Soft Comput
DOI 10.1007/s00500-016-2284-x

METHODOLOGIES AND APPLICATION

An empirical study of some software fault prediction techniques for the number of faults prediction

Santosh S. Rathore[1] · **Sandeep Kumar**[1]

Abstract During the software development process, prediction of the number of faults in software modules can be more helpful instead of predicting the modules being faulty or non-faulty. Such an approach may help in more focused software testing process and may enhance the reliability of the software system. Most of the earlier works on software fault prediction have used classification techniques for classifying software modules into faulty or non-faulty categories. The techniques such as Poisson regression, negative binomial regression, genetic programming, decision tree regression, and multilayer perceptron can be used for the prediction of the number of faults. In this paper, we present an experimental study to evaluate and compare the capability of six fault prediction techniques such as genetic programming, multilayer perceptron, linear regression, decision tree regression, zero-inflated Poisson regression, and negative binomial regression for the prediction of number of faults. The experimental investigation is carried out for eighteen software project datasets collected from the PROMISE data repository. The results of the investigation are evaluated using average absolute error, average relative error, measure of completeness, and prediction at *level l* measures. We also perform Kruskal–Wallis test and Dunn's multiple comparison test to compare the relative performance of the considered fault prediction techniques.

Communicated by V. Loia.

✉ Sandeep Kumar
sandeepkumargarg@gmail.com

Santosh S. Rathore
sunnydec@iitr.ac.in

[1] Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, Roorkee, India

Keywords Software fault prediction · Zero-inflated Poisson regression · Genetic programming · Multilayer perceptron · Kruskal–Wallis test · Dunn's multiple comparison test

1 Introduction

From software development perspective, dealing with software faults is a vital and foremost important task. Presence of faults not only deteriorates the quality of the software, but also increases the development and maintenance cost of the software (Menzies et al. 2010). Therefore, identifying which software module is likely to be fault prone during early phases of software development may help in improving the quality of software system. By predicting number of faults[1] in software modules, we can guide software testers to focus on faulty modules first.

The objective of software fault prediction is to detect faulty software modules before the testing phase by using some structural characteristics of the software system. Software fault prediction model is generally constructed using the fault dataset from the previous releases of similar software projects, and later, it is applied to predict faults in the currently under development software system. A variety of techniques has been used earlier for software fault prediction (Basili et al. 1993; Khoshgoftaar et al. 1997; Catal 2011; Rathore and Kumar 2016a). Most of the techniques focused on classifying software modules into faulty or non-faulty categories. However, this type of prediction does not provide enough logistics to allot the limited quality assurance resources efficiently and optimally. Some of the faulty software modules can have relatively larger number

[1] Number of faults and fault counts both are same term. We used them interchangeably in this paper.

Published online: 28 July 2016

of faults compared to other modules and may require some extra efforts to fix them. So, allocating the quality assurance resources solely based on faulty and non-faulty information may result in inefficient use of resources.

The idea of the prediction of number of faults is attractive because it provides the probability that a certain number of faults will occur in the given software module that best describes the fault occurrence pattern of the given software system. This type of prediction can be very useful for the quality assurance team to narrow down the testing efforts to the modules having more number of faults.

Count models such as Poisson regression and negative binomial regression can be used to estimate the probability that certain number of faults will occur in each software module. Apart from count models, some other prediction techniques such as multilayer perceptron, decision tree regression, and genetic programming can also be used for predicting the fault occurrences in software modules. Khoshgoftaar et al. have evaluated count models for the number of faults prediction and found that count models were able to predict the number of faults accurately (Gao and Khoshgoftaar 2007). Some of the authors such as Afzal et al. (2008), (Rathore and Kumar 2016) and (Khoshgoftaar et al. 1992a), have investigated genetic programming, decision tree regression, and multilayer perceptron in the context of the prediction of number of faults and found that these techniques performed significantly accurate. However, no comprehensive investigation is available in the literature that has compared count models and other fault prediction techniques such as genetic programming, multilayer perceptron, and decision tree for the prediction of number of faults in given software modules. Gao and Khoshgoftaar (2007) performed a comparative study of five count models for the prediction of number of faults. They used hypothesis testing and other statistical measures to evaluate the relative performance of the considered count models. However, no evaluation was provided to assess the potential of the used count models for the prediction of number of faults. Additionally, no comparison was provided with other fault prediction techniques.

This paper presents a comprehensive investigation of six different fault prediction techniques for the prediction of number of faults. The techniques investigated are negative binomial regression (NBR), zero-inflated Poisson (ZIP) regression, multilayer perceptron (MLP), genetic programming (GP), decision tree regression (DTR), and linear regression (LR). Out of these six techniques, LR, MLP, NBR, and ZIP have been reported in various existing studies for the number of faults prediction, whereas it has been found that DTR and GP have been explored by only few researchers for the number of faults prediction and need further investigation. The experimental study is performed for eighteen software project datasets collected from the PROIMSE data repository (Menzies et al. 2016). The prediction accuracy and performance of the built fault prediction models are evaluated using average absolute error, average relative error, measure of completeness, and prediction at *level l* measures. The relative performance of six fault prediction techniques is evaluated by Kruskal–Wallis test and Dunn's multiple comparison test. The analysis and observations from this work can be helpful to obtain the ensemble of these methods for the prediction of number of faults. A similar study of the ensemble methods on the prediction of software maintainability has been recently reported in the literature (Elish et al. 2015).

The remainder of the paper is organized as follows. Section 2 describes the related work. Detail of fault prediction techniques along with the calibration of fault prediction models is given in Sect. 3. Performance evaluation measures are also discussed in the same section. Section 4 discusses the details of experimental design and the results of the presented empirical study. A comparative analysis of the proposed work with other similar existing works is given in Sect. 5. Section 6 discusses the possible threats to the validity of the study. The conclusions are given in Sect. 7.

2 Related work

Some of the techniques used for predicting the number of software faults include negative binomial regression (Ostrand et al. 2004), count models (Gao and Khoshgoftaar 2007), genetic programming (Afzal et al. 2008), generalized linear regression (Graves et al. 2000), and neural network (Rathore and Kumar 2015b).

Graves et al. (2000) reported a study for the prediction of number of faults using generalized linear regression technique. The study has been performed using various change history metrics collected from a large telephone switching system. The results suggested that the module size and other used complexity metrics produced the poor prediction accuracy. The best results were obtained by the combination of software metrics including modules age, changes made to module, and the ages of the changes. However, no performance measure was used to evaluate the appropriateness of generalized linear regression for the prediction of number of faults.

Ostrand et al. have performed a study for the prediction of number of faults and fault density using negative binomial regression (NBR) technique (Ostrand et al. 2004). A NBR model has been developed that predict the number of faults and fault density for the next release of the software system. The prediction was performed over two large industrial software systems. The results showed that developed fault prediction model was accurately predicting the number of faults in the software systems. Similar type of studies was reported in Ostrand et al. (2005a, b). Janes et al. (2006)

performed an experimental investigation for the identification of defect-prone classes using design metrics. The study was performed over five real-time telecommunication systems using three count models (Poisson regression, NBR, and zero-inflated NBR). The results found that zero-inflated NBR model produced the best performance among the other used count models.

In another study, Liguo (2012) performed an investigation to explore the effectiveness of negative binomial regression (NBR) technique to predict fault counts in given software modules. The performance of negative binomial regression model was compared with logistic regression model. The results found that in predicting software modules being fault-prone, logistic regression model outperformed NBR model. However, the results also suggested that NBR produced better prediction accuracy when used to predict multiple faults in one module. In 2007, Gao and Khoshgoftaar (2007) performed an empirical study to compare the performance of five count models for the prediction of number of faults. The study was carried out over two industrial software systems. The results found that zero-inflated negative binomial regression and hurdle negative binomial regression models produced better prediction accuracy. In another similar study, (Khoshgoftaar and Gao 2007) compared the performance of Poisson regression (PRM) and zero-inflated Poisson regression (ZIP) models with logistics regression model. The results found that PRM and ZIP models have the similar prediction accuracies as the logistic regression models. Out of ZIP and PRM models, ZIP model produced better prediction accuracy.

Scanniello et al. (2013) performed a study for software fault prediction using a clustering approach, and fault prediction models are built using the properties of classes in each cluster. The experiments were performed over 29 releases of eight open-source software systems. The results found that the presented clustering approach performed significantly accurate for the software fault prediction. Kpodjedo et al. (2009) investigated the effectiveness of various software design metrics for the prediction of number of defects and defect density over three open-source software systems. The fault prediction models were built using multi-dimensional linear regression, logistic regression, and Poisson regression techniques. The results found that the combination of design metrics with traditional software metrics significantly improved the performance of the software fault prediction. In 2010, Bacchelli et al. (2010) explored the use of popularity metrics of software components for the software defect prediction. The experimental study was carried out over four open-source software systems, and principal component analysis and regression analysis were used to build software fault prediction models. Results indicated that the use of only popularity metrics does not provide significant results for the number of faults prediction. However, the use of popularity metrics with other change metrics improved the fault prediction results.

Afzal et al. (2008) performed a study for the prediction of fault counts using genetic programming (GP). The prediction model was based on weekly fault count data and was built over three industrial software systems. The results showed that GP model produced statistically significant results for goodness of fit and predictive accuracy for fault count prediction. In another study, Marinescu (2014) performed an investigation for predicting changes and defects in the given software modules using the genetic programming. The experiments were performed over four open-source software systems having various object-oriented metrics. The results found that built fault prediction model predicted defects and changes in the software modules with significant precision and recall values. Recently, Rathore and Kumar (2015a) presented an approach for the number of faults prediction using genetic programming (GP) over several open-source software projects. Their results found that GP model has predicted number of faults in given software modules with significant accuracy and completeness.

In 2015, Chen and Yutao (2015) performed a study for the prediction of number of defects using different regression techniques. The experiments were performed over six open-source software projects, and results found that decision tree regression produced the best fault prediction results among the used different regression techniques.

Most of the studies reported in the literature focused on the usage of count models for the prediction of number of faults. Very few studies performed fault count prediction using other set of techniques. The study reported in Gao and Khoshgoftaar (2007) evaluated and compared various count models for number of faults prediction. However, no comparison has been provided with other fault prediction techniques. In contrast, our study aims to evaluate the performance of six fault prediction techniques including two count model-based techniques that are reported better than other count models-based techniques in the literature and four other fault prediction techniques for the number of faults prediction. We also perform statistical analysis to compare the relative performance of used fault prediction techniques.

3 Fault prediction techniques

This section presents the description of six fault prediction techniques that are used for building fault prediction models. We have used two count model-based techniques and four other fault prediction techniques. The previous studies related to count models found that negative binomial regression (Gao and Khoshgoftaar 2007) and zero-inflated Poisson regression models (Khoshgoftaar and Gao 2007) have produced significant results for fault prediction. There-

fore, we have selected these two techniques to perform the comparative study. Previously, some authors also applied linear regression and multilayer perceptron for the prediction of number of faults in given software modules (Graves et al. 2000; Khoshgoftaar et al. 1992a). Recently, genetic programming has been used by researchers to predict the fault counts in the software system (Afzal et al. 2008; Rathore and Kumar 2015a). Based on these observations, these six fault prediction techniques have been selected for building and evaluating fault prediction models.

3.1 Count models

Count model is a form of generalized linear model that is used to model the data, where the response variable is of count type. Count model is used when response variable follows the distribution other than a normal distribution (Hilbe 2012). A most commonly used count model is Poisson regression model (PRM). Other used count models are negative binomial regression (NBR), zero-truncated count models, zero-inflated count models, and hurdle count models. Poisson regression and negative binomial regression are the two basic types of count models. Other count models have been derived from these models by combining different distributions. For example, zero-inflated count models such as zero-inflated Poisson regression and zero-inflated negative binomial regression are the specific cases of PRM and NBR, respectively, which is used to model the data with excessive number of zeros in response variable. Similarly, hurdle-Poisson regression and hurdle negative binomial regression are the special cases of PRM and NBR, which assumed that zeros and positives in the response variable are not come from the same data generation process.

In addition to these count models, some other count models are also available, such as Poisson–inverse mixture model and random effect count models (Cameron and Trivedi 2013). However, it was observed these count models produced the similar results as produced by basic count models presented (Gao and Khoshgoftaar 2007). Therefore, these count models are not included in the study.

3.1.1 Negative binomial regression

Negative binomial regression is a Poisson–Gamma mixture model that addresses the overdispresion issue of the Poisson regression model (Greene 2011). Various maximization methods such as Newton–Raphson method and Berndt–Hall–Hall–Hausman (BHHH) can be used to estimate the parameters of the negative binomial regression. The probability distribution function of negative binomial regression can be obtained by the function given in Eq. 1.

$$Pr(y_i|x_i) = \frac{\Gamma(y_i + \alpha^{-1})}{y_i \Gamma(\alpha^{-1})} \left(\frac{\alpha^{-1}}{\alpha^{-1} + \mu_i}\right)^{\alpha^{-1}} \left(\frac{\mu_i}{\alpha^{-1} + \mu_i}\right)^{y_i} \tag{1}$$

Where, gamma function is denoted by $\Gamma(.)$ and μ represents the mean value. α is a constant parameter, and y_i represents the Poisson distribution. The expected value of y_i is given by $E(y_i|x_i) = e^{x_i\beta} = \mu_i$. Conditional variance is given by Eq. 2.

$$Var(y_i|x_i) = \mu(1 + \alpha\mu_i) = e^{x_i\beta}(1 + \alpha e^{x_i\beta}) \tag{2}$$

Since both μ and α are positive in negative binomial regression, the value of conditional variance of y exceeds the value of condition mean. If $\alpha = 0$, then condition variance becomes equal to the conditional mean (see Eq. 2) and the negative binomial regression reduced to Poisson regression. Further details of negative binomial regression can be found in Hilbe (2012).

3.1.2 Zero-inflated Poisson regression

Zero-inflated model is used to model the dataset with an excessive number of zeros in the response variable. It is a commonly occurring phenomenon in the software fault prediction process. A zero-inflated Poisson regression (ZIP) is a special case of Poisson regression to handle zero-inflated count data.

ZIP model was first introduced by Lambert (1992). It assumes that all zeros in the data occur from two sources, i.e., perfect and non-perfect sources. Perfect source represents the modules in which no faults occur, and the non-perfect source represents the modules in which the number of faults follows the Poisson distribution. In ZIP, parameter ψ is used to denote the probability of the modules from perfect source, and $1 - \psi$ is used to denote the probability of the modules from non-perfect source. The probability density function of response variable under ZIP is given by Eq. 3.

$$Pr(y_i|x_i, \mu_i, \psi_i) = \begin{cases} \psi_i + (1 - \psi_i)e^{\mu_i}, & y_i = 0 \\ (1 - \psi_i)\frac{e^{\mu_i}\mu_i^{y_i}}{y_i!}, & y_i = 1, 2, 3 \ldots \end{cases} \tag{3}$$

Additionally, two other link functions given in Eqs. 4 and 5 have been used to obtain the ZIP model,

$$\ln(\mu_i) = X_i'\beta \tag{4}$$

$$\text{logit}(\psi_i) = \ln\frac{\psi_i}{1 - \psi_i} = X_i'\gamma \tag{5}$$

where $y = [y_0, y_1, y_2, \ldots, y_k]$ is an unknown parameter vector of dimensions $(k + 1) * 1$.

The expected mean and variance of ZIP model are given by Eqs. 6 and 7, respectively,

$$E(y_i = \mu_i(1 - \psi_i) \tag{6}$$

$$Var(y_i) = (1 - \psi_i)(\mu_i + \psi_i \mu_i^2) \tag{7}$$

Maximum likelihood estimation (MLE) technique has been used to estimate the parameter values of ZIP. The detail of the ZIP can be found in Cameron and Trivedi (2013).

Probability density functions given in Eqs. 1 and 3 are used to define the relative likelihood of the independent variables in NBR and ZIP, respectively. The parameters of negative binomial regression and zero-inflated Poisson regression are estimated by using the maximum likelihood estimation. Since some of the software metrics have relatively large values, we have performed a square root transformation of these metrics. We also took the logarithmic transformation of LOC metric. These transformations helped us to fit the models better in term of the log likelihood ratio. The construction of NBR and ZIP models has been done using STATA tool.[2] The parameters of these models are initialized as follows. Newton–Raphson is used as optimization technique, standard error type is selected as robust, and maximum number of iterations can be up to 16000. Rests of the parameters are initialized with their default values.

3.2 Other fault prediction techniques

3.2.1 Linear regression

Linear regression is a statistical technique that establishes a relationship between a dependent variable and one or more independent variables using some statistical formula (Cohen et al. 2002). The model is in the form of an equation where dependent variable is represented in terms of independent variables. The general form of linear regression can be described using Eq. 8.

$$Y = b_0 + b_1 X_1 + b_2 X_2 + \cdots + b_n X_n \tag{8}$$

where Y represents the dependent variable and $X_1, X_2, \ldots X_n$ represent independent variables. Factors $b_1, b_2, \ldots, b_n$ are the coefficients of the independent variables. The additional constant b_0 is called intercept. It shows the prediction that the model would make if all the Xs were zero. The values of the coefficients and intercept are estimated by least squares method (Strutz 2011).

A linear regression model is of two types (Draper and Smith 1998). (a) Univariate linear regression, where dependent variable is influenced by only one independent variable, and (b) multivariate linear regression, where dependent variable is influenced by more than one independent variables. In our study, we have used multivariate linear regression model.

We have used Weka[3] implementation of linear regression to calibrate the fault prediction models. Least square method is to estimate the parameter values for linear regression. Rests of the parameters are initialized to the default values.

3.2.2 Multilayer perceptron

Multilayer perceptron (MLP) or simply neural network (NN) is a prediction algorithm that is inspired from the working of the biological neural network (Kotsiantis 2007). MLP consists a series of processing elements interconnected through the connection weights in the form of layers. During the training phase, based on the domain knowledge, they develop an internal representation that maps the input stimulus space to the output response space. MLP utilized a supervised learning technique called back-propagation algorithm for training the network.

The working of MLP can be described using Eqs. 9 and 10.

$$net_k = w_{1k}x_1 + w_{2k}x_2 + \cdots + w_{mk}x_m + b_k \tag{9}$$

$$O_k = f(net_k) \tag{10}$$

For the kth processing element, MLP computes a weighted sum of its input elements x_j (independent variable) and basis b_k using some statistical formula. This weighted sum is then processed by activation function to generate the output O_k. $w_{1k}, w_{2k}, \ldots, w_{mk}$ are the weights associated with each input layer. Function $f(.)$ denotes the activation function.

The working of MLP can be described as follows. The training data are iteratively fed into the neural network. After each iteration, the output of the neural network is compared with the desired output and error is calculated. The error is used to update the weight of the hidden layer and again feedback to the neural network. The weight is updated in such a way that after each iteration error decreases and neural network model produces the output that is closer to the desired output.

We have used Weka implementation of multilayer perceptron to construct fault prediction model. The parameter values for the multilayer perceptron are initialized as follows: To train the neural network, we have used back-propagation algorithm, and sigmoid is used as the activation function. Number of hidden layers is five. For the rest of the parameters, default value as define in Weka is used.

[2] Stata: Data Analysis and Statistical Software. http://www.stata.com/.

[3] Weka Data Mining Tool. http://www.cs.waikato.ac.nz/ml/weka/.

3.2.3 Genetic programming

Genetic programming (GP) is an evolutionary algorithm that mimics the working of biological evaluation to find the best possible solution to perform a user specified task (Smith 1980). It is a search-based algorithm, which searches the optimal solutions to perform a given computational task. GP is a specialized case of genetic algorithm (GA), but compared to GA, the population structure (individual) of GP is not fixed length character string (Goldberg 1989). The individuals in GP are expressed as syntax trees with the nodes indicating the instructions to execute and are called functions. The leaves of the tree are called terminals, which may consist of independent variables of the problem and random constants.

Generally, genetic programming starts with an arbitrarily generated population of a potential solution space. Subsequently, it iteratively transforms an initial population of potential solutions into the next generation of the potential solutions by applying genetic operators (typically crossover and mutation). The individuals who will survive in the next iterations are selected using a fitness function. The whole process is continued in this manner until the termination condition is met, which is generally bound to the maximum number of generations. The single best individual who has survived after the termination condition met is chosen as resulting solution and used for prediction.

The genetic programming model has been constructed using GPLAB version 3.0 toolbox (a genetic programming toolkit) available for the MATLAB programming language. Use of genetic programming for fault prediction requires tuning of various control parameters. Depending upon the characteristics of the software project, users can select various parameter values accordingly. GPLAB toolbox provides the interface to the user to select values of the various controlling parameters. All the controlling parameters with their default values are defined in the *availableparams.m* file of GPLAB toolbox. In order to decide best suitable parameter values for corresponding datasets, we have performed a series of experiments and adjusted the parameter values after the experiments. Initially, we tried different function sets and selection methods and kept rest of the parameter values fixed. Iteratively, we have varied others parameter values such that the error between actual value and predicted value minimized. Once we determine the best possible values for all the required parameters, we run all the experiments with these values.

In each iteration of genetic programming, the priority of each individual from the population is ranked based on the values of the fitness function. Fitness function is a type of objective function that is used to identify, how good a potential solution is relative to the other potential solutions. It helps to select which potential solution will continue in the next generation and which will die. The fitness function in our experiments is the square root of the absolute difference between the predicted values and the actual values of the number of faults in all fitness cases, as given in Eq. 11.

Table 1 Control parameter values used for the GP implementation

Control parameters	Value
Population size	200
Number of generation	1000
Termination condition	Limited to maximum generation value
Function set	{+, −, ∗,sin,cos,/,log}
Terminal set	{x}
Tree initialization	Ramped half-and-half
Genetic operator	Crossover, mutation
Selection method	Roulette wheel
Elitism	Replace

$$\text{Fitness} = (1/n)\sum_{i=1}^{n} sqrt(|(\bar{E}_i - E_i|) \tag{11}$$

where $\bar{E}_i$ is the predicted value of the number of faults in a software module and E_i is the corresponding actual value of the number of faults and n is the number of modules. Since, in our datasets, some of the attributes contain outlier values. Therefore, square root of the difference is used to mitigate the effect of outliers on the selection of potential solution for the next generations. Table 1 shows the values of the main parameters used in building the GP models. The rest of the parameter values are kept default as available in GPLAB toolbox.

3.2.4 Decision tree regression

Decision tree regression is a type of decision tree for inducing trees of regression models. It is also known as M5P algorithm, which is an implementation of Quinlan's M5 algorithm (Quinlan et al. 1992). M5P algorithm combines the capability of conventional decision tree with linear regression functions at the nodes (Wang and Yao 2013). In M5P algorithm initially, a conventional decision tree algorithm is used to build a tree. This decision tree uses a splitting criterion that minimizes the intra-subset variation in the class values of instances that go down each branch. The attribute which maximizes the expected error reduction is chosen as the root node. The standard deviation reduction is calculated using the formula given in Eq. 12.

$$\text{SDR} = sd(T) - \sum_i \frac{|T_i|}{|T|} \times sd(T_i) \quad (12)$$

Next, the tree is pruned back from each leaf. Finally, a smoothing procedure is used to compensate the sharp discontinuities, which will inevitably occur between adjacent linear models at the leaves of the pruned tree.

We have used Weka implementation of M5P algorithm. For experimentation, the minimum number of instances to allow at a leaf node is set to 4. Rests of the parameters are initialized with their default values.

3.3 Performance evaluation measures

We have used four different performance evaluation measures to evaluate the prediction results. These performance evaluation measures are average absolute error, average relative error, prediction at *level l*, and measure of completeness. The description of these measures is given in this section.

(i) Average absolute error (AAE): AAE measures the average magnitude of the errors in a set of prediction. It shows the difference between the predicted value and the actual value. It is defined by Eq. 13.

$$\text{AAE} = (1/n) \sum_{i=1}^{n} |(\bar{Y}_i - Y_i)| \quad (13)$$

(ii) Average relative error (ARE): ARE calculates how large the absolute error is compared with the total size of the object measured. It is defined by Eq. 14.

$$\text{ARE} = (1/n) \sum_{i=1}^{n} |(\bar{Y}_i - Y_i)|/(Y_i + 1) \quad (14)$$

Here, $\bar{Y}_i$ is the predicted number of faults in a software module and Y_i is the corresponding actual value of faults. n is the number of modules. In the case of ARE, sometimes value of Y_i can be zero. To mitigate this issue, we added '1' with the value of Y_i at the denominator to make the definition always well defined (Khoshgoftaar et al. 1992). A small value of AAE and ARE measures indicates that we have a good prediction model (Conte et al. 1986).

(iii) Prediction at *level l*: It measures the percentage of observations whose value lies within $l\,\%$ of the actual value (Conte et al. 1986). It is defined by Eq. 15.

$$Pred(l) = k/n \quad (15)$$

Where k is the number of observations whose value lies within $l\,\%$ of the actual values, l is the threshold value, and n is the total number of observations. R. Veryard (Veryard 2014) suggested that for a model to be considered acceptable, value of l should be less than or equal to 0.30. Therefore, we have used 0.30 as the value of l. This means that the predicted value must be within the 30 % range of the actual value to consider the prediction acceptable for the given module.

(iv) Measure of completeness: Completeness of a prediction model is defined as a ratio of the number of faults found in the modules predicted as faulty to the total number of faults in the software system (Briand and Jurgen 2002). It indicates the percentage of faults that have been found when we used the given prediction model.

Completeness value higher than 100 % indicates that prediction model has predicted a large number of faults falsely. It will result in the increment of fault finding efforts. The low value of completeness indicates that prediction model has left many faults undetected. These faults would then slip through the subsequent phases of software development, where they require more efforts to detect.

4 Empirical case study

4.1 System description

We have used eighteen software project datasets publicly available in PROMISE data repository[4] to perform the experiments. Out of eighteen datasets, fifteen datasets belong to the Apache Camel, Apache Xerces, Apache Xalan, Apache Ant, and PROP projects. These fifteen datasets contain 20 object-oriented metrics. The rest of the three datasets belong to Eclipse project[5] and contain various source code metrics.

PROP dataset has been collected from an industrial software system, which was written in the Java programming language. We have used V4, V40, V85, V121, V157, and V185 versions of the PROP dataset. Xerces, Xalan, Ant, and Camel are the Apache products, written in Java and C++ programming languages, and are developed in an open-source environment. Xerces is a Apache collection of software libraries used for parsing, validating and manipulating XML. Currently, its parsers are available for Java, C++, and Perl programming languages. Camel is rule-based routing and mediation engine that offers the interfaces for the Enterprise Integration Patterns (EIPs). It is an integration framework, which implements all EIPs. Xalan is an Extensible Stylesheet Language Transformations (XSLT) processor that trans-

[4] PROMISE data repository. http://openscience.us/repo/defect/.

[5] Eclipse data repository. https://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse/.

forms XML documents into HTML. Currently, its parsers are available in C++ and Java programming languages. Ant is a software tool used for automating the software building process. Ant contains various Java libraries and command line tools to help in building Java applications. It is written in Java programming language. Eclipse is an integrated development environment (IDE) system, which contains a base workspace and an extensible plug-in system for customizing the environment. It is written in Java programming language. Initially, it was developed by IBM corporation, and later, it become openly available under the GNU General Public License.

Most of the techniques used in this study require a relatively large size of data to train the fault prediction model. Therefore, we have selected datasets of medium size or larger. In Venkata et al. (2006), it is defined that dataset having more than 500 modules falls under the category of medium size. Based on this criterion, we have selected eighteen datasets out of the total datasets available in PROMISE data repository. In this study, a software module is referred to as a class in the given software system (Jureczko 2011).

The dependent variable, in our study, is the number of faults in software modules. The fault dataset contained the value of the number of faults that were found during the various phases of software development. It includes the faults found in requirement, design, coding, and testing. phases of software system. These fault values were recorded in a database associated with the other information of software modules. A description of these datasets is given in Table 2.

Table 3 provides brief description of the used software metrics. The detail description of these metrics can be found in Jureczko (2011).

Table 2 Description of the fault datasets used for the study

Dataset	# Modules	Non-commented LOC	# Faulty Modules	% of faults
PROP V4	3022	664KLOC	265	9.61
PROP V40	4053	848KLOC	467	13.02
PROP V85	3078	643KLOC	948	44.50
PROP V121	2998	628KLOC	426	16.56
PROP V157	2497	438KLOC	367	17.23
PROP V185	2826	593KLOC	269	10.52
Xerces 1.3	503	167KLOC	69	13.71
Xerces 1.4	589	141KLOC	438	74.36
Camel 1.2	609	66KLOC	217	55.35
Camel 1.4	873	98KLOC	145	19.91
Camel 1.6	966	113KLOC	189	19.56
Ant 1.7	746	208KLOC	167	22.38
Xalan 2.4	724	225KLOC	111	15.33
Xalan 2.5	804	304KLOC	388	48.25
Xalan 2.6	886	411KLOC	412	46.50
AS _ eclipse 2.0	6730	796KLOC	976	14.50
AS _ eclipse 2.1	7889	987KLOC	855	10.83
AS _ eclipse 3.0	10594	1305KLOC	1569	14.81

4.2 Experimental design

Using the methodologies discussed in Sect. 3, we have built six software fault prediction models. We have used a cross-validation approach to build and evaluate the fault prediction models (Kohavi 1995). In K-fold cross-validation approach, original fault dataset, D, is randomly divided into K mutually exclusive folds $D_1, D_2, \ldots, D_K$ of approximately equal size. The learning technique is then trained and tested K times, and each time it is trained over D/D_t and is tested over D_t. Overall accuracy is estimated by averaging the results of each round. The overview of the used cross-validation approach is given in Fig. 1. The reason to choose cross-validation approach is to avoid any biased results due to the lucky data splitting (Gao and Khoshgoftaar 2007).

We have used tenfold cross-validation approach to perform the experiments. Each of the complete dataset is divided into ten mutually exclusive folds of approximately equal size . To partition the dataset for the cross-validation, we generate stratified fold of the dataset. Stratification is important for cross-validation in creating K number of fold from the dataset as it keeps the data distribution in each fold close to that in the entire dataset (Witten and Frank 2005). We have used RemoveFolds[6] filter available in the Weka machine learning tool to partition the dataset. For each iteration, ninefolds are used as training dataset and rest onefold is used as testing dataset as shown in Fig. 1. This process goes through ten iterations, and the results are averaged over the iterations. For example, consider Camel 1.2 dataset, we have divided Camel 1.2 dataset into tenfolds of approximately equal size. Original Camel 1.2 dataset contained 609 modules. We have divided it into tenfolds; therefore, each fold consists 61 modules (approximately). For the first iteration, first ninefolds (548 modules, approximately) are used to train the learning technique and last fold (61 modules, approximately) is used as test dataset to evaluate the performance of learning technique. This process continues for the ten iterations, and error rate of each iteration is averaged over the iterations to calculate the overall error rate.

We have built different fault prediction models using six different fault prediction techniques (as described in Sect. 3) for eighteen software fault datasets. A total of 108 (6×18) experiments have been performed in the presented study. Finally, we have evaluated the performance of fault prediction techniques using average absolute error (AAE), average

[6] RemoveFolds Filter. http://weka.sourceforge.net/doc.dev/weka/filters/supervised/instance/StratifiedRemoveFolds.html.

Table 3 Description of the software metrics

Metric	Description
WMC	Number of methods defined in a class
CBO	Count the number of classes coupled to class
RFC	Count the number of distinct methods invoke by a class in response to a received message
DIT	Depth of a class within the class hierarchy from the root of inheritance
NOC	Number of immediate descendants of a class
IC	Count the number of coupled ancestor classes of a class
CBM	Count the number of added or redefined methods those are coupled with the inherited methods
CA	Count the number of dependent classes for a given class
CE	Count the number of classes to which a class depends
MFA	Shows the fraction of the methods inherited by a class to the methods accessible by the functions defined in the class
LCOM	Subtraction of method-pairs that do not share a field to the method-pairs that do
LCOM3	Counts the number of connected components in a method graph
CAM	Computes the cohesion among methods of a class based on the parameters list
MOA	Count the number of data members declared as class type
NPM	Number of public methods defines in a class
DAM	Computes the ratio of private attributes in a class
AMC	Measures the average method size for each class
LOC	Counts the total number of lines of code of a class
CC	Counts the number of logically independent paths in a method
Other Software Metrics	pre, NOM_sum, NSM_avg, ArrayCreation, ArrayInitializer, ArrayType, CharacterLiteral, ConditionalExpression, ContinueStatement, DoStatement, FieldAccess, Javadoc, LabeledStatement, ParenthesizedExpression, PrefixExpression, QualifiedName, ReturnStatement, SuperMethodInvocation, SwitchStatement, ThisExpression, ThrowStatement

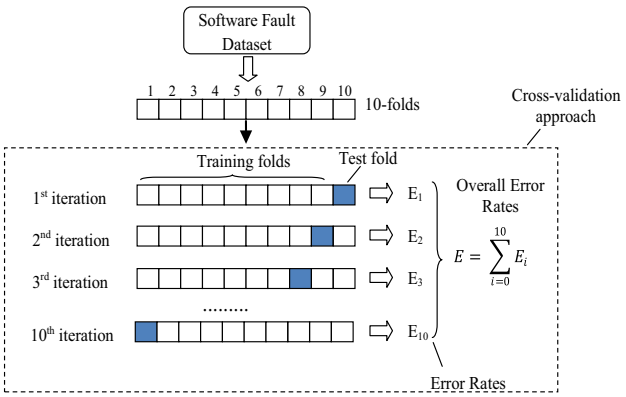

Fig. 1 Overview of experimental design

relative error (ARE), prediction at *level l*, and measure of completeness analysis. To assess the magnitude of difference among different fault prediction techniques, we have performed Kruskal–Wallis test and Dunn's multiple comparison test.

4.3 Results and analysis

We initially examine the results of AAE and ARE analysis of used fault prediction techniques. Later, we discuss the results of prediction at *level l*, and measure of completeness, Kruskal–Wallis test, and Dunn's multiple comparison test.

4.3.1 AAE and ARE analysis

Table 4 shows the results of AAE and ARE (error rate) analysis for all the eighteen datasets. Table contained the median values of the absolute error and relative error for all the datasets. A relative comparison based on the AAE and ARE values showed that linear regression (LR) and decision tree regression (DTR) techniques produced the lower error rate values for most of the datasets and exhibited better prediction accuracy compared to other considered fault prediction techniques. Genetic programming and multilayer perceptron are the third and fourth best fault prediction techniques, respectively. While negative binomial regression (NBR) and zero-inflated poisson regression (ZIP) have produced relatively lower prediction accuracy, the AAE and ARE values of NBR and ZIP were higher compared to other used fault prediction techniques.

Figure 2 shows the box-plot diagrams of AAE and ARE measures. *X*-axis indicates the fault prediction techniques under consideration, and *y*-axis indicates the median values of the errors produced by the used fault prediction techniques.

Table 4 AAE and ARE analysis of six fault prediction techniques for all datasets

Datasets		Linear regression	Multilayer perceptron	Decision tree regression	Genetic programming	Negative binomial regression	Zero-inflated poisson regression
Xerces 1.3	AAE	0.56	0.74	0.34	0.46	1.24	0.96
	ARE	0.39	0.5	0.2	0.33	1.16	0.77
Xerces 1.4	AAE	1.73	2.24	1.68	2.58	1.07	0.98
	ARE	0.57	0.68	0.47	1.03	0.5	0.42
Camel 1.2	AAE	1.07	0.98	1.01	1.02	0.94	0.81
	ARE	0.61	0.51	0.56	0.6	0.6	0.57
Camel 1.4	AAE	0.48	0.63	0.43	0.62	0.84	0.66
	ARE	0.27	0.38	0.24	0.41	0.73	0.5
Camel 1.6	AAE	0.67	0.92	0.63	0.7	0.93	0.84
	ARE	0.4	0.64	0.35	0.41	0.78	0.68
Ant 1.7	AAE	0.38	0.53	0.39	0.61	0.97	1.37
	ARE	0.2	0.27	0.2	0.4	0.88	1.1
Xalan 2.4	AAE	0.23	0.33	0.25	0.23	1.18	0.92
	ARE	0.12	0.22	0.14	0.16	1.08	0.81
Xalan 2.5	AAE	0.54	0.75	0.56	0.49	0.79	0.78
	ARE	0.39	0.51	0.4	0.34	0.55	0.56
Xalan 2.6	AAE	0.53	0.63	0.52	0.68	1.01	1.06
	ARE	0.33	0.36	0.3	0.45	0.71	0.6
Prop V4	AAE	0.11	0.14	0.12	0.11	0.64	0.45
	ARE	0.06	0.08	0.06	0.08	0.61	0.41
Prop V40	AAE	0.2	0.22	0.19	0.23	0.75	0.26
	ARE	0.1	0.12	0.1	0.16	0.73	0.23
Prop V85	AAE	0.91	0.89	0.83	0.93	1.12	0.89
	ARE	0.49	0.46	0.43	0.46	0.64	0.44
Prop V121	AAE	0.26	0.28	0.27	0.26	0.64	0.4
	ARE	0.15	0.13	0.14	0.18	0.56	0.26
Prop V157	AAE	0.31	0.37	0.34	0.46	0.69	0.65
	ARE	0.16	0.19	0.17	0.32	0.57	0.49
Prop V185	AAE	0.19	0.24	0.21	0.21	0.81	0.32
	ARE	0.1	0.11	0.12	0.16	0.78	0.22
AS_Eclipse 2.0	AAE	0.24	0.29	0.24	0.3	0.64	0.33
	ARE	0.14	0.14	0.13	0.19	0.53	0.18
AS_Eclipse 2.1	AAE	0.15	0.16	0.16	0.15	0.57	0.44
	ARE	0.08	0.07	0.08	0.1	0.52	0.37
AS_Eclipse 3.0	AAE	0.23	0.23	0.24	0.3	0.62	0.32
	ARE	0.13	0.11	0.13	0.19	0.52	0.17

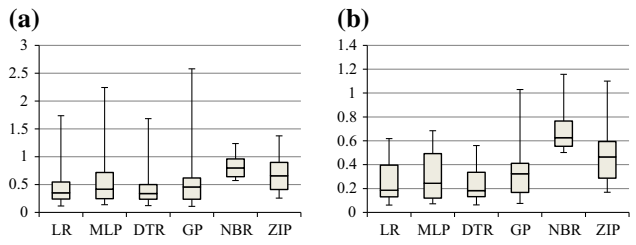

Fig. 2 Box-plot analysis for AAE and ARE measures for all the datasets

The different parts of box-plot show the minimum, maximum, median, first quartile, and third quartile of the samples. The line in the middle of the box shows the median of the samples. The results of box-plot analysis are summarized as below:

– With respect to AAE measure, DTR technique produced the lowest median value, LR, MLP, and GP produced moderate median values, whereas NBR and ZIP pro-

Table 5 Kruskal–Wallis test: AAE over all datasets

Kruskal–Wallis test for AAE	
K (observed value)	185.725
K (critical value)	11.07
DF	5
p value (two-tailed)	<0.0001
Alpha	0.05

H0: the samples come from the same population. Ha: The samples do not come from the same population. As the computed *p* value is lower than the significance level alpha = 0.05, we reject the null hypothesis H0 and accept the alternative hypothesis Ha

Table 6 Kruskal–Wallis test: ARE over all datasets

Kruskal–Wallis test for ARE	
K (observed value)	324.435
K (critical value)	11.070
DF	5
p value (two-tailed)	<0.0001
Alpha	0.05

H0: The samples come from the same population. Ha: The samples do not come from the same population. As the computed *p* value is lower than the significance level alpha = 0.05, we reject the null hypothesis H0 and accept the alternative hypothesis Ha

duced the highest median values. GP produced the lowest minimum values, LR, DTR and MLP produced moderate minimum values, and NBR and ZIP produced the highest minimum values. For maximum value, ZIP and NBR produced the lowest values, LR and DTR produced moderate values, and GP and MLP produced the highest values.

- With respect to ARE measure, LR and DTR produced the lowest median values, GP and MLP produced moderate median values, and NBR and ZIP produced the highest median values. Similarly, LR and DTR produced the lowest minimum values, MLP and GP produced moderate minimum values, and ZIP and NBR produced the highest minimum values. DTR produced the lowest maximum value, LR, GP, and MLP produced the moderate maximum values, and NBR and ZIP produced the highest maximum values.
- Overall, it is clear from the figures that for both the measures (AAE and ARE), LR and DTR have performed similar and outperformed other considered fault prediction techniques. GP and MLP performed moderately, while NBR and ZIP performed relatively poor compared to other used fault prediction techniques.

To further analyze, whether the performance difference of six fault prediction techniques is statistically significant or not, we have performed Kruskal–Wallis test on the fault prediction techniques under consideration. Since the predicted value of faults follow non-normal distribution, we have analyzed the data using the Kruskal–Wallis test (Casella 2008). Kruskal–Wallis (also known as one-way ANOVA by rank) is a nonparametric test used to test whether the samples are drawn from the same population or not. Absolute error and relative error values of each dataset are used to perform the Kruskal–Wallis test. During tenfold cross-validation, each of the dataset was split into ten parts. So, there are 180 comparison points giving during this test. The results of Kruskal–Wallis test are presented in Tables 5 and 6.

The results of the Kruskal–Wallis test indicate that the six used fault prediction techniques considered in this study have statistically significant performance difference from each other for at least one sample. During this test, the significant value, i.e., α is set to 0.05, indicating 95 % of the confidence interval (Juristo and Moreno 2013). For both the AAE and ARE measures, *p* value is lower than the value of α, and thus, the alternate hypothesis is accepted, which depicts that the six fault prediction techniques under consideration performed significantly different from each other.

Further, we have conducted multiple comparison test to determine the samples in which a significant difference has been found in the Kruskal–Wallis test. We proceeded with Dunn's multiple comparison test with mean as dependent variable (Bland and Altman 1995). If the difference of the group mean surpasses the critical value, then we can conclude that the difference is significant at given *p* value. Table 7 shows the results of multiple comparison test for AAE and ARE measures. The tables also contained the minimum, maximum, mean, and standard deviation values of each fault prediction technique (sorted in ascending order). The observations drawn from the Dunn's multiple comparison test are as summarized below:

- With respect to AAE measure, treatments involving NBR and ZIP techniques have performed significantly different from the other fault prediction techniques. For the rest of the treatments, no significant differences have been found.
- With respect to ARE measure, treatments involving DTR, NBR, and ZIP techniques have performed significantly different from the other considered fault prediction techniques. For all other treatments, no significant difference has been found.
- Overall, it was found that LR, GP and MLP fault prediction techniques did not perform significantly different from each other, whereas DTR, NBR and ZIP fault prediction techniques performed significantly different from the other considered fault prediction techniques.

Table 7 Dunn's multiple comparison test: for AAE and ARE, *(no diff.= no difference and sig. diff.= significant different)

AAE Measure						ARE measure					
Variable	Observations	Minimum	Maximum	Mean	SD	Variable	Observations	Minimum	Maximum	Mean	SD deviation
DTR	180	0.079	2.383	0.481	0.411	DTR	180	0.002	1.942	0.254	0.213
LR	180	0.059	2.441	0.507	0.428	LR	180	0.036	0.885	0.270	0.192
MLP	180	0.073	2.967	0.621	0.546	MLP	180	0.045	1.670	0.347	0.290
GP	180	0.028	5.724	0.643	0.810	GP	180	0.028	2.167	0.361	0.309
ZIP	180	0.046	6.933	0.819	0.801	ZIP	180	0.030	2.885	0.511	0.339
NBR	180	0.074	6.567	0.965	0.647	NBR	180	0.046	1.796	0.704	0.266
p value = 0.0033, critical value = 96.50, n = 180						p value= 0.003, critical value = 96.50, n = 180					
Treatments	Absolute diff.	p value	Results			Treatments	Absolute diff.	p value	Results		
LR versus MLP	72.231	0.028	No diff.			LR versus MLP	68.667	0.037	No diff.		
LR versus DTR	20.572	0.532	No diff.			LR versus DTR	29.553	0.369	No diff.		
LR versus GP	37.872	0.249	No diff.			LR versus GP	91.631	0.005	No diff.		
LR versus NBR	344.908	<0.0001	Sig. diff.			LR versus NBR	461.853	<0.0001	Sig. diff.		
LR versus ZIP	207.361	<0.0001	Sig. diff.			LR versus ZIP	266.703	<0.0001	Sig. diff.		
MLP versus DTR	92.803	0.005	No diff.			MLP versus DTR	98.219	0.003	Sig. diff.		
MLP versus GP	34.358	0.296	No diff.			MLP versus GP	22.964	0.485	No diff.		
MLP versus NBR	272.678	<0.0001	Sig. diff.			MLP versus NBR	393.186	<0.0001	Sig. diff.		
MLP versus ZIP	135.131	<0.0001	Sig. diff.			MLP versus ZIP	198.036	<0.0001	Sig. diff.		
DTR versus GP	58.444	0.075	No diff.			DTR versus GP	121.183	0.000	Sig. diff.		
DTR versus NBR	365.481	<0.0001	Sig. diff.			DTR versus NBR	491.406	<0.0001	Sig. diff.		
DTR versus ZIP	227.933	<0.0001	Sig. diff.			DTR versus ZIP	296.256	<0.0001	Sig. diff.		
GP versus NBR	307.036	<0.0001	Sig. diff.			GP versus NBR	370.222	<0.0001	Sig. diff.		
GP versus ZIP	169.489	<0.0001	Sig. diff.			GP versus ZIP	175.072	<0.0001	Sig. diff.		
NBR versus ZIP	137.547	<0.0001	Sig. diff.			NBR versus ZIP	195.150	<0.0001	Sig. diff.		

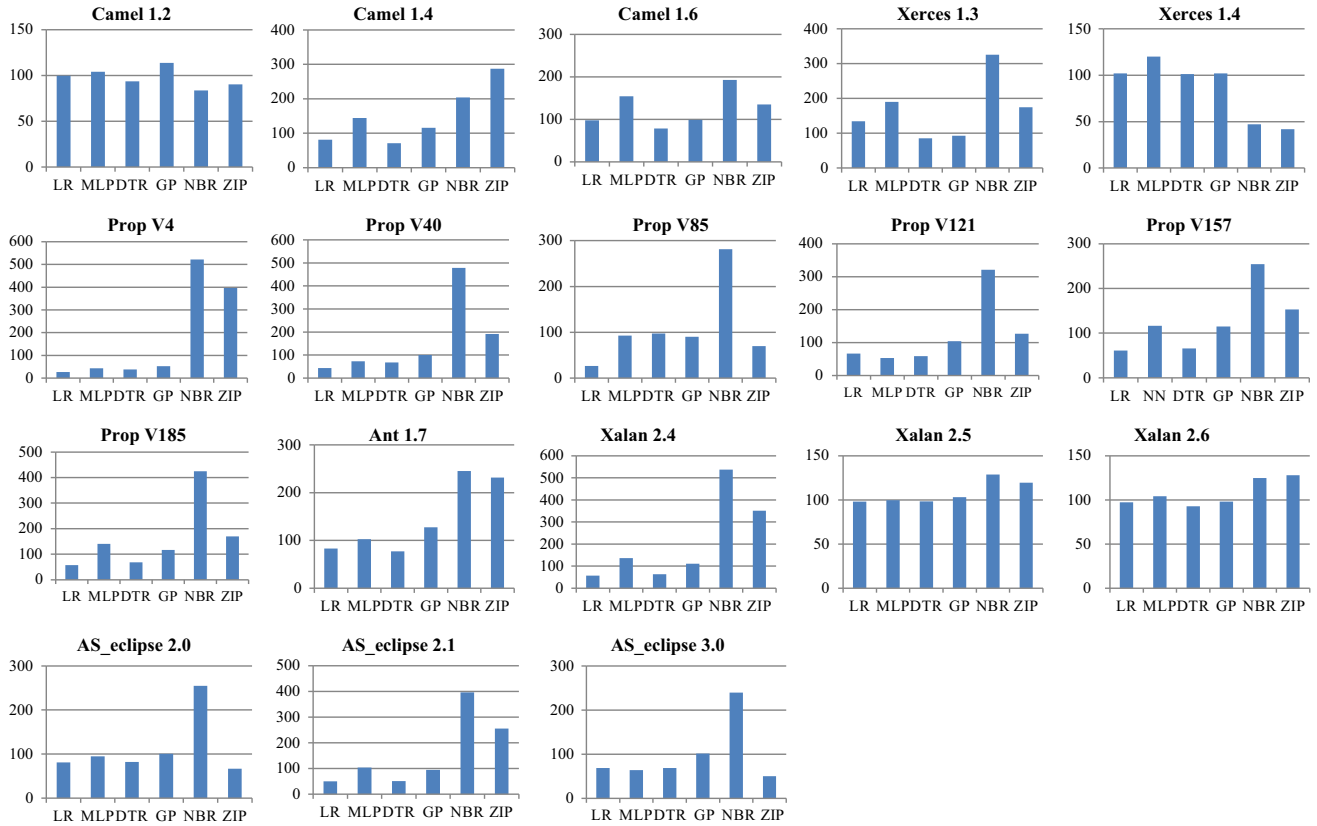

Fig. 3 Completeness analysis of six fault prediction techniques for all datasets, (*x*-axis shows the name of the considered fault prediction techniques, and *y*-axis shows the completeness value achieved by each fault prediction technique). *GP* genetic programming, *MLP* multilayer perceptron, *LR* linear regression, *NBR* negative binomial regression, *ZIP* zero-inflated poisson regression, *DTR* decision tree regression

– DTR technique produced the lowest mean values for both AAE and ARE measures. LR, MLP and GP techniques produced the moderate mean values, while NBR and ZIP techniques produced the higher mean values.

4.3.2 Measure of completeness

The results of measure of completeness analysis are given in Fig. 3. Each subfigure is labeled with the name of the dataset for which the study was performed. A prediction model with completeness close to 100 % is preferred. It is clear from the figures that for the most number of datasets (13 out of 18), genetic programming has achieved the completeness close to 100 %. While in rest of the datasets, DTR, LR and MLP achieved completeness close to 100 %, NBR and ZIP techniques generally performed poor for completeness measure.

4.3.3 Prediction at level l analysis

Figure 4 shows the results of *pred(0.3)* of each fault prediction technique for all eighteen datasets. It can be seen from the figures that generally GP and DTR techniques produced the higher *pred(0.3)* value compared to other fault prediction techniques. LR and MLP performed moderately, while NBR and ZIP performed poorly for this evaluation measure. However, none of the fault prediction technique achieved the highest *pred (0.3)* value consistently, but generally, for all the datasets the top four slots are occupied by DTR, GP, MLP, and LR techniques. When considering the results of six used fault prediction techniques using all four-performance evaluation measures, a number of observations can be drawn. The observations based on the empirical study have been summarized below:

– For AAE measure, decision tree regression produced the lowest mean value and it generally outperformed all other fault prediction models, while negative binomial regression model performed worst for AAE measure.
– For ARE measure, again decision tree regression has produced the best prediction results followed by linear regression, multilayer perceptron, and genetic programming, respectively.
– Overall, among the considered fault prediction techniques in the study, decision tree regression, genetic

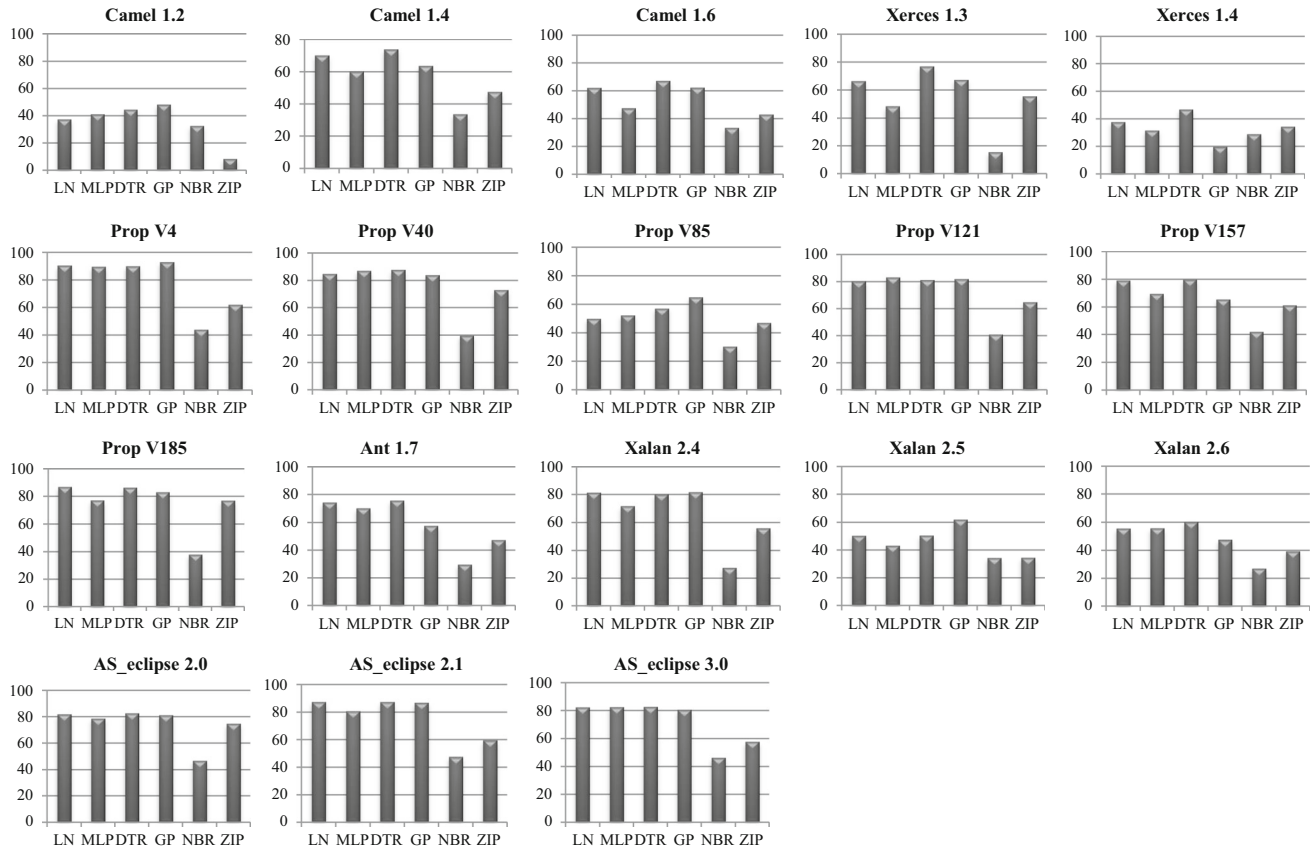

Fig. 4 Pred(0.3) for each model for all datasets, (*x*-axis shows the name of the considered fault prediction techniques, and *y*-axis shows the pred(0.30) value achieved by each fault prediction technique). *GP* genetic programming, *MLP* multilayer perceptron, *LR* linear regression, *NBR* negative binomial regression, *ZIP* zero-inflated poisson regression, *DTR* decision tree regression

programming, multilayer perceptron, and linear regression achieved better prediction accuracy. The results of measure of completeness and *pred(0.3)* analysis also confirmed the prediction capability of these techniques.

4.4 Significant metrics for the prediction of number of faults

In this section, we discuss important set of software metrics that can be used for the prediction of number of faults. We have analyzed the results of the linear regression analysis and identified the set of software metrics that can be used for building the fault prediction models for the prediction of number of faults. Table 8 shows the sets of significant metrics after applying the linear regression analysis on different eighteen datasets. It is clear from the table that out of twenty considered object-oriented metrics, 'lcom', 'wmc', 'loc', 'ca', and 'ce' are the metrics, which are identified as significant metrics for most of the datasets. Further, 'rfc', 'npm', 'dit, 'cbo' 'moa', and 'mfa' are the metrics, which are occurring as significant metrics for significant number of datasets.

Out of twenty-one source code metrics, 'Pre', 'NOM _sum', 'NSM _avg', 'ArrayInitializer', 'ArrayType', 'Java doc', 'DoStatement', 'QualifiedName', and 'SuperMethodInvocation' are the metrics, which are identified as significant metrics for most of the datasets having source code metrics. Further, 'NORM _ThrowStatement', 'LabeledStatement' and 'NORM _ConditionalExpression' are the metrics, which are occurring as significant metrics for the some of the datasets.

5 Comparative analysis

In this section, we perform a comparative analysis of the presented work with the existing works. In this study, we have evaluated the performance of six software fault prediction techniques. Out of six techniques, four techniques linear regression (LR), multilayer perceptron (MLP), negative binomial regression (NBR), and zero-inflated Poisson regression (ZIP) have been evaluated by the various researchers earlier, whereas the techniques genetics programming (GP) and decision tree regression (DTR) are evaluated

Table 8 Significant set of software metrics identified after linear regression analysis

Datasets	Significant metrics
Xerces 1.3	wmc, dit, cbo, rfc, lcom, ca, ce, npm, Lcom3, loc, moa, mfa, cam, ic, cbm, amc, and max_cc
Xerces 1.4	wmc, dit, rfc, lcom, ca, ce, npm , loc, dam, moa, cam, cbm, and max_cc
Camel 1.2	noc, lcom, ce, npm, loc, dam, moa, ic, cbm, max_, and avg_cc
Camel 1.4	wmc, dit, noc, cbo, rfc, lcom, ca, ce, npm, loc, moa, mfa, ic, and amc
Camel 1.6	wmc, dit, noc, rfc, lcom, ca, npm, lcom3, loc, dam, moa, cam, ic, and max_cc
Ant 1.7	wmc, rfc, lcom, ce, npm, loc, moa, amc, max_cc, and avg_cc
Xalan 2.4	wmc, dit, cbo, rfc, lcom, ca, ce, loc, moa, mfa, and cbm
Xalan 2.5	wmc, dit, rfc, lcom, ca, ce, npm, loc, and amc
Xalan 2.6	dit, cbo, rfc, lcom, ca, ce, npm, lcom3, loc, dam, mfa, cbm, max_cc, and avg_cc
Prop V4	wmc, cbo, lcom, ca, npm, loc, dam, cam, ic, cbm, and amc
Prop V40	wmc, cbo, lcom, ca, ce, npm, lcom3, loc, dam, moa, mfa, and cam
Prop V85	wmc, noc, cbo, lcom, ca, ce, lcom3, loc, dam, cam, ic, cbm, and amc
Prop V121	wmc, dit, rfc, lcom, ca, ce, npm, lcom3, loc, moa, and amc
Prop V157	wmc, cbo, rfc, lcom, ca, ce, loc, cam, and ic
Prop V185	wmc, dit, noc, cbo, rfc, lcom, ca, ce, npm, lcom3, loc, mfa, cam, and avg_cc
AS_ Eclipse 2.0	pre, NOM_ sum, NSM_ avg, ArrayType, CharacterLiteral, ConditionalExpression, DoStatement, FieldAccess, Javadoc, LabeledStatement, QualifiedName, ReturnStatement, SuperMethodInvocation, SwitchStatement, ThisExpression, NORM_ CharacterLiteral, NORM_ CompilationUnit, NORM_ ConditionalExpression, NORM_ NumberLiteral, NORM_ SwitchStatement, and NORM_ ThrowStatement
AS_ Eclipse 2.1	pre, NOM_ sum, ArrayCreation, ArrayInitializer, ArrayType, DoStatement, Javadoc, LabeledStatement, QualifiedName, SuperMethodInvocation, ThrowStatement, NORM_ CharacterLiteral, NORM_ ConditionalExpression, NORM_ NullLiteral, NORM_ SynchronizedStatement, NORM_ ThrowStatement, and NORM_ Modifier
AS_ Eclipse 3.0	pre, NSM_ avg, ArrayInitializer, DoStatement, Javadoc, MethodDeclaration, NullLiteral, QualifiedName, SingleVariableDeclaration, StringLiteral, SuperMethodInvocation, WhileStatement, InstanceofExpression, NORM_ ConditionalExpression, NORM_ Initializer, NORM_ SwitchCase, and NORM_ TypeLiteral

by few researchers only for the prediction of number of faults and need further evaluation on the more number of datasets. In this work, along with the four known fault prediction techniques (LR, MLP, NBR, and ZIP), we have also explored the capabilities of GP and DTR for the prediction of number of faults. There are some of the similar works available in the literature related to the empirical analysis of some fault prediction models. Table 9 summaries these available works.

It can be seen from the table that most of the earlier studies reported the contextual information of the fault prediction model along with the other fault prediction model building information, such as detail of the techniques used and calibration of the fault prediction model. Only a couple of studies provided the information regarding the significant set of metrics for the prediction of number of faults. In this work, we have provided the information regarding the significant set of metrics and provided a comparative analysis of the experimental results with the existing works. Further, it can be observed that most of the available works have considered only few fault prediction techniques for empirical analysis.

In Table 10, the summary of the results reported in various available works is presented. The table shows only those studies, which have reported their results in terms of error rate measures and are shown.

6 Threats to validity

In this section, the possible threats that may affect the validity of our experimental investigation are presented.

Construct Validity We have evaluated the effectiveness of six fault prediction techniques in terms of error rate, *pred(0.3)*, and completeness measures. We have used eighteen software project datasets from the PROMISE data repository. However, other open-source as well commercial datasets are available, which can be used for evaluation. Further, some other evaluation measures such as precision, recall, and analysis of association between predicted and actual values can also be used.

Internal Validity In this paper, we have used six fault prediction techniques to build the fault prediction models. Some of the techniques such as GP and others may require the optimization of various control parameters. We have tried to optimize the control parameters best to our efforts. However,

Table 9 Summary of comparative analysis

Reported work	Aim of the study	Contextual information reported	Model building information reported	Information of significant software metrics	Fault prediction models analyzed
Scanniello et al. (2013)	Intra-release defect prediction using clustering approach	Yes	Yes	No	A clustering approach
Marinescu (2014)	Evaluation of genetic programming for changes and defects prediction	Yes	Yes	No	Genetic programming
Kpodjedo et al. (2009)	Evaluation of design metrics for defect prediction	Yes	Yes	Yes	Poisson regression
Bacchelli et al. (2010)	Evaluation of developer metrics for the prediction of defect-prone classes	Yes	Yes	Yes	Principal component and regression analysis
Chen and Yutao (2015)	Evaluation of regression techniques for predicting defect numbers	Yes	Yes	No	Linear regression, bayesian regression, support vector regression, nearest neighbor regression, decision tree regression and gradient boosting regression
Afzal et al. (2008)	Fault count prediction using genetic programming	Yes	Yes	No	Genetic programming
Gao and Khoshgoftaar (2007)	Study of count models for software quality estimation	Yes	Yes	No	Poisson regression, negative binomial regression, zero-inflated poisson and negative binomial regression and hurdle regression
Graves et al. (2000)	Number of faults prediction using generalized linear regression	Yes	Yes	No	Generalized linear regression
Our Work	Study of six fault prediction techniques for the number of faults prediction	Yes	Yes	Yes	Linear regression, multilayer perceptron, decision tree regression, genetic programming, negative binomial regression and zero-inflated poisson regression

with the use of different fault datasets and with different set of software metrics, control parameter values may vary.

External Validity We have performed the experimental investigation over the datasets available in public data repositories. The system developed in the organization may possess different fault pattern. One needs to take care of the underlying pattern of software system before applying any fault prediction technique.

7 Conclusions and future work

This paper evaluated and compared the performance of six fault prediction techniques for the prediction of number of faults in given software modules. We have used four known fault prediction techniques, i.e., LR, MLP, NBR, and ZIP for the prediction of number of faults. In addition, we have investigated two techniques, i.e., GP and DTR, which until now have not been fully explored for the prediction of number of faults. The experiments were performed for eighteen software project datasets available publicly. AAE, ARE, measure of completeness, and prediction at *level l* measures have been used to evaluate the results of fault prediction models. In addition, Kruskal–Wallis test and Dunn's multiple comparison test were performed to assess the relative performance of the used fault prediction techniques. The results found that among the used different fault prediction techniques, decision tree regression, genetic programming, multilayer perceptron, and linear regression demonstrated better prediction performance for all the datasets under consideration. The analysis of results obtained from Kruskal–Wallis test and Dunn's multiple comparison test suggested that except negative binomial regression (NBR) and zero-inflated Poisson regression (ZIP) techniques all other techniques have performed significantly accurate for the prediction of number of faults. Generally, NBR and ZIP techniques produced

Table 10 Summary of results of related works

Evaluation measures	(Our work)						Khoshgoftaar and Gao (2007)	
	LR	MLP	DTR	GP	NBR	ZIP	PR	ZIP
AAE	0.11–1.7	0.13–2.2	0.12–1.68	0.107–2.57	0.57–1.23	0.25–1.37	1.55–1.75 (fit data) 1.67–1.99 (test data)	1.388–1.61 (fit data) 1.47–1.75 (test data)
ARE	0.06–0.61	0.7–0.68	0.06–0.56	0.7–1.03	0.50–1.15	0.16–1.1	0.68–0.762 (fit data) 0.73–0.76 (test data)	0.639–0.663 (fit data) 0.66–0.68 (test data)

Evaluation measures	Afzal et al. (2008)	Chen and Yutao (2015)						Scanniello et al. (2013)
	GP	LR	BRR	SVR	NNR	DTR	GBR	Clustering approach
AAE	–	–						0.0-0.33
ARE	0.16–0.0992	Within-project	0.4–2.7	0.35–2.6	0.4–2.5	0.5–2.3	0.4–2.9	0.4–2.4	–
		Cross-project	0.4–2.0	0.4–2.1	0.4–2.3	0.4–2.0	0.3–2.2	0.3–2.1	

the worst prediction accuracy. Thus, it is observed that the count models (NBR and ZIP) generally underperformed as compared to other considered fault prediction techniques. In the future, it may be tried to investigate and evaluate ensemble of these methods for the number of faults prediction to overcome the limitations of individual techniques.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

Informed consent This article does not contain any studies with human participants.

References

Afzal W, Torkar R, Feldt R (2008) prediction of fault count data using genetic programming. In: IEEE International conference on Multitopic, INMIC'08, pp 349–356

Bacchelli A, DAmbros, M, Lanza M (2010) Are popular classes more defect prone?. In: Fundamental approaches to software engineering, Springer, pp 59–73

Basili V, Briand L, Melo W (1993) Object-oriented metrics that predict maintainability. J Syst Soft 23(2):111–122

Bland JM, Altman DG (1995) Multiple significance tests: the bonferroni method. BMJ 310(6973):170

Briand L, Jurgen W (2002) Empirical studies of quality models in object-oriented systems. Adv Comput J 56:97–166

Cameron AC, Trivedi PK (2013) Regression analysis of count. Cambridge University Press, Cambridge

Casella G (2008) Statistical design. Springer, New York

Catal C (2011) Software fault prediction: a literature review and current trends. Expert Syst Appl J 38(4):4626–4636

Chen M, Yutao M (2015) An empirical study on predicting defect numbers. In: Proceedings of software engineering and knowledge engineering conference, SEKE'15, 2015, pp 397–402

Cohen J, Cohen P, West SG, Aiken LS (2002) Applied multiple regression and correlation analysis for the behavioral sciences, 3rd edn. Routledge, London

Conte SD, Dunsmore HE, Shen VY (1986) Software engineering metrics and models. Benjamin-Cummings Publishing Co. Inc, Redwood City

Draper NR, Smith H (1998) Applied regression analysis, 3rd edn. Wiley, Hoboken

Elish MO, Aljamaan H, Ahmad I (2015) Three empirical studies on predicting software maintainability using ensemble methods. Soft Comput J 19(9):1–14

Gao K, Khoshgoftaar TM (2007) A comprehensive empirical study of count models for software fault prediction. IEEE Trans Softw Eng 50(2):223–237

Goldberg DE (1989) Genetic algorithms in search optimization and machine learning, 1st edn. Addison-Wesley Longman Publishing Co.Inc, Boston

Graves T, Karr A, Marron J, Siy H (2000) Predicting fault incidence using software change history. IEEE Trans Softw Eng 26(7):653–661

Greene WH (2011) Econometric analysis. 7th edn. Pearson, New York

Hilbe JM (2012) Negative binomial regression, 2nd edn. Jet Propulsion Laboratory California Institute of Technology and Arizona State University, California

Janes A, Scotto M, Pedrycz W, Russo B, Stefanovic M, Succi G (2006) Identification of defect-prone classes in telecommunication software systems using design metrics. Inf Sci J 176(24):3711–3734

Jureczko M (2011) Significance of different software metrics in defect prediction. Softw Eng Int J 1(1):86–95

Juristo N, Moreno AM (2013) Basics of software engineering experimentation. Springer, New York

Khoshgoftaar T, Pandya A, More H (1992a) A neural network approach for predicting software development faults. In: Third international symposium on software reliability engineering, pp 83–89

Khoshgoftaar TM, Munson JC, Bhattacharya BB, Richardson GD (1992b) Predictive modeling techniques of software quality from software measures. IEEE Trans Softw Eng 18(11):979–987

Khoshgoftaar TM, Ganesan K, Allen BE, Ross DF, Munikoti R, Goel N, Nandi A (1997) Predicting fault-prone modules with case-based reasoning. In: Proceedings of the eighth international symposium on software reliability engineering, ISSRE '97. IEEE computer society

Khoshgoftaar TM, Gao K (2007) Count models for software quality estimation. IEEE Trans Reliab 56(2):212–222

Kohavi R et al (1995) A study of cross-validation and bootstrap for accuracy estimation and model selection. IJCAI 14:1137–1145

Kotsiantis SB (2007) Supervised machine learning: a review of classification techniques. In: Proceedings of the 2007 conference on emerging artificial intelligence applications in computer engineering: real word AI systems with applications in e health, HCI, Information Retrieval and Pervasive Technologies, The Netherlands, pp 3–24

Kpodjedo S, Ricca F, Antoniol G, Galinier P (2009) Evolution and search based metrics to improve defects prediction. In: 1st International symposium on search based software engineering, 2009, pp 23–32

Lambert D (1992) Zero-inflated poisson regression, with an application to defects in manufacturing. Technom J 34(1):1–14

Liguo Y (2012) Using negative binomial regression analysis to predict software faults: a study of apache ant. Inf Technol Comput Sci J 4(8):63–70

Marinescu C (2014) How good is genetic programming at predicting changes and defects?. In: 2014 16th International symposium on symbolic and numeric algorithms for scientific computing, IEEE, pp 544–548

Menzies T, Milton Z, Burak T, Cukic B, Jiang Y, Bener A (2010) Defect prediction from static code features: current results, limitations, new approaches. Autom Softw Eng J 17(4):375–407

Menzies T, Krishna R, Pryor D (2016) The promise repository of empirical software engineering data. North Carolina State University. http://openscience.us/repo

Ostrand TJ, Weyuker EJ, Bell RM (2004) Where the bugs are. In: Proceedings of 2004 international symposium on software testing and analysis, pp 86–96

Ostrand TJ, Weyuker EJ, Bell RM (2005a) Predicting the location and number of faults in large software systems. IEEE Trans Softw Eng 31(4):340–355

Ostrand TJ, Weyuker EJ, Bell RM (2005b) Predicting the location and number of faults in large software systems. IEEE Trans Softw Eng 31(4):340–355

Quinlan JR et al. (1992) Learning with continuous classes. In: 5th Australian joint conference on artificial intelligence, vol 92, pp 343–348

Rathore SS, Kumar S (2015a) Predicting number of faults in software system using genetic programming. In: 2015 International conference on soft computing and software engineering, pp 52–59

Rathore SS, Kumar S (2015b) Comparative analysis of neural network and genetic programming for number of software faults prediction. In: Presented in 2015 national conference on recent advances in electronics and computer engineering (RAECE'15) held at IIT Roorkee, India

Rathore SS, Kumar S (2016a) A decision tree logic based recommendation system to select software fault prediction techniques. Computing, 1–31. doi:10.1007/s00607-016-0489-6

Rathore SS, Kumar S (2016b) A decision tree regression based approach for the number of software faults prediction. ACM SIGSOFT Softw Eng Notes 41(1):1–6

Scanniello G, Gravino C, Marcus A, Menzies T (2013) Class level fault prediction using software clustering. In: 2013 IEEE/ACM 28th international conference on automated software engineering, IEEE, pp 640–645

Smith SF (1980) A learning system based on genetic adaptive algorithms. PhD thesis, Pittsburgh, PA, USA. AAI8112638

Strutz T (2011) Data fitting and uncertainty. Vieweg and Teubner Verlag Springer, New York

Venkata UB, Bastani BF, Yen IL (2006) A unified framework for defect data analysis using the mbr technique. In: Proceeding of 18th IEEE international conference on tools with artificial intelligence, ICTAI '06, 2006, pp 39–46

Veryard R (2014) The economics of information systems and software. Elsevier Science, Amsterdam

Wang S, Yao X (2013) Using class imbalance learning for software defect prediction. IEEE Trans Reliab 62(2):434–443

Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques. Morgan Kaufmann, Burlington