The Journal of Systems and Software 167 (2020) 110592

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Transformed *k*-nearest neighborhood output distance minimization for predicting the defect density of software projects

Cuauhtémoc López-Martín[a,*], Yenny Villuendas-Rey[b], Mohammad Azzeh[c], Ali Bou Nassif[d], Shadi Banitaan[e]

[a] *Department of Information Systems,Universidad de Guadalajara, Zapopan, Jalisco, México*
[b] *Centro de Innovación y Desarrollo Tecnológico en Cómputo, Instituto Politécnico Nacional, México*
[c] *Department of Software Engineering, Applied Science Private University, Amman, Jordan*
[d] *Department of Computer Engineering, University of Sharjah, Sharjah, UAE*
[e] *Department of Electrical and Computer Engineering and Computer Science, University of Detroit Mercy, Detroit, MI, USA*

ARTICLE INFO

Article history:
Received 12 November 2019
Accepted 30 March 2020
Available online 28 April 2020

Keywords:
Software defect density prediction
Case-based reasoning
Transformed k-nearest neighborhood output distance minimization
Support vector regression
Neural networks
ISBSG

ABSTRACT

Background: Software defect prediction is one of the most important research topics in software engineering. An important product measure to determine the effectiveness of software processes is the defect density (DD). Cased-based reasoning (CBR) has been the prediction technique most widely applied in the software prediction field. The CBR involves *k*-nearest neighborhood for finding the number (*k*) of similar software projects selected to be involved in the prediction process.

Objective: To propose the application of a *transformed k-nearest neighborhood output distance minimization* (TkDM) algorithm to predict the DD of software projects to compare its prediction accuracy with those obtained from statistical regression, support vector regression, and neural networks.

Method: Data sets were obtained from the ISBSG release 2018. A leave-one-out cross validation method was performed. Absolute residual was used as the prediction accuracy criterion for models.

Results: Statistical significance tests among models showed that the TkDM had the best prediction accuracy than those ones from statistical regression, support vector regression, and neural networks.

Conclusions: A TkDM can be used for predicting the DD of new and enhanced software projects developed and coded in specific platforms and programming languages types.

1. Introduction

Software Engineering Process is a knowledge area of Software Engineering (SE) concerned with work activities accomplished by software engineers to develop, maintain, and operate software (Bourque and Fairley, 2014). An important product measure to determine the effectiveness of software processes is defect density (DD), which has been defined as the total number of defects divided by the size of the software (Shah et al., 2012). Software size has been mainly measured in either source lines of code or function points (FP) (Sheetz et al., 2009).

A defect has been defined as an "imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be either repaired or replaced.", it is caused by a person committing an error, and it is associated to other SE knowledge area termed Software Quality that refers to "desirable characteristics of software products, to the extent to which a particular software product possess those characteristics, and to processes, tools, and techniques used to achieve those characteristics" (Bourque and Fairley, 2014).

On the other hand, the Software Engineering Management knowledge area is the application of management activities such as planning, coordinating, measuring, monitoring, controlling, and reporting to ensure that software products and software engineering services are delivered efficiently, effectively, and to the benefit of stakeholders, whereas software project planning (SPP) addresses the activities undertaken to prepare for a successful software engineering project from the management perspective (Bourque and Fairley, 2014). Software prediction (SP*)* belongs to SPP. SP has been applied to variables related to software projects such as size, effort, duration or defects. The types of software projects can be classified as new or enhancement (ISBSG, 2018).

* Corresponding author.
E-mail address: banitash@udmercy.edu (S. Banitaan).

https://doi.org/10.1016/j.jss.2020.110592
0164-1212/© 2020 Elsevier Inc. All rights reserved.

Machine learning techniques have been applied in the SP field such as cased-based reasoning (CBR, also termed *analogy based estimation* (Bardsiri et al., 2014; Idri et al., 2015)), neural networks (NN), support vector regression (SVR), genetic programming, genetic algorithms, decision trees, and association rules (Afzal and Torkara, 2011; Wen, et al., 2012; Gautam and Singh, 2018;). As reference, in the software effort prediction field, CBR has been the technique reported as the most applied, whereas SVR and NN have been reported with the best prediction accuracy (Wen et al., 2012). A SVR is a type of support vector machine (SVM) having regression as purpose (García-Floriano et al., 2018.). CBR, NN and SVR are useful to learn from non-linear relationships (López-Martín, 2015; Vapnik, 1998; Cortes and Vapnik, 1995) and non-linear relationships are common between software metrics and software defects (Kumar, 2013). Systematic reviews on CBR studies showed that the NN and SVR performances have commonly been compared to that of CBR (Bardsiri et al., 2014; Idri et al., 2015).

Since the CBR has outperformed other prediction models, that the application of CBR is still limited, and that its application has been recommended for SP (Idri et al., 2015), in the present study, we propose the application of CBR for predicting the DD of software projects. CBR includes as components to the similarity function, the associated retrieval rules, and solution function (Bardsiri et al., 2014). We propose a novel transformation for retrieving the correct output, by minimizing the distance among the outputs of the *k*-nearest neighborhoods of the project we want to predict. In addition, we suggest a strategy to predict the correct output, and we introduce an inverse transformation as solution function. Our proposed algorithm is based on the concepts of transformation (Poularikas, 2010), and minimization (Walser, 1999). We have termed to our algorithm as *transformed k-nearest neighborhood output distance minimization* (TkDM).

The *k*-nearest neighborhood does not tend commonly to have good performances for regression purposes. Due to the differences in the outputs for regression, and to the difficulty of finding adequate distance functions, we assuming that applying mathematical transformations to the outputs may contribute to solving this drawback.

In CBR, the *k*-nearest neighborhood corresponds to the number of similar projects selected to be involved in the CBR prediction process. The search of those *k* software projects has been an important issue in software prediction field since the *k* value can significantly affect the prediction accuracy (Bardsiri et al., 2014). A systematic review on CBR revealed that 42% of the studies defined a fixed number for *k*, 37% of them determined a range to find the best number for *k*, whereas in 21% of the studies, a dynamic selection of *k* was performed (Bardsiri et al., 2014). In our study, the *k* value coincides to that type of studies which a range is determined and tried to find the best number for *k*.

In relating the CBR process (Bardsiri et al., 2014) to our study: (1) since DD differs between types of development (López-Martín, 2019), we involve separated historical data sets of new and enhancement software projects, (2) variables from them are selected from those data sets to be compared, and (2) we retrieve projects similar to *x* project, and predict the DD for that *x* project.

Software defect prediction models have been used to (1) identify software metrics, modules, classes, files or subsystems that are more likely to be defective (defect-prone) (Choudhary, et al., 2018; Malhotra, 2015), (2) predict the number of defects (Marinescu, 2014; Rathore and Kumar, 2017; Ostrand et al., 2005), and (3) predict DD (López-Martín et al., 2018; Verma and Kumar, 2017; Kumar et al., 2013; Mandhanet al., 2015; Yadav and Yadav 2015; Rahmani and Khazanchi, 2010; Kutlubay et al., 2007; Knab et al., 2006; Nagappan and Ball, 2005; Sherriff et al., 2005).

Regarding DD, we identified studies on (1) data analysis (Li et al., 2017; Tengeri et al., 2016; Yamashita et al., 2016; Nugroho and Chaudron, 2014; Faller and Zhu 2013; Garousi et al., 2013; Shah et al., 2012a; Shah et al., 2012b; Hasim and Rahman, 2011), (2) reduction (Verma and Kumar, 2014), and (3) definitions (Shah et al., 2013). As for DD prediction, the techniques applied have been support vector regression (López-Martín, et al., 2018), neural networks (Kumar et al., 2013; Kutlubay, et al., 2007), fuzzy logic (Kumar et al., 2013; Yadav and Yadav, 2015), decision trees (Kutlubay et al., 2007; Knab et al., 2006), and statistical regressions (Verma and Kumar, 2017; Mandhan et al., 2015; Rahmani and Khazanchi, 2010; Nagappan and Ball, 2005; Sherriff et al., 2005).

The contribution of our study is the application of the TkDM algorithm to predict the DD in new and enhancement projects using FP as the independent variable. The TkDM prediction accuracy is compared to those obtained from the application of two types of SVR (i.e., ε-SVR, and υ-SVR) and four types of NNs (i.e., multilayer backpropagation neural network (MLP), radial basis function neural network (RBFNN), general regression neural network (GRNN), and cascade correlation neural network (CCNN)). As for ε-SVR and υ-SVR, four types of kernels were used by type of SVR (i.e., linear, polynomial, radial basis function, and sigmoid). Moreover, since the prediction accuracy of a new proposed prediction model should at least outperform that of a statistical regression (Kitchenham and Mendes, 2009), the prediction accuracies obtained from the TkDM, ε-SVR, υ-SVR, MLP, RBFNN, GRNN, and CCNN models are also compared to that of a simple linear regression (SLR).

In the present study, the prediction models are applied to software projects obtained from the International Software Benchmarking Standards Group (ISBSG) Release 2018. The data sets are selected following the ISBSG guidelines by taking into account the quality data, type of development, development platform, and programming language type (ISBSG Guidelines, 2018).

A systematic review on CBR considered to the wide diversity of criteria used such as data sets, metrics, and evaluation methods, as the most important limitation to compare the prediction accuracy among CBR and other types of models (Bardsiri et al., 2014), thus, in the experiments of our study, we used the same data sets, dependent and independent variables, and validation method.

The hypothesis to be investigated in our research is the following:

H_1:The DD prediction accuracy with the TkDM is statistically better than the accuracy obtained by SVR, NN, and SLR when function points is used as the independent variable.

Section 2 presents studies where DD has been predicted. Section 3 describes the criteria to evaluate the prediction accuracy of prediction models. Section 4 describes the TkDM, ε-SVR, υ-SVR, MLP, RBFNN, GRNN, and CCNN used to predict DD. Section 5 presents the criteria followed to select the data sets of software projects. Section 6 details how the models are trained and tested. Section 7 includes the results obtained from the models, whereas Section 8 presents the discussion including our conclusions, limitations, and future work.

2. Related work

Ten studies related to DD prediction were identified. We classified them into three categories: (1) studies analyzing the relationship between metrics (i.e., independent variables) and DD by means of statistical methods such as coefficients of correlation and determination, as well as linear or multiple linear regression models, (2) studies comparing the prediction accuracy between two machine learning models, and (3) studies applying a machine learning model. They are described next.

2.1. Studies that analyze the correlation between DD and metrics

Verma and Kumar (2017) analyze the relationship between DD and five metrics obtained from 62 open source software projects. The metrics are size (lines of code), defects, number of developers, number of downloads, and commits. The authors formulate six hypotheses (1) software size has a negative relationship with DD, (2) number of defects has a positive relationship with DD, (3) more number of developers has lower DD, (4) more downloads increase the DD, (5) number of commits has a positive relationship with DD, and (6) software size, number of defects, number of developers, number of downloads and number of commit are jointly related to DD. Their analysis is based on statistically significance of the coefficient of determination. As conclusions, the first, second, third and sixth hypotheses were accepted.

Mandhan et al. (2015) analyze the relationship between DD and seven class level metrics obtained from 145 software projects. These metrics are coupling, depth, cohesion, response, weighted methods, comments, and lines of code. Their analysis is based on statistically significance of the coefficient of determination. They conclude that there is a significant level of acceptance for DD prediction with these static metrics individually and jointly.

Rahmani and Khazanchi (2010) analyze the relationship between DD and three metrics obtained from 44 open source software projects. The metrics are software size, number of downloads, and number of developers. The authors formulate four hypotheses (1) software size is positively correlated with DD, (2) number of developers are positively correlated with DD, (3) the number of downloads is positively correlated with DD, and (4) software size, and number of developers jointly have a relationship with DD. Their analysis is based on statistically significance of the coefficient of determination. As conclusions, the second and fourth hypotheses were accepted.

Nagappan and Ball (2005) propose eight relative code churn metrics. Data were obtained from 2465 binaries compiled from 96,189 files (in some cases, one binary consists of more than one file). Data were obtained from a large project. A hold-out was used: 1645 and 820 binaries were used to train and test, respectively. Their analysis is based on Pearson and Spearman correlation coefficients by correlating actual versus predicted DD. Results show all correlations positive and statistically significant. They conclude that their proposed measures can be used as efficient predictors of DD.

Sherriff et al. (2005) propose five metrics. The validation method used was hold-out: 14 and 6 projects were used to train and test, respectively. They present a scatter plot showing actual versus predicted DD by project. They conclude that the resulting DD prediction is indicative of the actual system DD.

2.2. Studies that compare the prediction accuracy between two machine learning models

Kumar et al. (2013) apply fuzzy logic (FL) and MLP to predict the DD of subsequent product releases of two software projects. The three independent variables used are complexity, total lines of code, and pre-release defects. The accuracy criteria for comparing the prediction of models were mean absolute residual (MAR) and root mean square error (RMSE). They conclude that the MLP provides better results than the fuzzy logic system.

Kutlubay et al. (2007) classify modules obtained from nine NASA data sets as defective and defect-free, and then they selected the defective modules to apply RBFNN and decision tree (DT) to predict DD. The validation method used was k-fold (k=10). The accuracy criterion for comparing the prediction of models was Root Mean Square Error (RMSE). They conclude that the accuracy of DT was statistically better than that of RBFNN for the nine data sets.

2.3. Studies that apply a machine learning model

Knab et al. (2006) use a DT to predict DD in source code files of seven releases of an open source web browser project. Sixteen metrics are analyzed. The validation method used was k-fold (k=10). They did not compare the DT prediction accuracy to other prediction model. They conclude that (1) metrics from the same release, and evolution data (such as the number of modification reports), are useful for DD prediction, and (2) lines of code, number of functions, and change couplings have little predictive power with regard to DD.

Yadav and Yadav (2015) apply a FL model for predicting DD at each phase of development life cycle of 20 projects from the top most reliability relevant metrics by design, coding, and testing phase. The accuracy criteria for comparing the prediction of models were Mean Magnitude of Relative Error (MMRE) and Balanced Mean Magnitude of Relative Error (BMMRE). They conclude that the predicted DD are found very near to the actual defects detected during testing.

López-Martín et al. (2018) apply two types of SVR termed ε-support vector regression (ε-*SVR*) and υ-support vector regression (υ-*SVR*) to a set of 21 software projects also used in the present study. The prediction accuracy of the mentioned SVRs was compared to that of statistical regression. Based on a statistical significance test, they conclude that the υ-SVR with polynomial kernel was better than that of statistical regression when new software projects were developed on mainframes and coded in programming languages of third generation.

After the analysis of the previous ten studies, we can identified that five of them analyze the correlation between DD and proposed metrics, two studies apply two machine learning models and compare their prediction accuracy to each other (one of them MLP and RBFNN, and the second one FL and MLP), and the remaining three studies, a machine learning model is applied by study (DT, FL, and SVR), and in two of these latter three studies, the prediction accuracy of each model is compared regarding either releases of a project, or development life cycle phase.

In our study, we propose the application of the TkDM algorithm to predict the DD of new and enhancement projects using FP as the independent variable. A leave-one-out cross validation method (LOOCV) is used to train and test the models, whereas the prediction accuracy for evaluate the performance of models is absolute residuals (AR).

3. Accuracy criteria

Absolute residuals (AR) has been recommended over others accuracy criteria such as magnitude of relative error (MRE) or the magnitude of error relative to the estimate (MER) since AR is a non-asymmetric measure (Shepperd and MacDonell, 2012). Thus, in our study we used the AR as criterion to evaluate the accuracy prediction of models. The equation AR is the following:

$$AR_i = |Actual\ DefectDensity_i - Predicted\ DefectDensity_i|$$

The AR are calculated for each observation i the DD of which are predicted. The aggregation of the AR over multiple observations (N) can be obtained by their mean (MAR) as shown as follows:

$$MAR = (1/N)\sum_{i=1}^{N} AR_i$$

The MdAR correspond to the median of the ARs. The accuracy of a prediction model is inversely proportional to MAR.

In addition to MAR, and MdAR, we used two accuracy measures: standardized accuracy (SA) and effect size (Δ). The SA examines whether the prediction model generates predictions better than random guessing, whereas the Δ is used to examine that the predictions are not produced by chance. The value of Δ is recommended to be larger or equal to 0.5. SA and Δ are calculated as follows (Shepperd and MacDonell, 2012):

$$SA = 1 - \frac{MAE}{MAE_{po}}$$

$$\Delta = \frac{MAE - \overline{MAE}_{po}}{SP_o}$$

4. Description of prediction models

4.1. Transformed k-nearest neighborhood output distance minimization

This algorithm is described from formulated assumptions, steps and based on a LOOCV.

Let X = set of FPs, Y = set of DDs, FP^t = a testing value for a project in X, and DD^t = that testing value in DD corresponding to FP^t, then:

Assumption 1: To near values in X, the corresponding values in Y are also near.

Step 1: To select a k positive integer whose value is smaller than the data size of X.

Step 2: To find in X those k-nearest neighborhood near to FP^t. These k-nearest neighborhood are termed as $x^i, 1 \le i \le k$, and their corresponding values in Y as $y^i, 1 \le i \le k$.

Assumption 2: Based on elementary mathematical operations**,** it is possible to apply a T transform to $y^i, 1 \le i \le k$ such that the maximum distance between pairs from the k transformed values $y^{T(i)}, 1 \le i \le k$ is smaller than a PV predetermined value.

Step 3: To apply the T transform to $y^i, 1 \le i \le k$, values for obtaining the k transformed values as follows:

$$y^{T(i)} = \left(a \cdot x^i\right)^{\left(\frac{b}{y^i}\right)}, \; 1 \le i \le k$$

Where a y b are parameters to be adjusted to PV.

Step 4: To find the a y b parameter values by means of a seeding algorithm such that they allow that the maximum distance between pairs of the k transformed values $y^{T(i)}, 1 \le i \le k$ is smaller than PV.

Assumption 3: The mean obtained from the k transformed values $y^{T(i)}, 1 \le i \le k$ is an acceptable estimation for the transformed value of DD^t, which corresponds to FP^t.

Step 5: To calculate the mean from those k transformed values $y^{T(i)}, 1 \le i \le k$ as follows:

$$mean\left(y^{T(i)}\right) = \frac{\sum_{i=1}^{k} y^{T(i)}}{k}$$

Step 6: Based on the expressions obtained from the steps 3 and 5, and setting fixed values for the a y b parameters, it is possible to obtain the P predicted value for DD^t from the values FP^t and $mean(y^{T(i)})$. This is achieved by means of elementary algebraic operations obtaining the inverse transform as follows:

$$mean\left(y^{T(i)}\right) = \left(a \cdot FP^t\right)^{\left(\frac{b}{P}\right)}$$

By finding P:

$$P = \frac{b \cdot log\left(a \cdot FP^t\right)}{\log\left(mean\left(y^{T(i)}\right)\right)}$$

Step 7: To calculate the absolute residual (AR) by applying the following expression:

$$AR = \left|P - DD^t\right|$$

Step 8: To calculate MAR from all ARs.

4.2. Support vector regression (SVR)

In SVM, the optimization problem implies to find the maximum margin separating a hyperplane by correctly classify the training points as possible. An SVM model represents this optimal hyperplane with support vectors. An SVR, a type of SVM that can be applicable to regression problems, is characterized by the use of kernels, sparse solution, and Vapnik-Chervonenkis control of the margin and the number of support vectors (Awad and Khanna, 2015).

An SVM has a set of slack variables $\mathcal{E}_i$, which correspond to the distance of the input vector x^i from the decision hyperplane. These slack variables are associated with a parameter C, where C>0, which is used to control the over-training (García-Floriano et al., 2018).

SVR is trained based on a symmetrical loss function (i.e., Vapnik's ε-insensitive), which equally penalizes high and low misestimates. This loss function corresponds to ε-insensitive of Vapnik, which a flexible tube of minimal radius is formed symmetrically around the estimated function, such that the absolute values of errors less than a certain threshold ε are ignored both above and below the estimate. It permits that points outside the tube are penalized, but those points within the tube, either above or below the function, receive no penalty (Awad and Khanna, 2015).

An SVR finds a function *f(x)* that has most ε deviation from the actually obtained target y^i for the training data x^i, and at the same time is as flat as possible (Ma and Guo, 2014).

There are two types of SVR named ε-support vector regression (ε*-SVR)* and υ-support vector regression (υ*-SVR*) (Chang and Lin, 2011; Smola and Schölkopf, 2004; Schölkopf et al., 2000). The ε*-insensitive loss function* is the base for ε*-SVR* prediction. It is minimized by the υ*-SVR*, which uses other υ parameter having as possible values the [0,1] interval. The number of support vectors is also controlled by the υ parameter, that is, the υ*-SVR* compresses data and generalizes the prediction error bounds. Thus, there are two parameters ε and C in ε*-SVR*, and υ*-SVR* has the two parameters υ and C.

The kernel functions for ε*-SVR* and υ*-SVR* are the following (Vapnik, 1998; Cortes and Vapnik, 1995):

- Linear: $K(x, z) = x \cdot y$, where x and z are data patterns
- Polynomial: $K(x, z) = (\gamma (x \cdot z) + c_0)^d$, where γ: slope parameter, c_0: trade-off between major terms and minor terms of the generated polynoms, d: polynom degree, and *x,z*: data patterns
- Radial basis function (RBF): $K(x, z) = \exp(-\gamma |x - z|^2)$, where γ controls the radial basis function spread, and x,z: data patterns.
- Sigmoid: $\mathrm{K(x, z) = tanh(}\gamma \mathrm{(x \cdot z) + c_0)}$, where γ controls the radial base function spread, $\mathrm{c_0}$: independent term, and x,z: data patterns

4.3. Multilayer backpropagation neural network (MLP)

A MLP is a traditional feedforward neural network (Werbos, 1974; Rumelhart et al., 1985; Reed and Marks, 1999). The structure of a MLP involves an input layer to perceive the signal, at least one hidden layer that represents the computational engine of the MLP, and an output layer to make a prediction. The nodes represent neurons with logistic activation. Fig. 1 shows an MLP network where neurons of one layer used as input for neurons on the next layer (Riedmiller and Lernen, 2014). After a network receives the input, the neurons' layers process the signal and the output layer generates the output. The signal is processed by neurons in parallel within the same layer. The signal flows from the input layer to the output layer.

The training phase for an MLP network works by adjusting the bias and weights parameters in order to maximize some measure of fit to the training data. During training, the network is process-

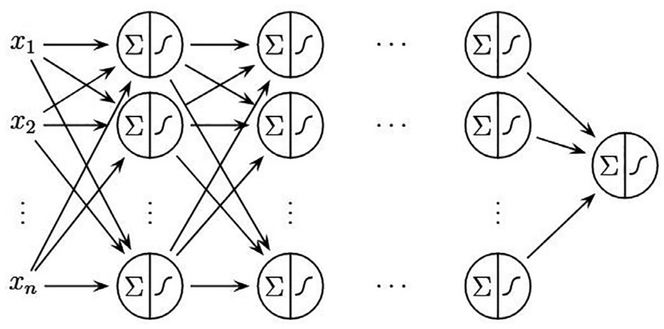

Fig. 1. Multilayer Perception Netwrok (Riedmiller and Lernen, 2014).

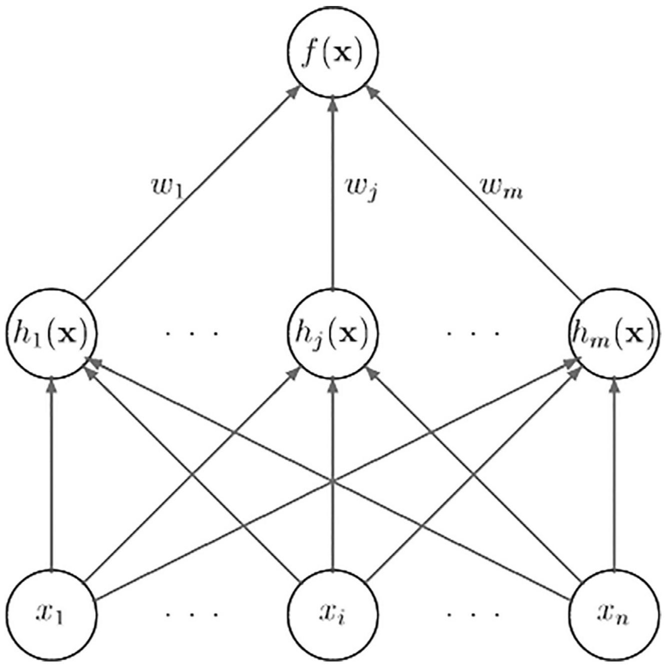

Fig. 2. The traditional RBFNN Netwrok (Møller, 1993).

ing the input and changing the parameters using learning algorithms through several iterations, in order to enhance its performance (Riedmiller and Lernen, 2014). Back-Propagation (BP) algorithm which is a variation of gradient decent learning is one of the most popular methods for MLP training (Rumelhart et al., 1985). The conjugate gradient algorithm is another algorithm which can be employed to train the network (Nassif et al., 2016; Møller, 1993).

4.4. Radial basis function neural network (RBFNN)

An RBFNN network is a feedforward artificial neural network whose structure contains an input layer, a hidden layer, and an output layer (Haykin, 1999). Fig. 2 shows the architecture of an RBFNN network (Orr, 1996). Each component of the input vector feeds to m basis functions in the hidden layer. The outputs of the basis functions are combined with weights into the output $f(x)$. An RBFNN network mostly employs the Gaussian function and it computes the distance from the center C_i to the input X. Each RBF function has a radius denoted by σ. Each neuron may have a different radius. The RBF function is shown in Eq. (1).

$$f(x) = \exp\left(-\frac{||X - C_i||^2}{2\,\sigma_i^2}\right) \tag{1}$$

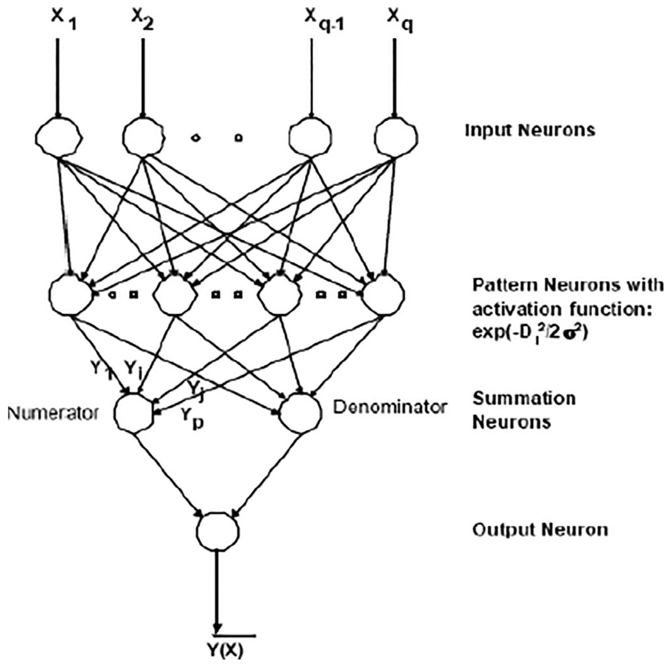

Fig. 3. The GRNN Netwrok (Galorath and Evans, 2006).

Where C_i represents center and σ_i represents the radius of the i^{th} neuron. The Euclidean distance is usually used as the distance between the input X and the center C_i. The RBFNN models are known to be less sensitive to noise and usually faster than other neural networks (Møller, 1993).

4.5. General regression neural network (GRNN)

A GRNN is a variation to RBFNN (Specht, 1991). It contains an input layer, two hidden layers, and an output layer. Fig. 3 shows the structure of GRNN. The first hidden layer takes input from the input neurons and is consisted of pattern neurons. Each neuron applies an RBF function to compute the distance between the input X and the center C_i. The output of the first hidden layer feeds into the second hidden layer (summation neurons). The second hidden layer is composed of the summation neurons namely the denominator and the numerator. The denominator summation adds the values of the weights fed from each of the first hidden layer neurons while the numerator adds the weights multiplied by the output values of the pattern neurons. The output neuron computes the predicted output value by dividing the numerator neuron's value by the denominator neuron's value (Møller, 1993).

4.6. Cascade correlation neural network (CCNN)

In a CCNN, the learning process begins with no hidden units. A connection with an adjustable weight is added between every input and every output units (Fahlman and Lebiere, 1990). The hidden units are added to the CCNN network one at a time in order to reduce the residual errors. Each new hidden unit obtains a connection from each input unit and from each hidden unit. The input weights for the hidden unit are frozen when the unit is added to the CCNN network. Each hidden unit adds one layer to the network; creating high-order feature detectors. The learning process repeats up until the error becomes small. The structure of CCNN is shown in Fig. 4. There is a bias input that is set to + 1. CCNN learns quickly, does not require backpropagation, and can build deep nets faster than deep backpropagation networks (Fahlman and Lebiere, 1990).

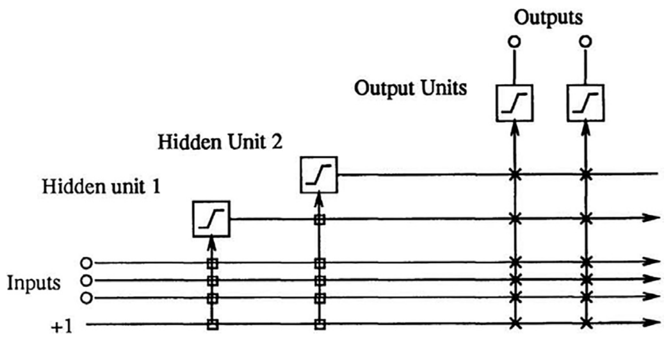

Fig. 4. The CCNN Netwrok with two hidden units (Fahlman and Lebiere, 1990).

5. Data sets of software projects

We searched available data related to our study. There are two main repositories commonly used in the software engineering field: PROMISE (PROMISE, 2019) and ISBSG (Fernández-Diego and González-Ladrón-de-Guevara, 2014). We analyzed the attributes of the PROMISE data sets with the objective of finding any attribute related to defects. In *CM1, JM1, KC1, KC2, PC1, DATATRIEVE*, and *Class-level data for KC1* data sets, a defect categorical attribute is reported (i.e. *false* or *true*); *Nickle, XFree86*, and *X.org* do not report any attribute related to defects; *MODIS, CM1*, and *QoS* are data sets related to software requirements rather than to defects; whereas *Desharnais, Cocomo81, Cocomo NASA*, and *COCOMO NASA 2* do not provide data regarding defects but of development effort.

As for ISBSG, it contains an attribute specifically related to DD by software project. In our study, FP is the independent variable used to predict DD. FP is obtained by calculating the internal logical files, and external interface files, as well as external inputs, external outputs, and external inquiries of the software project (López-Martín, 2015), whereas the dependent variable is DD, which is defined as defects by 1000 FP and calculated as follows (ISBSG Field Descriptions, 2018):

$$Total\ defects\ delivered * 1000/functional\ size$$

DD is defined as the number of defects by 1000 functional size units of delivered software in the first month of use of the software. It is expressed as *defects by 1000 function points*.

The software projects used in our study were obtained from the public ISBSG data set Release 2018. This release contains 8,261projects developed between the years 1989 and 2016. The data of these projects were submitted to the ISBSG from 26 different countries. Among them are United States, Spain, Australia, Japan, Netherlands, Finland, France, India, Canada and Denmark (ISBSG Demographics, 2018). The type of organization for which projects were developed are Communication, Insurance, Manufacturing, Government, Banking, Medical & health care, Financial, Electronics, Service industry, retail, and Professional services.

ISBSG release 2018 contains four types of developments for software projects: new, enhanced, re-development, and migration (ISBSG Field Descriptions, 2018). In accordance with the ISBSG Guidelines for use of the ISBSG data (ISBSG Guidelines, 2018), data sets should be selected taking into account their data quality rating, unadjusted function point rating, type of development, development platform, programming language type, as well as functional sizing method. Table 1 shows the number of projects as selected from the first three mentioned guidelines (A and B mean higher quality), and Table 2 according to the remaining guidelines. Pre-IFPUG V4 projects should not be mixed with V4 and post V4, and NESMA can only be mixed with IFPUG V4+ (ISBSG Guidelines, 2018).

Two of the final 140 new software projects of Table 2 were excluded because they had a defect density value lower than one (zero and 0.1 values). Table 3 classifies the software projects of Table 2 based on their develop platform (MF: mainframe, MR: midrange, Multi: multiplatform, PC: personal computer) and language type (3GL and 4GL: third and fourth generations, respectively, ApG: application generator).

In our study, with the goal of obtaining a better generalization from our conclusions based on statistical significance, those nine data sets from Table 3 higher than twenty projects were selected to be analyzed. A scatter plot (FP vs. DD) was generated by data set. The nine scatter plots showed skewness (i.e., there were fewer large projects than small projects), heteroscedasticity (i.e., the variability of DD decreased with FP), and outliers. Table 4 includes four statistical distribution tests commonly used to data normality applied for dependent and independent variables by data set (the Chi-squared test was not possible apply it to four datasets because it needs at least thirty data).

Since in Table 4 the smallest p-value amongst the tests performed is less than 0.01 for the eighteen cases, we can reject the idea that the variables come from a normal distribution with 99% confidence. Thus, the natural logarithm (*ln*) was applied to data of the nine datasets for normalizing them as suggested by Kitchenham and Mendes (2009). In Table 5, outliers were identified by means of studentized residuals greater than 2 in absolute value.

Regarding coefficient of correlation (*r*), tree data sets of Table 5 present a p-value higher than 0.05, thus, they did not presented statistically significant non-zero correlation at the 95% confidence level when outliers were removed. Two data sets had a higher percentage of outliers (one around the fifth part, and the second one corresponding to the half of the data set). Therefore,

Table 1
Software projects selected from the ISBSG Guidelines.

Atribute	Selected value(s)	Number of projects
Data quality rating	A and B	7,780
Unadjusted function point rating	A and B	6,429
Defect density	Not null	669
Type of development	New	211
	Enhanced	440
	Re-development	18

Table 2
Criteria for selecting software projects according their development platform (DP), language type (LT), and functional sizing methods (FSM).

Attribute	Selected value(s)	Number of projects		
		New	Enhanced	Re-development
DP	Not null	170	363	18
LT	Not null	167	361	18
FSM	IFPUG V4+ and NESMA	140	341	18

Table 3
Projects classified by development platform and language type (DP: development platform, LT: language type).

DP	LT	Number of projects		
		New	Enhanced	Re-development
MF	3GL	24	77	3
	4GL	8	12	2
	ApG	4	36	0
MR	3GL	11	45	0
	4GL	10	11	0
Multi	3GL	31	78	7
	4GL	13	33	2
PC	3GL	11	32	2
	4GL	26	15	2
	ApG	0	2	0
Total		138	341	18

Table 4
Normal statistical tests by data set (DP: development platform, LT: language type, NP: number of projects, FP: function points, DD: defect density).

Type of development	DP	LT	NP	Variable	Statistical test			
					Chi-squared	Shapiro-Wilk	Skewness	Kurtosis
New	MF	3GL	24	FP	–	0.0006	0.0223	0.0132
				DD	–	0.0017	0.0461	0.0606
	Multi	3GL	31	FP	0.0000	0.0000	0.0264	0.2732
				DD	0.0000	0.0000	0.0000	0.0000
	PC	4GL	26	FP	–	0.0000	0.0018	0.0005
				DD	–	0.0000	0.0002	0.0000
Enhancement	MF	3GL	77	FP	0.0000	0.0000	0.0016	0.0269
				DD	0.0000	0.0000	0.0004	0.0020
	MF	ApG	36	FP	0.0000	0.0000	0.0035	0.0015
				DD	0.0000	0.0000	0.0000	0.0000
	MR	3GL	45	FP	0.0203	0.0009	0.1471	0.5812
				DD	0.0000	0.0000	0.0000	0.0000
	Multi	3GL	78	FP	0.0009	0.0000	0.0001	0.0000
				DD	0.0000	0.0000	0.0000	0.0000
	Multi	4GL	33	FP	0.0005	0.0000	0.0137	0.0146
				DD	0.0075	0.0033	0.1707	0.6547
	PC	3GL	32	FP	0.0000	0.0000	0.0001	0.0000
				DD	0.0005	0.0000	0.0014	0.0002

Table 5
Outliers analysis by data set (DP: development platform, LT: language type, NP: number of projects).

Type of development	DP	LT	NP	Outliers		NP final	Statistics without outliers			Simple linear regression equation
				Number	Percentage		r	p-value r	r^2	
New	MF	3GL	24	3	0.13	21	-0.80	0.0000	0.64	$\ln(DD) = 6.42 - 0.79*\ln(FP)$
	Multi	3GL	31	13	0.42	18	-0.16	0.5150	0.02	–
	PC	4GL	26	8	0.31	18	-0.22	0.3782	0.04	–
Enhancement	MF	3GL	77	7	0.09	70	-0.83	0.0000	0.70	$\ln(DD) = 6.86 - 0.84*\ln(FP)$
	MF	ApG	36	4	0.11	32	-0.32	0.0690	0.10	–
	MR	3GL	45	3	0.07	42	-0.74	0.0000	0.55	$\ln(DD) = 8.18 - 1.07*\ln(FP)$
	Multi	3GL	78	15	0.19	63	-0.71	0.0000	0.51	–
	Multi	4GL	33	4	0.12	29	-0.77	0.0000	0.59	$\ln(DD) = 6.05 - 0.65*\ln(FP)$
	PC	3GL	32	16	0.50	16	-0.98	0.0000	0.97	–

in our study, a data set of 21 new projects, and three data sets of 70, 42 and 29 enhancement projects are used to train and test the DD prediction models. Table 5 includes four simple linear regression equations corresponding to these four data sets. The ANOVA p-value for these four equations was equal to 0.0000, that is, there is a statistically significant relationship between the variables at the 99% confidence level. As for coefficient of determination (r^2), it has values between 0.55 and 0.70, which means that the four SLRs as fitted explain between 55% and 70% of the variability in the DD.

Fig. 1 shows row data for that data set of Table 5 consisting of 24 projects (López-Martín et al., 2018), whereas Fig. 2 shows Fig. 1 normalized data (this data set had three outliers) (López-Martín et al., 2018). Figs. 3–5 depict the enhancement project data sets without outliers. All figures show that the higher the value of *FP*, the lower the *DD* is in the first month of use of the software. This pattern coincides with that reported in studies on DD (Shah et al., 2012a; Moeller and Paulish, 1993; Shen et al., 1985; Basili and Perricone, 1984; Gaffney 1984; Akiyama, 1971).

Therefore, in our study the following four data sets are used:

- Twenty-four new software projects developed in MF and coded in 3GL.
- Seventy software enhancement projects developed in MF and coded in 3GL.
- Forty-four software enhancement projects developed in MR and coded in 3GL.

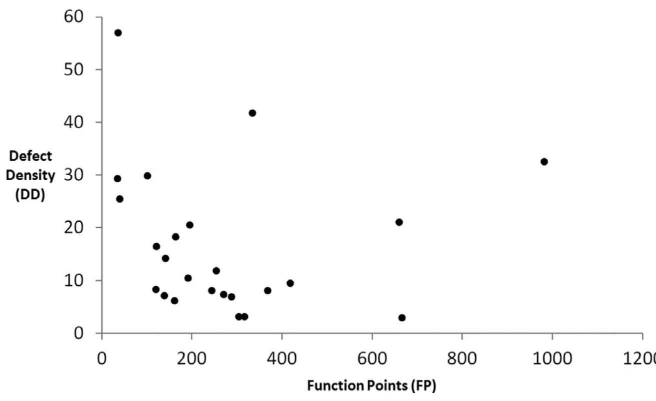

Fig. 5. Row data set of 24 new MF and 3GL projects.

- Forty-four software enhancement projects developed in Multi and coded in 4GL

6. Training and testing the models

Holdout, leave-one-out cross-validation (LOOCV), and k-fold cross-validation (k > 1) are the validation methods majorly used in the software prediction field when machine learning models have been applied (Wen et al., 2012). Since LOOCV is a determinis-

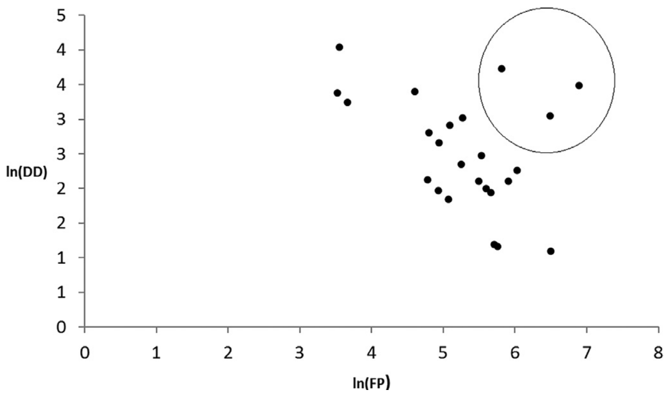

Fig. 6. Normalized data set from Fig. 1 (outliers enclosed in a circle).

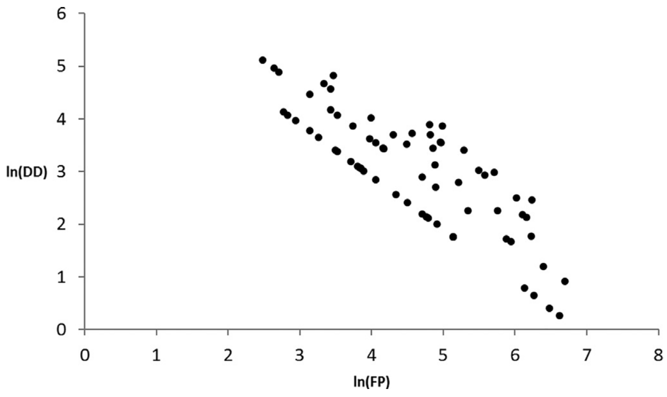

Fig. 7. Normalized data set for 70 MF and 3GL enhancement projects.

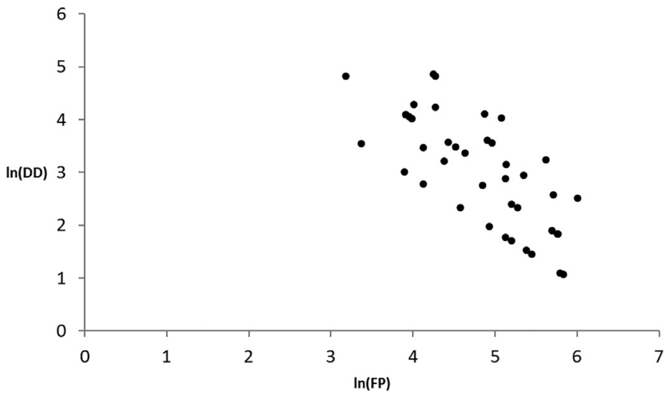

Fig. 8. Normalized data set for 42 MR and 3GL enhancement projects.

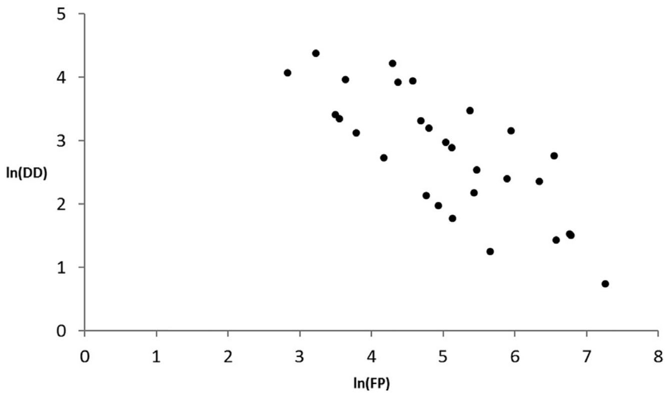

Fig. 9. Normalized data set for 29 Muti and 4GL enhancement projects.

Table 6
υ-SVR parameter values by type of support vector regression.

Parameter	Data set size			
	21	70	42	29
Kernel Type	Polynomial	Polynomial	Linear	Polynomial
γ	0.2	0.0	–	0.0
coef0	0	0.08	–	0.0
Degree	2	2	–	2
C	1	1	0.85	1
υ	0.09	0.4	0.9	0.5

tic procedure which can be exactly reproduced by any other researcher with the same particular data set (Gautam and Singh, 2018), we used the LOOCV to train and test the DD prediction models.

The TkDM was trained and tested in accordance with the algorithm described in the Section 4.1 "*Transform k-nearest neighborhood unidimensional minimization*" of this study. We experimentally found that the number of $k = 6$ was that value having best results when predicted DD for the four data sets.

As for statistical regressions, since data sets with 21, 70, 42 and 29 software projects were used, then 21, 70, 42 and 29 SLRs of the type $ln(DD) = a - b^{*}ln(FP)$ were generated. In the following subsections, the procedures to train and test the SVR, and the four types of NNs are described.

Regarding SVR, Table 6 shows the final parameters by type of SVR that generated the best accuracy in its data set. A *υ-SVR* had the best accuracy for the four data sets. A polynomial kernel having as expression $\gamma x_i x{+}coef0_{degree})$ had the best accuracy in three cases, and a linear kernel in the fourth one. If Coefficient coef0 is equal to zero, then the kernel is homogeneous. The shrinking heuristic had "yes" as value for the four cases, it means that it saves training time by identifying and removing some bounded elements (like C or $\alpha_i = 0$), which leads to a smaller optimization problem (Fan et al., 2005).

In accordance with neural networks, in the MLP one hidden layer with one hidden neuron generated the best accuracy in all the datasets. The activation function used in the hidden layer was logistic, whereas a linear function was used in the output neuron. The conjugate gradient algorithm was used in training the model.

As for RBFNN, two hidden neurons were used for the data set with size equal to 21, where six, four and three hidden neurons were used for data sets with 70, 42 and 29 projects, respectively.

Regarding GRNN, the sigma value for the predictor for the data set with size equal to 21 was 0.83, whereas for the predictor in data sets having 70, 42 and 29 projects were 0.14, 0.48 and 0.38, respectively.

In the CCNN, the number of neurons in the input, hidden and output layers were 1, 0, and 1, respectively for data sets with 21 and 29 projects, whereas the number of neurons in the input, hidden and output layers were 1, 1, and 1, respectively for data sets with 70 and 42 projects.

7. Results

The MAR and MdAR obtained by model are shown in Table 7. Since conclusions related to software prediction should be based on statistically significance (Kitchenham and Mendes, 2009), a suitable statistical test is selected based on the number of data sets to be compared, data dependence, and data distribution (Ross, 2004). In our study, the ARs of two data sets at the same time will be compared (one from TkDM and one from each other prediction model), data are dependent because DD by project was predicted applying all the models. As for data distribution, normality tests by a new data set obtained from the differences between ARs ob-

Table 7
Accuracy results for prediction models.

Data set	Model	MAR	MdAR	*SA*	Δ
21	TkDM	0.29	0.30	67.3	0.98
	SLR	0.42	0.49	52.9	0.85
	υ-SVR	0.36	0.40	60.0	0.97
	MLP	0.45	0.52	49.5	0.80
	RBFNN	0.49	0.52	44.5	0.72
	GRNN	0.47	0.39	47.3	0.76
	CCNN	0.45	0.40	49.7	0.80
70	TkDM	0.44	0.40	65.1	0.98
	SLR	0.54	0.55	56.9	0.88
	υ-SVR	0.49	0.52	61.0	0.94
	MLP	0.55	0.56	56.9	0.87
	RBFNN	0.53	0.50	58.3	0.89
	GRNN	0.54	0.48	57.2	0.87
	CCNN	0.56	0.51	55.6	0.85
42	TkDM	0.41	0.33	66.0	0.99
	SLR	0.59	0.53	51.2	0.81
	υ-SVR	0.57	0.60	53.1	0.84
	MLP	0.66	0.64	45.7	0.72
	RBFNN	1.03	0.61	12.7	0.25
	GRNN	0.63	0.52	48.4	0.76
	CCNN	0.63	0.48	48.0	0.76
29	TkDM	0.34	0.27	69.3	0.99
	SLR	0.55	0.45	51.8	0.84
	υ-SVR	0.51	0.49	55.3	0.90
	MLP	0.64	0.56	43.9	0.72
	RBFNN	0.79	0.61	30.9	0.50
	GRNN	0.60	0.48	49.6	0.78
	CCNN	0.54	0.54	49.4	0.80

Table 8
Normality distribution test results (p-values) by data set (ID: Insufficient data for performing statistical test).

Data set	Model compared to TkDM	Normality test			
		Chi-Squared	Shapiro-Wilk	Skewness	Kurtosis
21	SLR	ID	0.4130	0.3773	0.9037
	υ-*SVR*	ID	0.4751	0.6028	0.6934
	MLP	ID	0.3477	0.6625	0.3009
	RBFNN	ID	0.1639	0.8014	0.0457
	GRNN	ID	0.6455	0.5537	0.9620
	CCNN	ID	0.4582	0.4289	0.9293
70	SLR	0.2694	0.4982	0.6292	0.7923
	υ-*SVR*	0.4042	0.3348	0.9828	0.0925
	MLP	0.5646	0.4683	0.5680	0.9339
	RBFNN	0.0094	0.3624	0.6320	0.2746
	GRNN	0.0997	0.2184	0.6244	0.4384
	CCNN	0.3675	0.2203	0.2072	0.1986
42	SLR	0.0411	0.4407	0.8632	0.5595
	υ-*SVR*	0.0642	0.5637	0.8416	0.7548
	MLP	0.3463	0.0200	0.0958	0.3596
	RBFNN	0.0000	0.0000	0.0000	0.0000
	GRNN	0.0203	0.0596	0.3093	0.9554
	CCNN	0.0642	0.0333	0.1569	0.2531
29	SLR	ID	0.3654	0.3852	0.3788
	υ-*SVR*	ID	0.7436	0.9581	0.6403
	MLP	ID	0.9115	0.9699	0.8439
	RBFNN	ID	0.0000	0.0000	0.0000
	GRNN	ID	0.6197	0.5614	0.5778
	CCNN	ID	0.5475	0.4704	0.6080

tained between the two models compared at time are performed. Table 8 includes the results of normality statistical tests applied by new data set. If the smallest p-value amongst the tests performed for the data sets is less than 0.05, we can reject the idea that each data set of differences comes from a normal distribution with 95% confidence, and the comparison between the accuracy of the two models should be based on their medians (i.e., based on Wilcoxon test), otherwise on their means (i.e., based on *t-paired* test). Table 9 includes the p-values obtained by comparing SLR accuracy with that of each model.

Table 9
Statistical test results by comparing the SLR accuracy with accuracy of each model.

Data set	Model compared to TkDM	Statistical test	p-value
21	SLR	t-paired	0.0152
	υ-*SVR*	t-paired	0.0497
	MLP	t-paired	0.0079
	RBFNN	Wilcoxon	0.0091
	GRNN	t-paired	0.0173
	CCNN	t-paired	0.0378
70	SLR	t-paired	0.0568
	υ-*SVR*	t-paired	0.0499
	MLP	t-paired	0.0465
	RBFNN	Wilcoxon	0.1325
	GRNN	t-paired	0.0404
	CCNN	t-paired	0.0292
42	SLR	Wilcoxon	0.0172
	υ-*SVR*	t-paired	0.0241
	MLP	Wilcoxon	0.0003
	RBFNN	Wilcoxon	0.0029
	GRNN	Wilcoxon	0.0024
	CCNN	Wilcoxon	0.0000
29	SLR	t-paired	0.0042
	υ-*SVR*	t-paired	0.0223
	MLP	t-paired	0.0027
	RBFNN	Wilcoxon	0.0006
	GRNN	t-paired	0.0027
	CCNN	t-paired	0.0245

In addition, the results showed in Table 7 on SA and effect size show the meaningful of the prediction. In other words, it can display how better the model prediction rather than random guessing. The results of SA over all datasets confirm that not all models are predicting well. We can then notice that TkDM is the superior model amongst all models, over all datasets. Also, the MAR accuracy measure tells us that the TkDM is the best model with less average of absolute residuals.

8. Discussion

Software defect prediction is one of the most important research topics in software engineering (Li et al. 2018), and DD is a software quality attribute, which gives the reliability measure of the software product (Kumar et al., 2013). Therefore, we proposed the application of a transform k-nearest neighborhood unidimensional minimization (TkDM) to predict the DD of new and enhancement projects. The TkDM prediction accuracy based on AR, standardized accuracy (SA), and effect size (Δ) was compared to those obtained from two types of SVR termed (i.e. ε-*SVR* and υ-*SVR*, each type used linear, polynomial, radial basis function, and sigmoid kernels), and four types of neural networks (i.e., MLP, RBFNN, GRNN, and CCNN)

Since DD varies according to properties such as project type (Raja and Tretter, 2009), the data set of projects selected for our study observed the ISBSG Guidelines, which suggest the type of project as one of their criteria taking into account their data quality rating, unadjusted function point rating, functional sizing method, type of development, development platform, and programming language generation.

In accordance with Tables 7 and 9, the following hypotheses derived from that formulated in the Introduction section of this study, can be accepted at 95% confidence level when function points is used as the independent variable:

H_1:The DD prediction accuracy of new software projects developed in MF and coded in 3GL with TkDM is statistically better than the accuracies obtained by SLR, υ-*SVR,* MLP, RBFNN, GRNN and CCNN

H_2:The DD prediction accuracy of software enhancement projects developed in MF and coded in 3GL with TkDM is statistically better than the accuracies obtained by SLR, υ-*SVR,* MLP, RBFNN, GRNN and CCNN

H_3:The DD prediction accuracy of software enhancement projects developed in MidRange and coded in 3GL with TkDM is statistically better than the accuracies obtained by SLR, υ-*SVR,* MLP, RBFNN, GRNN and CCNN

H_4:The DD prediction accuracy of software enhancement projects developed in Multiplatform and coded in 4GL with TkDM is statistically better than the accuracies obtained by SLR, υ-*SVR,* MLP, RBFNN, GRNN and CCNN

We can then conclude that a TkDM can be used for predicting the DD in the first month of use of the software of new and enhancement projects developed in MF and coded in 3GL, as well as for enhancement projects developed in MidRange and coded in 3GL, and developed in Multiplatform and coded in 4GL.

In comparing our study with previous ones:

- In spite of the importance of DD prediction, we only identified ten studies related to this topic since 2005, and only three of them compare the prediction accuracy between machine learnig models (López-Martín et al., 2018; Kumar et al., 2013, Kutlubay et al., 2007).
- We did not find any study applying a TkDM to predict DD when using projects selected by following the guidelines suggested for the ISBSG.
- We identified two studies which applied NN, one of them a MLP was better than FL (Kumar et al., 2013), and the second one DT was better than RBFNN (Kutlubay et al., 2007), and one applying SVR in which a υ-SVR was better than a SLR (López-Martín et al., 2018).
- Researchers have approached their efforts in analyzing how DD changed with size (Koru et al., 2009). Figs. 2 to 5 of our study coincides with conclusions of previous studies in the sense that large projects exhibited lower DD than medium and small projects. This pattern may be explained because larger projects tend to be developed more carefully (Moeller and Paulish, 1993). A recent study published in 2018, reports that larger modules tend to have more defects but have a lower DD (Zhou et al., 2018).
- The value for k reported in a systematic review on CBR is between 1 and 23, and the majority of studies reported between one and five (Bardsiri et al. 2014). In our study, after applying several experiments, we found that six was a useful value for k for the four data sets

Regarding limitations of our study, although the last version of the ISBSG release 2018 consists of 8,261 projects, after we followed the criteria suggested by the ISBSG for selecting the data sets for new and enhancement projects, we could only use data sets of 21, 70, 42 and 29 projects to apply the models. A second limitation is related to the empirical manner for finding the parameters for SVR and NN, as well as for finding the *k* value for TkDM.

A validation threat of our study is related to the independent variable used (i.e., FP), which is also predicted, therefore, the TkDM prediction accuracy depends of the size prediction accuracy.

Since $k = 6$ was empirically found, future work could be related with finding it from an intelligent algorithm. Moreover, in accordance with step 4 described in Section 4.1 of this study, an empirically procedure was performed for finding the *a y b* parameter values, thus, future work could be related to the proposal of an intelligent algorithm such as a metaheuristic for finding those two mentioned parameters. Finally, future work could be related to the application of other types of transforms.

Declaration of Competing Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this manuscript.

Acknowledgement

The authors would like to thank the CUCEA, Universidad de Guadalajara, México, Centro de Innovación y Desarrollo Tecnológico en Cómputo, Instituto Politécnico Nacional, México, and the Consejo Nacional de Ciencia y Tecnología (CONACyT), México; the Applied Science Private University, Amman, Jordan; University of Sharjah, Sharjah, UAE; and the University of Detroit Mercy, Michigan, USA, for their support during the development of this work

References

Afzal, W., Torkara, R., 2011. On the application of genetic programming for software engineering predictive modeling: a systematic review. J. Expert Syst. Appl. 38, 11984–11997. doi:10.1016/j.eswa.2011.03.041.

Akiyama, F., 1971. An example of software system debugging. Inf. Process. 71, 353–379.

Awad, M., Khanna, R., 2015. Support Vector Regression, Efficient Learning Machines. Apress Berkeley, CA doi:10.1007/978-1-4302-5990-9_4.

Bardsiri, V.K., Jawawi, D.N.A., Khatibi, E. 2014. Towards improvement of analogy-based software development effort estimation: a review, Int. J. Softw. Eng. Knowl. Eng. (IJSEKE), World Scientific, 24 (7), 1065–1089. DOI: 10.1142/S0218194014500351.

Basili, V.R., Perricone, B.T., 1984. Software errors and complexity: an empirical investigation. Commun. ACM 27 (1), 42–52. doi:10.1145/69605.2085.

Bourque, P., Fairley, R. 2014. Guide to the Software Engineering Body of Knowledge, SWEBOK V3.0, IEEE Computer Society.

Chang, C.C., Lin, C.J., 2011. LIBSVM: A library for support vector machines. ACM Trans. Intell. Syst. Technol. 2 (3), 1–27. doi:10.1145/1961189.1961199.

Choudhary, G.R., Kumar, S., Kumar, K., Mishra, A., Catal, C. 2018. Empirical analysis of change metrics for software fault prediction, Comput. Electr. Eng., Elsevier, 67, 15-24. DOI: 10.1016/j.compeleceng.2018.02.043.

Cortes, C., Vapnik, V.N., 1995. Support-vector networks. Mach. Learn. 20, 273–297. doi:10.1023/A:1022627411411.

Fahlman, S.E., Lebiere, C., 1990. The cascade-Correlation Learning Architecture. In Advances in Neural Information Processing Systems. Morgan Kaufmann Publishers Inc.

Faller, D., Zhu, Y.., 2013. Defect-density assessment in evolutionary product development: a case study in medical imaging. IEEE Softw. 30, 81–87. doi:10.1109/MS.2012.111.

Fan, R.E., Chen, P.H., Lin, C.J., 2005. Working set selection using second order information for training support vector machines. J. Mach. Learn. Res. 6, 889–1918.

Fernández-Diego M, González-Ladrón-de-Guevara F (2014) Potential and limitations of the ISBSG dataset in enhancing software engineering research: a mapping review, Inf. Softw. Technol., Elsevier. 56(6): 527-544. DOI: 10.1016/j.infsof.2014.01.003

Gaffney, J.R., 1984. Estimating the number of faults in code. IEEE Trans. Softw. Eng. 10 (4). doi:10.1109/TSE.1984.5010260.

Galorath, D.D., Evans, M.W., 2006. Software Sizing, Estimation, and Risk Management. Auerbach Publications, Boston, MA, USA.

García-Floriano, A., López-Martín, C., Yáñez-Márquez, C., Abran, A., 2018. Support vector regression for predicting software enhancement effort. Inf. Softw. Technol. 97, 99–109. doi:10.1016/j.infsof.2018.01.003.

Garousi, V., Kotchorek, R., Smith, M., 2013. Test cost-effectiveness and defect density: a case study on the android platform. Adv. Comput. 89, 163–206. doi:10.1016/B978-0-12-408094-2.00005-9.

Gautam, S.S., Singh, V., 2018. The state-of-the-art in software development effort estimation. J. Softw. Evolut. Process 1-24, e1983. doi:10.1002/smr.1983.

Hasim, N., Rahman, A.A., 2011. Defect density: a review on the calculation of size program. In: Proceedings of the International Society for Optical Engineering, Fourth International Conference on Machine Vision (ICMV 11) doi:10.1117/12.920991.

Haykin, S., 1999. Neural Networks, a Comprehensive Foundation, Second Ed. Pearson.

Idri, A., Amazal, F.A., Abran, A., 2015. Analogy-based software development effort estimation: a systematic mapping and review. Inf. Softw. Technol. 58, 206–230. doi:10.1016/j.infsof.2014.07.013.

ISBSG Demographics. 2018. International Software Benchmarking Standards Group.

ISBSG Field Descriptions, 2018. International Software Benchmarking Standards Group.

ISBSG, Guidelines for use of the ISBSG data, Release 2018, 2018. International Software Benchmarking Standards Group.

Kitchenham, B., Mendes, E., 2009. Why comparative effort prediction studies may be invalid. In: Proceedings of the 5th International Conference on Predictor Models in Software Engineering, PROMISE doi:10.1145/1540438.1540444.

Knab, P., Pinzger, M., Bernstein, B., 2006. Predicting Defect Densities in source code files with Decision Tree Learners. In: Proceedings of the International Workshop on Mining software Repositories doi:10.1145/1137983.1138012.

Koru, A.G., Zhang, D., Emam, K.E., Liu, H., 2009. An investigation into the functional form of the size-defect relationship for software modules. IEEE Trans. Softw. Eng. 35 (2), 293–304. doi:10.1109/TSE.2008.90.

Kumar, V., Sharma, A., Kumar, R., 2013. Applying soft computing approaches to predict defect density in software product releases: an empirical study. Comput. Inf. 32, 203–224.

Kutlubay, O., Turhan, B., Bener, A.B., 2007. A Two-step model for defect density estimation. In: Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO) doi:10.1109/EUROMICRO.2007.13.

Li, Z., Jing, X.Y., Zhu, X., 2018. Progress on approaches to software defect prediction. IET Softw. 12 (3), 161–175. doi:10.1049/iet-sen.2017.0148.

Li, Z., Liang, P., Li, B., 2017. Relating alternate modifications to defect density in software development. In: Proceedings of the IEEE/ACM 39th IEEE International Conference on Software Engineering Companion doi:10.1109/ICSE-C.2017.132.

López-Martín, C., 2015. Predictive accuracy comparison between neural networks and statistical regression for development effort of software project. Appl. Soft Comput. 27, 434–449. doi:10.1016/j.asoc.2014.10.033.

López-Martín, C., Mohammad, A., Bou-Nassif, A., Banitaan, S., 2018. υ-SVR polynomial kernel for predicting the defect density in new software projects. In: Proceedings of the 17th IEEE International Conference on Machine Learning and Applications, pp. 1377–1382. doi:10.1109/ICMLA.2018.00224.

López-Martín, C., 2019. Software Defect Density Analysis. In: Proceedings of the 28th International Conference on Software Engineering and Data Engineering (SEDE 2019) doi:10.29007/rh9l.

Ma, Y., Guo, G., 2014. Support Vector Machines Applications. Springer.

Malhotra, R., 2015. A systematic review of machine learning techniques for software fault prediction. Appl. Soft Comput. 27, 504–518. doi:10.1016/j.asoc.2014.11.023.

Mandhan, N., Verma, D.K., Kumar, S., 2015. Analysis of approach for predicting software defect density using static metrics. In: Proceedings of the IEEE International Conference on Computing, Communication and Automation doi:10.1109/CCAA.2015.7148499.

Marinescu, C., 2014. How good is genetic programming at predicting changes and defects? In: Proceedings of the 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), pp. 544–548. doi:10.1109/SYNASC.2014.78.

Moeller, K.H., Paulish, D., 1993. An empirical investigation of software fault distribution. In: Proceedings of the IEEE First International Software Metrics Symposium doi:10.1109/METRIC.1993.263798.

Møller, M.F., 1993. A scaled conjugate gradient algorithm for fast supervised learning. Neural Netw. 6 (4), 525–533. doi:10.1016/S0893-6080(05)80056-5.

Nagappan, N., Ball, T. 2005. Use of relative code churn measures to predict system defect density, Proceedings of the 27th International Conference on Software Engineering. DOI: 10.1145/1062455.1062514

Nassif, A.B., Azzeh, M., Capretz, L.F., Ho, D., 2016. Neural network models for software development effort estimation: a comparative study. Neural Comput. Appl. 27 (8), 2369–2381. doi:10.1007/s00521-015-2127-1.

Nugroho, A., Chaudron, M.R.V., 2014. The impact of UML modeling on defect density and defect resolution time in a proprietary system. Empir. Softw. Eng. Springer 19, 926–954. doi:10.1007/s10664-013-9243-2.

Orr, M. J. 1996. Introduction to radial basis function networks.

Ostrand, T.J., Weyuker, E.J., Bell, R.M., 2005. Predicting the location and number of faults in large software systems. IEEE Trans. Softw. Eng. 31 (4), 340–355. doi:10.1109/TSE.2005.49.

PROMISE, 2019. http://promise.site.uottawa.ca/SERepository/datasets-page.html

Poularikas, A.D., 2010. Transforms and Applications Handbook. CRC press.

Rahmani, C., Khazanchi, D., 2010. A study on defect density of open source software. In: Proceedings of the 9th International Conference on Computer and Information Science doi:10.1109/ICIS.2010.11.

Raja, U., Tretter, M.J., 2009. Antecedents of open source software defects: A data mining approach to model formulation, validation and testing. Inf. Technol. Manag. 10 (4), 235–251. doi:10.1007/s10799-009-0062-5.

Rathore, S.S., Kumar, S., 2017. Towards an ensemble based system for predicting the number of software faults. Expert Syst. Appl. 82, 357–382. doi:10.1016/j.eswa.2017.04.014.

Reed, R., MarksII, R.J., 1999. Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks. Mit Press.

Riedmiller, M., Lernen, A.M., 2014. Multi Layer Perceptron. Machine Learning Lab Special Lecture. University of Freiburg.

Ross, S.M., 2004. Introduction to probability and statistics for engineers and scientists.

Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1985. Learning Internal Representations by Error Propagation (No. ICS-8506). California University of San Diego La Jolla Institute for Cognitive Science.

Schölkopf, B., Smola, A., Williamson, R.C., Bartlett, P.L., 2000. New support vector algorithms. Neural Comput. 12 (5), 1207–1245. doi:10.1162/089976600300015565.

Shah, S.M.A., Morisio, M., Torchiano, M., 2012a. An overview of software defect density: a scoping study. In: Proceedings of the IEEE 19th Asia-Pacific Software Engineering Conference doi:10.1109/APSEC.2012.93.

Shah, S.M.A., Morisio, M., Torchiano, M., 2012b. The impact of process maturity on defect density. In: Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement doi:10.1145/2372251.2372308.

Shah, S.M.A., Morisio, M., Torchiano, M., 2013. Software defect density variants: a proposal. In: Proceedings of the IEEE 4th International Workshop on Emerging Trends in Software Metrics (WETSoM) doi:10.1109/WETSoM.2013.6619337.

Sheetz, S.D., Henderson, D., Wallace, L., 2009. Understanding developer and manager perceptions of function points and source lines of code. J. Syst. Softw. 82, 1540–1549. doi:10.1016/j.jss.2009.04.038.

Shen, V.Y., Yu, T., Thebaut, S.M., Paulsen, L.R., 1985. Identifying error-prone software—an empirical study. IEEE Trans. Softw. Eng. 11 (4), 317–323. doi:10.1109/TSE.1985.232222.

Shepperd, M., MacDonell, S., 2012. Evaluating prediction systems in software project estimation. Inf. Softw. Technol. 54, 820–827. doi:10.1016/j.infsof.2011.12.008.

Sherriff, M., Nagappan, N., Williams, L., Vouk, M., 2005. Early estimation of defect density using an in-process haskell metrics model. ACM SIGSOFT Softw. Eng. Notes 30 (4). doi:10.1145/1082983.1083285.

Smola, C.J., Schölkopf, B., 2004. A tutorial on support vector regression. Stat. Comput. 14 (3), 199–222. doi:10.1023/B:STCO.0000035301.49549.88.

Specht, D.F., 1991. A general regression neural network. IEEE Trans. Neural Netw. 2 (6), 568–576. doi:10.1109/72.97934.

Tengeri, D., Vidács, L., Beszédes, A., Jász, J., 2016. Relating code coverage, mutation score and test suite reducibility to defect density. In: Proceedings of the IEEE International Conference on Software Testing, Verification and Validation Workshops doi:10.1109/ICSTW.2016.25.

Vapnik, V.N., 1998. Statistical Learning Theory. Wiley-Interscience.

Verma, D., Kumar, S. 2014. An Improved Approach for Reduction of Defect Density Using Optimal Module Sizes, Advances in Software Engineering, Hindwai Publishing Corporation, Article ID 803530. DOI: 10.1155/2014/803530

Verma, D., Kumar, S., 2017. Prediction of defect density for open source software using repository metrics. J. Web Eng. 16 (3-4), 93–310.

Walser, J.P., 1999. Integer Optimization by Local Search: A Domain-Independent Approach. Springer-Verlag.

Wen, J., Li, S., Lin, Z., Hu, Y., Huang, C., 2012. Systematic literature review of machine learning based software development effort estimation models. Inf. Softw. Technol. 54 (1), 41–59. doi:10.1016/j.infsof.2011.09.002.

Werbos, P. J. 1974. Beyond regression: New tools for prediction and analysis in the behavioral sciences. Ph. D. thesis, Harvard University, Cambridge, MA, 1974.

Yadav, H.B., Yadav, D.K., 2015. A fuzzy logic based approach for phase-wise software defects prediction using software metrics. Inf. Softw. Technol. 63, 44–57. doi:10.1016/j.infsof.2015.03.001.

Yamashita, K., Huang, C., Nagappan, M., Kamei, Y., Mockus, A., Hassan, A.E., Ubayashi, N., 2016. Thresholds for size and complexity metrics: a case study from the perspective of defect density. In: Proceedings of the IEEE International Conference on Software Quality, Reliability and Security doi:10.1109/QRS.2016.31.

Zhou, Y., Yang, Y., Lu, H., Chen, L., Li, Y., Zhao, Y., Qian, J., Xu, B., 2018. How far we have progressed in the journey? an examination of cross-project defect prediction. ACM Trans. Softw. Eng. Methodol. (TOSEM) 27 (1), 1–51. doi:10.1145/3183339.