampling is involved); and (iii) it ensures the largest possible amount of data is used for training. In this procedure, one observation is removed from the dataset, and then each prediction model is built with the remaining observations and evaluated in predicting the value of the observation that was removed. The process is repeated each time removing a different observation.

### 4.4 Evaluation criteria

In order to evaluate and compare the performance of the fault density prediction models, 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) and standard deviation magnitude of relative error (StdMRE). MMRE over a dataset of $n$ observations is calculated as follows:

$$MMRE = \frac{1}{n}\sum_{i=1}^{n} MRE_i$$

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

$$MRE_i = \frac{|x_i - \hat{x}_i|}{x_i}$$

In addition to MMRE, we used StdMRE since it is less sensitive to the extreme values compared to MMRE.

### 4.5 Results and discussions

The prediction results of the fault density models under investigation in terms of MMRE and StdMRE are reported in Table 3. The SVR model achieved the lowest (best) MMRE value. The MMRE values of the RT and DENFIS models were very competitive, but the StdMRE value of the DENFIS model was the lowest. In addition, it can be observed that the MMRE

**Table 3** Prediction results

| Model | MMRE | StdMRE |
|---|---|---|
| MLP | 5.422 | 8.190 |
| RBF | 1.693 | 1.871 |
| KNN | 1.849 | 3.140 |
| RT | 0.945 | 0.799 |
| DENFIS | 0.901 | 0.273 |
| SVR | 0.758 | 0.702 |

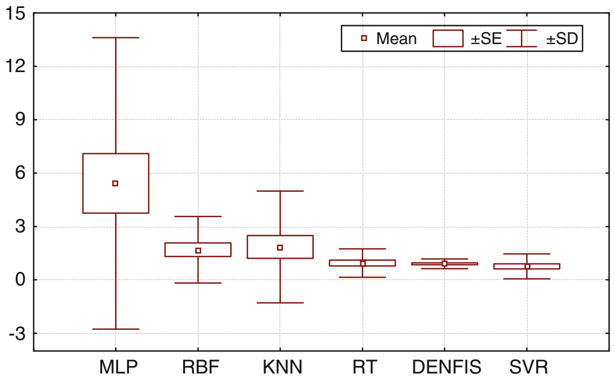

**Fig. 1** *Boxplots* of MRE

values of the RBF and KNN models are almost the double of the MMRE values of the RT, DENFIS, and SVR models. Clearly, the MLP model had the worst MMRE and StdMRE values.

Figure 1 shows MRE boxplots of the fault density prediction models to allow visual comparison of their performance. The middle of each box represents the MMRE of each model. As can be seen, the DENFIS model has the narrowest box and the smallest whiskers (i.e. the lines above and below from the box). That is followed by the SVR and RT models. The MLP model, however, has the widest box and the biggest whiskers. In summary, comparing these boxpolts clearly shows that the DENFIS, SVR, and RT models were more accurate in predicting fault density compared to the MLP, RBF, and KNN models.

Figure 2 plots the MMRE versus StdMRE that were achieved by each fault density prediction model; as another way to visualize the performance of these models. A good prediction model should appear in the lower left corner of this plot. It can be observed that the RT, DENFIS, and SVR models appear in the lower left corner of this plot as they achieved lower MMRE and StdMRE values compared to the other models. Moreover, the MLP model appears in the upper right corner of this plot, which indicates its poor performance.

In order to determine if there is a statistically significant difference among the prediction models, we performed the two-tailed Wilcoxon signed-rank test at 0.05 level of significance (95 % confidence level). This test was chosen because it does not require any underlying distribution in the data, and is not influenced by outlier data points. Table 4 presents the Z statistic values of the two-tailed Wilcoxon signed-rank test for the MRE values among all the six fault density prediction models. Significant results (i.e. $p < 0.05$) are in boldface. The results indicate that there is no significance difference among the KNN, RT, DENFIS,

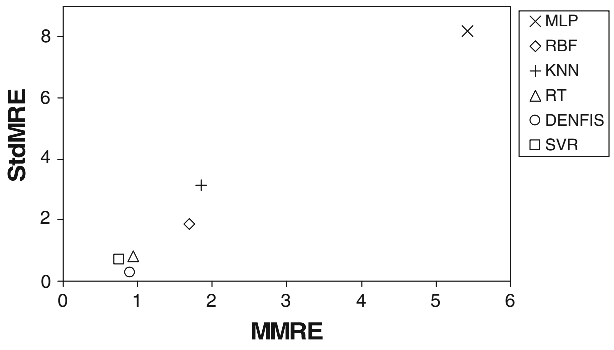

**Fig. 2** MMRE versus StdMRE

**Table 4** Wilcoxon signed-rank test

| | MLP | RBF | KNN | RT | DENFIS | SVR |
|---|---|---|---|---|---|---|
| MLP | | | | | | |
| RBF | **2.257**$*$ | | | | | |
| KNN | **2.029**$*$ | 0.829 | | | | |
| RT | **3.286**$*$ | **2.771**$*$ | 0.343 | | | |
| DENFIS | **2.743**$*$ | 1.114 | 0.171 | 0.200 | | |
| SVR | **3.286**$*$ | **2.543**$*$ | 0.886 | 1.343 | 1.686 | |

* Significant at $p < 0.05$

and SVR models. However, both the RT and SVR models are significantly better than the RBF model. In addition, the MLP model is significantly worse than the five other models.

## 5 Conclusion

This paper has investigated and empirically evaluated and compared six popular computational intelligence models in the context of fault density prediction in aspect-oriented systems. These models were MLP, RBF, KNN, RT, DENFIS, and SVR. The models were trained and tested, using leave-one-out procedure, on a dataset that consists of twelve aspect-level metrics (explanatory variables) that measure different structural properties of an aspect. It was observed that the DENFIS, SVR, and RT models were more accurate in predicting fault density compared to the MLP, RBF, and KNN models. The MLP model was the worst model, and all the other models were significantly better than it.

This paper provides original empirical evidences and interesting results for both software quality assurance and computational intelligence communities. It is the first comparative study in the context of fault density prediction in aspect-oriented systems. More studies, as more datasets in this emerging paradigm become available, should be conducted to further support the findings of this paper, and to accumulate knowledge. In addition, it would be interesting, as future work, to apply computational intelligence models to predict other quality attributes of aspect-oriented systems such as maintainability, and also to estimate the effort required to develop aspect-oriented systems.

## References

Basili V, Briand L, Melo W (1996) A validation of object-oriented design metrics as quality indicators. IEEE Trans Softw Eng 22:751–761

Bishop C (1995) Neural networks for pattern recognition. Oxford University Press, Oxford

Briand L, Wust J, Daly J, Porter V (2000) Exploring the relationships between design measures and software quality in object-oriented systems. J Syst Softw 51:245–273

Buhmann M, Albowitz M (2003) Radial basis functions:theory and implementations. Cambridge University Press, Cambridge

Burges C (1998) A tutorial on support vector machines for pattern recognition. Data Min Knowl Discov 2:121–167

Ceylan E, Kutlubay F, Bener A (2006) Software defect identification using machine learning techniques. In: The 32nd EUROMICRO conference on software engineering and advanced applications (EUROMICRO-SEAA'06), pp 240–247

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

Cristianini N, Shawe-Taylor J (2000) An introduction to support vector machines and other kernel-based learning methods. Cambridge University Press, Cambridge

Duda R, Hart P, Stork D (2001) Pattern classification, 2nd edn. wiley, New York

El-Emam K, Melo W, Machado J (2001) The prediction of faulty classes using object- oriented design metrics. J Syst Softw, 56:63–75

Elish K, Elish M (2008) Predicting defect-prone software modules using support vector machines. J Syst Softw 81:649–660

Filman R, Elrad T, Clarke S, Aksit M (2004) Aspect-oriented software development. Addison-Wesley Professional, Reading

Fioravanti F, Nesi P (2001) A study on fault-proneness detection of object-oriented systems. In: Fifth European conference on software maintenance and reengineering, pp 121–130

Gun S (1998) Support vector machines for classification and regression. University of Southampton, technical report

Guo L, Ma Y, Cukic B, Singh H (2004) Robust prediction of fault-proneness by random forests. In: 15th international symposium on software reliability engineering (ISSRE'04), pp 417–428

Gyimothy T, Ferenc R, Siket I (2005) Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Trans Softw Eng 31:897–910

Han J, Kamber M (2001) Data mining: concepts and techniques, 2nd edn. Morgan Kauffman, San Francisco

Hardle W (1989) Applied nonparametric regression. Cambridge university, Cambridge

Kasabov N, Song Q (2002) DENFIS: dynamic evolving neural-fuzzy inference system and its application for time-series prediction. IEEE Trans Fuzzy Syst 10:144–154

Khoshgoftaar T, Allen E, Deng J (2002) Using regression trees to classify fault-prone software modules. IEEE Trans Reliab 51:455–462

Korhonen K, Kangas A (1997) A application of nearest-neighbour regression for generalizing sample tree information. Scand J For Res 12:97–101

Myrtveit I, Stensrud E, Shepperd M (2005) Reliability and validity in comparative studies of software prediction models. IEEE Trans Softw Eng 31:380–391

Orr M (1996) Introduction to radial basis function networks, Institute for Adaptive and Neural Computation of the Division of Informatics at Edinburgh University, Scotland technical report

Quah T, Thwin M (2003) Application of neural networks for software quality prediction using object-oriented metrics. In: International conference on software maintenance (ICSM'03), pp 116–125

Quinlan R (1986) Induction of decision trees. Mach Learn 1:81–106

Smola A (1998) Learning with kernels. In: Department of Computer Science: Technical University Berlin, Germany

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

Wang Y, Witten I (1997) Inducing model trees for continuous classes. In: 9th European conference on machine learning, pp 128–137

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

Yu P, Systa T, Muller (2002) Predicting fault-proneness using OO metrics an industrial case study. In: Sixth European conference software maintenance and reengineering, pp 99–107