Int J Syst Assur Eng Manag (July-Sept 2012) 3(3):194–208
DOI 10.1007/s13198-012-0123-8

ORIGINAL ARTICLE

An empirical study of software reliability prediction using machine learning techniques

Pradeep Kumar · Yogesh Singh

Received: 9 August 2011 / Revised: 3 June 2012 / Published online: 21 August 2012

Abstract The applications of machine learning techniques have shown remarkable improvements for the prediction of software reliability than traditional statistical techniques. In this paper, we apply some well-known machine learning methods such as artificial neural networks, support vector machines, cascade correlation neural network, decision trees and fuzzy inference system to predict the reliability of a software product. The proposed models have been evaluated using mean absolute error, root mean squared error, correlation coefficient and precision. The 16 software life cycle databases have been used for empirical studies. These databases are extracted from data and analysis center for software. A comparative analysis is performed in order to determine the importance of each method to assess the capability of software reliability prediction models. We also observe that these models may use in reliability predictions and results may be more close to the reality and precision is very effective with varied real-life failure datasets. Finally we conclude that proposed approach is more precise in its prediction capacity having better capability of generalization.

Keywords Software reliability · Artificial neural networks · Support vector machine · Decision trees · Machine learning techniques

P. Kumar (✉)
CS Department, IEC-CET, Greater Noida, India
e-mail: pksharma26@rediffmail.com

Y. Singh
M.S. University of Baroda, Baroda, India
e-mail: ys66@rediffmail.com

1 Introduction

Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment (Musa 1980). Software reliability modeling has gained a lot of importance in many critical and daily life applications, which has led to a tremendous work being carried out in the field of software reliability modeling. Software reliability growth models (SRGMs) successfully have been used for estimation and prediction of the number of errors remaining in the software (Goel and Okumoto 1979; Littlewood 1979; Musa 1980, 1998; Norman 1997; Lyu 2005; Kapur et al. 1999, 2010; Pham 2006). The software practitioners and potential users can assess the current and future reliability through software testing using these SRGMs. In past four decades, the classical models have remained one of the most attractive reliability growth models in monitoring and tracking reliability improvements (Musa 2005, 2007; Pham 2006). Classical models are the NHPP based models that have been widely applied successfully in many real-life applications for estimation and prediction of software reliability such as Musa-basic model, Musa–Okumoto model, Littlewood–Verral model, Goel–Okumoto model. Alternatively, some traditional statistical methods such as maximum likelihood estimation (MLE), least square estimation (LSE), analysis of variance (ANOVA), linear regression analysis (LRA) and logistic regression have also been applied for software reliability estimation and prediction (Kohavi 1995; Phillip 2003). The major challenges of these models do not lie in their technical soundness, but their validity and applicability in real world projects particularly in web-based systems. On the other hand, learning and generalization capabilities of artificial neural networks

(ANNs), and its proven successes in complex problem solutions has made it a viable alternative for predicting software failures during the testing phase (Karunanithi et al. 1992). The main advantage of ANN and other machine learning methods over NHPP based models is that it requires only past failure data as inputs, and less assumption required for modeling complex failure phenomena of software.

Machine learning is an approach concerned with the design and development of algorithms that allow computers to evolve the system behavior based on past and present failure data of software. Thus machine learning techniques are focused on learning automatically, recognizing complex patterns and making intelligent decisions based on past data. So that a machine is able to learn whenever it changes its structure, program, or data based on its inputs or in response to the external information in such a manner that it's expected future performance improves significantly (Kohavi 1995). Thus it is quite natural for software practitioners and researchers to know that which particular method tends to work well for a given failure dataset and up to what extent quantitatively (Aggarwal et al. 2006; Goel and Singh 2009; Singh and Kumar 2010a, b, c).

Here we conduct an empirical study of machine learning methods such as ANNs, SVMs, CCNN, DTs and fuzzy inference system (FIS) for the prediction of software reliability in order to draw stronger conclusions leading to widely accepted and well-formed theories. In this paper we briefly investigate and focus on three main issues: (i) How accurately and precisely do the machine learning based models predict the reliability of software product at any point of time during testing phase? (ii) Is the performance of SVMs and DTs better than CCNN and ANNs using back propagation network (BPN), radial basis function network (RBFN) and Elman network models? (iii) Correlate between SVMs and DTs for software reliability prediction since their performance varies when applied to past failure data in a realistic environment.

Rest of the paper is organized as follows: Sect. 2 describes the related work and in Sect. 3 we discuss software reliability prediction techniques. Section 4 elaborates about machine-learning methods and their performance analysis in detail. In Sect. 5 we discuss about empirical data collection and experimental set-up together with evaluation criteria. The experimental results and comparative performance of the machine learning methods are discussed in Sect. 6. Finally, the conclusions are drawn in Sect. 7.

2 Related work

Several machine learning techniques such as ANNs, SVMs, CCNN, DTs, Group Method of Data Handling (GMDH) Polynomial network, Gene Expression Programming (GEP), Genetic Programming (GP), FIS and Dynamic evolving neuro-fuzzy inference system (DENFIS) have been proposed in the literature for solving various classification and regression problems (Karunanithi et al. 1992; Kohavi 1995; Phillip 2003; Aggarwal et al. 2006, 2009; Jung Hua 2010; Ping and Hong 2006; Malhotra et al. 2009, 2011; Eduardo et al. 2010; Raj and Ravi 2008). There are few studies applied for the prediction of software reliability using machine learning methods based on past and present failure data of software. Some useful empirical studies based on multivariate linear regression and neural network methods have been carried out for prediction of software reliability growth trends. Although, multivariate linear regression method can address linear relationship but require large sample size and more independent variables (Jung Hua 2010). Many software reliability prediction models using ANNs have been applied for the prediction of software reliability successfully (Karunanithi et al. 1992; Singh and Kumar 2010c, d). However, effectiveness of neural network based prediction models depend on the behavior of dataset that is basically of fluctuating nature. Therefore ANNs suffers from overfitting the results while dealing with real-life unknown data sets. Overfitting occurs usually when the parameters of a model are tuned in such a way that the model fits the training data well but it has poor accuracy when applied on separate data not used for training.

The applications of SVM based machine learning approach in place of traditional statistical techniques has shown a remarkable improvement in the prediction of software reliability in the recent years (Xingguo and Yanhua 2007). SVM represents state of the art because of their generalization performance, ease of usability and rigorous theoretical foundations that practically can be used for modeling complex software failure behavior. The design of SVM is based on the extraction of a subset of the training data that serves as support vectors and therefore represents a stable characteristic of the data. The major limitation of SVMs is the increasing computational and storage requirement with respect to the number of training examples (Chen et al. 2008). Ping and Hong (2006) investigated the capabilities of SVMs for the prediction of software reliability with the help of simulated annealing algorithms (SA). In their study it is suggested that SVM model with simulated annealing algorithms (SVMSA) results in better predictions than other existing techniques in practice.

Yang and Xiang (2007) suggested an SVM-based model for software reliability prediction and pointed out that failure data collected from early phases of software development life cycle is more appropriate to be used which affect prediction accuracy. Xingguo and Yanhua (2007) investigated

the status of early prediction methods for software reliability by introducing SVM. They identified that early prediction model based on SVM is more accurate in its prediction with better capability of generalization. The main advantage of DTs are their descriptive nature, which allows practitioners to interpret the model's decision easily compared to other machine learning techniques such as ANNs and SVMs. While DTs do show their strengths in various real-life applications, these methods have rarely been used for predicting software reliability in practice.

The capability of fuzzy logic systems leads to the achievement of more efficient and decisive system in software reliability prediction. Due to large computation and low learning rate of prediction model, the machine learning techniques using FIS is found to be more effective than classical machine learning (Mueller and Lemke 1999). However, the present challenge is to make it even more efficient by incorporating a fairly new technique which can improve the prediction rate and require less computational resources.

3 Software reliability prediction using machine learning techniques

In this section, we explore some well-known and widely used machine learning methods (ANNs, SVMs, CCNN, DTs and FIS) for the prediction of software reliability based on past failure behaviour of software system. The ability of ANNs to model complex non-linear relationships and capability of approximating any measurable function make them attractive prospects for solving complex tasks without having to build an explicit model for the system. On the other hand, SVM is a learning system, which constructs an N-dimensional hyperplane that optimally separates the data set into two categories. The basic purpose of SVM modeling is to find the optimal hyperplane that separates clusters of vector in such a way that case with one category of the dependent variable on one side of the plane and the cases with the other category of the independent variable on the other side of the plane. The support vectors are the vectors near the hyperplane. Therefore SVMs can be used as an alternative training method for polynomial, radial basis function and multilayer perceptron networks using a kernel function. In SVMs, the weights of the network are found by solving a quadratic programming problem with linear constraints, rather than by solving a non-convex, unconstrained minimization problem as in case of training the neural network model. It is observed that SVM is capable to generalize well even in high dimensional spaces under small training sample conditions (Fei et al. 2005). Thus, SVMs have a better capability of generalization due to the structural risk minimization principle.

CCNN is a robust network model capable of producing better results with a small variation in the adjustment of parameters. Thus CCNN is capable of adjusting the number of hidden layers dynamically during the learning phase and hence can be generalized well for unknown failure dataset. DTs have been applied as a predictive model, which are capable of mapping observations of an item to the item's target value. In a tree structures, leaves represent the classifications and branches represent conjunctions of features that lead to those classifications. Therefore a DT can be used as a predictive machine learning model that can decide mean time to failure for the target value as a dependent variable of a new sample based on the attributes of independent variables. The internal nodes of a DT denote different attributes and the branches between nodes represents the possible values that these attributes can have in the observed samples. The objective of developing a software reliability prediction model using FIS is to make decisions about the software such as whether to release the system in its present state or continue testing. Thus, fuzzy logic can be utilized as a useful tool for decision making than a conventional intelligent system.

3.1 Research background

Our methodology for software reliability prediction using machine learning techniques is presented in detail (Sect. 4). The sources of data collection are discussed in Sect. 5. The Waikato Environment for Knowledge Analysis (Weka) is a comprehensive tool consisting of Java class libraries is used for implementing DTs. Weka is a freely open-source suite composed of several functions of the Weka class library that were used to conduct the experiments implemented on a Windows XP system with a Web browsing capability.

3.2 Dependent and independent variables

The dependent variable in our study is failure rate which is used to study the program failure rate per fault at the failure intervals. As the number of remaining faults change, the failure rate of the program changes accordingly. The failure rate of a system depends on time with the rate varying over life cycle of any software product. The dependent variable will be predicted based on the number of failures to be detected during testing phase after integrating all modules of software development life cycle. Testing time is the independent variable taken in terms of calendar time such as no. of weeks/days/Hrs/min./sec for the corresponding failures when detected and recorded.

4 Description of machine learning methods

The machine learning techniques deal with the issues of how to build and design computer programs that improve their performance for some specific task based on past observations. In this section we explore and discuss in detail some well-known and commonly used machine learning methods ANNs (BPN, RBFN and Elman network), SVM, CCNN, DTs and FIS for the prediction of software reliability as follows.

4.1 ANN modeling

Here we design a multilayer feed forward neural network referred to as M–H–Q network with M source nodes, H hidden nodes and Q nodes in the output layer (Aggarwal et al. 2009). The input nodes are connected to the output node through hidden layers of the network. The ANN repetitively adjusts different weights until the difference between estimated output and actual output from the network is minimized. The network gets trained by finding a vector of connection weights minimizing the sum of squared errors applied to all data sets. The input metrics are normalized using min–max normalization (Han and Kamber 2006). Min–max normalization performs a linear transformation on the original data. Suppose that minA and maxA are the minimum and maximum values of an attribute A then its mapping value v of A to v′ in the range of 0–1 is computed as follows:

$$v' = \frac{[v - \min A]}{[\max A - \min A]} \tag{1}$$

Thus mathematically, multilayer neural network architecture can be defined as follows:

$$net_k = b_k + \sum_{i=1}^{n} x_i w_{ki} \text{ and } y_k = f(net_k) \tag{2}$$

where n is number of input elements i.e., x_1, x_2, x_3,…x_n, w_{ki} is the set of connecting links associated with weights w_{k1}, w_{k2}, w_{k3},…w_{kn}, y_k is the output of previous layer of network and b_k is the bias which acts exactly as a weight on a connection from a unit when activation is always 1. The non-linear model of a neural network is shown in Fig. 1. The brief, description of various ANNs method applied for software reliability prediction described as: (i) Feed forward BPN, (ii) RBFN and (iii) Elman network is summarized as follows.

4.1.1 Architecture of different ANNs

The ANNs has been trained using standard error back propagation algorithm at a learning rate of 0.005 with a momentum of 0.60 having the minimum square error as the training-stopping criterion. The training procedure is carried out for entire failure datasets until network is able to provide desired responses. The input is feeded in terms of cumulative failures using input variable. The summary of the ANN architecture is shown in Table 1 consists of one hidden layer along with one unit in the output layer. The architecture of RBFN consists of three layers namely input layer, hidden layer and output layer. There exists n number of input neurons and m number of neurons with the hidden layer existing between the input and output layer. The interconnection between the input layer and hidden layer forms hypothetical connection, while the connections between hidden and output layer form weighted connections. The training algorithm is used for updating of weights in all the interconnections. Elman networks are two-layer back propagation networks, in addition to a feedback connection from the output of the hidden layer to its input. This feedback path allows Elman networks to recognize and generate temporal and spatial patterns. A simple recurrent network has activation feedback, which embodies short-term memory. A state layer is updated with the external input of the network together with the activation function from the previous forward propagation. This feedback is modified by a set of weights as to enable automatic adaptation through back propagation learning.

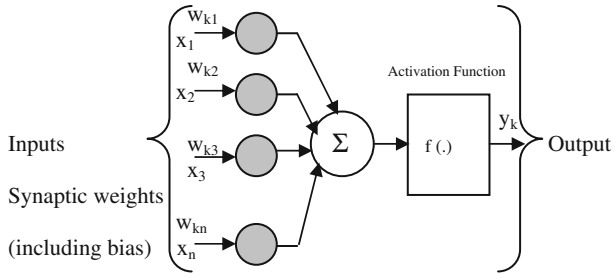

Fig. 1 Non-linear model of a neuron

4.1.2 Artificial neural network results

Here we present the result analysis of three ANNs (BPN, RBFN and Elman network) model applied for predicting the software reliability of software product. We have used MatLab version 7.0.1 on Windows XP platform to measure the performance of each network model using all sixteen failure datasets listed in Table 2. Tables 4, 5, and 6 presents the performance analysis of the model predicted using various parameters in terms of root mean square error (RMSE), R-value, slope of linear regression (LR), and *p* value. Out of all three ANNs presented here BPN outperforms both RBFN and Elman network model in terms of RMSE and slope of LR. However, the values of LR achieved through RBFN are better than other two methods

Table 1 Architecture of ANNs

Parameters	ANN structure		
	BPN	RBFN	Elman
Hidden layers	2	2	2
Input nodes	1	1	1
Output node	1	1	1
Learning rate	0.005	0.005	0.005
Momentum rate	0.6	0.6	0.6
Transfer function	Tansig	Tansig	Tansig
Training function	TrainBR	TrainBR	TrainBR
Training algorithm	Back-propagation	Back-propagation	Back-propagation

Table 2 Empirical data sets

Data sets	Software type	Number of errors detected
1	Real time command and control system	136
2	Real time command and control system	54
3	Real time command and control system	38
4	Real time command and control system	53
5	Real time commercial system	831
6	Commercial subsystem	73
14C	Real time system	36
17	Military system	38
27	Military system	41
40	Military system	101
SS1A	Operating system	112
SS1B	Operating system	375
SS1C	Operating system	277
SS2	Time sharing system	192
SS3	Word processing system	278
SS4	Operating system	196

whereas p values of Elman network are better indicators from the prediction point of view. The values of RMSE generated through RBFN and Elman is quite high perhaps due to the poor prediction performance of these two networks caused by fluctuating nature of data sets and less number of parameters used. Moreover, the values of correlation coefficients corresponding to BPN and RBFN have been very consistent between 80 and 90 % for both BPN and RBFN network models which is a positive outcome of our study but cannot be generalized due to inconsistent results of Elman network (Table 6).

4.2 Support vector machine modeling

SVM takes a set of input data and predicts for each given input, which of two possible classes the input is a member of, that makes SVM a non-probabilistic binary linear classifier. Thus, SVM is a learning system which constructs an N-dimensional hyperplane that optimally separates the data set into two categories. In SVM, a good separation is achieved by hyperplanes that has the largest distance to the nearest training data points of any class called functional margin. Ideally larger is the margin, lower the generalization error of classifier will be. The support vectors are the vectors near the hyperplane. The SVM modelling finds the oriented hyperplane so that the margin between the support vectors is maximized. SVM models are closely related to ANNs (Jung Hua 2010; Yang and Xiang 2007). Therefore SVMs can be used as an alternative training method for polynomial, radial basis function and multilayer perceptron networks using a kernel function (Phillip 2003). Summary of prediction measures using SVM model is presented in Table 7.

4.2.1 Mathematical model

The total number of software failures observed from several commercial and real-life projects is applied to the SVM model using cumulative number of detected software failures as inputs for predicting future failure behaviour of the software. Once the model has been trained and learnt the inherent internal property of the software failure process, it can be used for the prediction of software reliability realistically (Ping and Hong 2006). Here we consider n data points for training the SVM model defined as follows:

$$\{\{(x_1,\ y_1), (x_2,\ y_2), \ldots (x_n, y_n)\} \subset\ R^n\ \times\ R\} \tag{3}$$

where x_i is the input vector of dimension n, y_i is associated targeted value and n is the total number of observations. The linear regression function using Vapnik's $\in$-insensitive loss function (Vapnik 1995), can be written as:

$$f(x) = (w \cdot x\ ) + b,\ w\ \in\ R^n,\ b\ \in\ R \tag{4}$$

where w is the weight vector and b is the bias.

Thus, Vapnik regression function reflects the mapping of input and output of a process by learning a set of training data where the coefficients w and b can be determined by minimizing the regularized risk function as follows:

$$L_{\in}(y, f(x)) = \begin{bmatrix} 0,\ \text{when} |y - f(x)| \leq\ \in \text{and} \\ |y - f(x)| - \in,\ \text{for}\ |y - f(x)| \succ \in \end{bmatrix} \tag{5}$$

where $L_{\in}$is known as the $\in$- insensitive loss function, which is equivalent to the approximation accuracy required for the training (Yang and Xiang 2007; Jung Hua 2010; Fei

et al. 2005). Thus, regularized risk function can be written as:

$$R(W) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} L_{\in}[f(y_i, x_i)] \quad (6)$$

where C is the regularization constant that represents the trade-off between the approximation error and complexity of the model structure.

4.3 Cascade correlation neural network

Although, back-propagation neural network is the well-known and widely used multi-layer feed forward network. But, the disadvantage of multi-layer feed forward networks using error back propagation is that number of hidden layers and neurons in the network are problem specific. That is, it varies from task to task. Thus if large number of hidden neurons are used then the network will learn irrelevant details during the training and once trained it does not generalize well. Conversely, if a network is very small then it will not be able to learn from the training set properly. Therefore, we require such a network which could determine the size for a network automatically starting with a minimal network and then adding hidden neurons and connections as required. The CCNN is an alternative viable solution which helps in overcoming the shortcomings of BPN by adjusting the number of hidden layers dynamically during the learning phase (Scott and Christian 1991). Summary of prediction measures using CCNN model is presented in Table 8.

4.4 Decision tree modeling

The decision trees are useful to represent the information from machine learning algorithms. The DTs are composed of leaves and branches, where leaves represent the classifications and branches represent the conjunctions of features that lead to the classifications. Thus DTs are the decision-modeling tool that graphically displays the classification process of a given input for given output class labels. Decision tree can be applied to predict mean time to failures with the target value from the new samples based on the independent variables using past failures of a software system. The internal nodes of a DT denote different attributes, and the branches between nodes represent the possible values these attributes can have in the observed samples. DTs are one of the most popular classification algorithms applied in data mining and machine learning. The DT includes several algorithms such as Quinlan's ID3, C4.5, C5, J48 and CART (Ross 1993; Kohavi 1995; Witten and Frank 2011; Han and Kamber 2006). Here we discuss the capabilities of two commonly used algorithms (J48 and Naive Bayes) for decision tree induction. Further, in order to analyze the comparative performance of machine learning techniques used in our study an instant filter is applied to discretize the range of numeric attributes in the failure datasets into nominal attributes.

4.4.1 J48

J48 is a modified version of an earlier algorithm C4.5 developed by Quinlan (Ross 1993). In order to classify a new item using J48 we first create a decision tree based on the attribute values of available training data. So that whenever it encounters a set of items from the training set, it identifies the attribute that discriminates the various instances most clearly. J48 recursively classifies the tree until each leaf is pruned meaning that the data has been categorized as close to perfectly as possible (Witten and Frank 2011). In the process of DT, information gain that provides the most about data instances is classified as highest information gain. Moreover, among all possible values of information gain, if there is any other value for which there is no ambiguity then we terminate that branch and assign it to the target value that we have obtained so far. Alternatively, we look for another attribute providing highest information gain. Therefore we continue this process until we either get a clear decision of what combination of attributes provide us a particular target value, or we run out of attributes. In such event when we run out of attributes, or if we cannot get an unambiguous result from the available training set, then we assign this branch a target value that the majority of items under this branch possess. Now by checking all these respective attributes and their values observed in DT, we can predict the target value of new instances (Ross 1993).

4.4.2 Naive Bayes

The naive Bayes is a simple probabilistic classifier model which is based on the Bayesian theory. The naive Bayes classifiers can be trained very smoothly in a supervised learning dataset. The Naive Bayes algorithm is a classification algorithm based on Bayes rule that assumes the attributes of one category are all conditionally independent of one another from other category. Thus, Naive Bayes is a hybrid approach between decision-tree classifiers and Naive Bayes classifiers. The structure of Naive Bayes classifiers represents knowledge in the form of a tree which is constructed recursively. The leaf nodes are Naive Bayes categorizers for predicting a single class. We employed 10-fold cross-validation on all sixteen life-cycle failure datasets using Naive Bayes at the node to evaluate the utility of a node. The utility of the split is the weighted sum of utility of the nodes which depends on the number of instances that go through that node. The Naive Bayes

algorithm tries to approximate whether the generalization accuracy of Naive Bayes at each leaf is higher than a single Naive Bayes classifier at the current node (Kohavi 1995). Naive Bayes classifiers are generally easy to understand and the induction of these classifiers is extremely fast that require a single pass through the data. We found that Naive Bayes works well on real life failure datasets and thus it can be scaled up for accurate reliability prediction in real-life industrial projects. Summary of prediction measures is presented in Tables 9 and 10.

4.5 Fuzzy inference system (FIS)

Here we explore the capability of fuzzy inference system for the prediction of software reliability based on past and present failure behaviour of software. FIS is similar to a neural network type structure that is capable of mapping inputs through input membership functions and associated parameters. The parameters associated with the membership functions and corresponding associated output parameters are used to interpret the final output of the prediction model. The FIS structure was generated using genfis2 function from the Matlab Fuzzy logic Toolbox (2011). The basic steps of the FIS model are: (i) identification of input variables (cumulative failures and failure interval length) and output variable (failure time) (ii) development of fuzzy profile of the input/output variables (iii) defining relationships between input and output variables using fuzzy inference system. Thus FIS is capable of making decisions under uncertainty which can be applied for reliability prediction when applied to unknown failure datasets. We employed the reasoning capability of fuzzy logic which can be used as a technique of arriving at some concrete decision on whether to release the software in its present form or not. Table 10 presents the performance measure of FIS in terms of MAE, training RMSE and validated RMSE.

4.6 Training and validation method

We employed entire failure dataset shown in Table 2 by taking x_i as input (cumulative number of failures) detected during software testing time t_i to predict x_{i+1} as output during training and testing of each machine learning method presented in our study. Thus, past failure data is divided into two parts: training and testing data-set. The training data set is then applied to the proposed models for software reliability prediction and subsequently the parameters that lead to the best accuracies are selected. We apply k-cross validation, an alternative procedures that allows more of the data to be used for fitting and testing. Using k-cross validation, the entire dataset is randomly divided into k subsets (here $k = 10$) and each time one of the k subsets is used as the training data and other remaining (k-1) subsets are used to validate the prediction model for software reliability. Thus to maximize the utilization of failure datasets, cross-validation is an efficient method through repeatedly resampling the same data set randomly by reordering the dataset and then splitting up into ten folds of equal length (Kohavi 1995). The overall training and prediction process is illustrated through a flow diagram in Fig. 2.

5 Empirical data collection

In this paper we have used software failure data of various projects collected from Software Life Cycle Empirical/Experience Database (SLED) compiled and published by Data and Analysis Center for Software (Musa 1980). We assume that number of failures-detected is observed from the system-testing phase after confirmation of the integration of all modules and software components. The observation of failure and repair times is represented by $t_1, t_2, \ldots, t_n$ where t_i is the execution time of ith software failure assuming that each failure represents an independent sample from the same population. We can express $\Delta t_i = t_i - t_{i-1}$ as the time interval between the ith and $(i-1)$th failures while failure data is recorded in the form of (t_i, x_i) where x_i is the cumulative number of failures in the software at execution time t_i.

5.1 Experimental setup

Here we discuss the experimental set up to illustrate how machine learning methods have been used as an approximation tool for software reliability prediction realistically. The input to the network is number of cumulative failures while testing time (day of failure) is the output of the reliability prediction model. The standard error back propagation algorithm at a learning rate of 0.005 is used to train the network having the minimum square error as the training-stopping criterion with the help of neural network tool in MATLAB 7.01 for ANN models. SVM and CCNN have been implemented using DTReg predictive modeling software (Phillip 2003). DTs are implemented using the data mining package WEKA from the University of Waikato (Weka: classification and regression software 2010; Witten and Frank 2011). The classification tree based approach we have explored in our study are available in WEKA package. We have applied the mapping data to estimate the performance of each presented methods in our study using 10-fold cross validation to minimize the influence of training set variability.

5.2 Evaluation criteria

The performance of software reliability prediction model is measured in terms of several statistical parameters such as

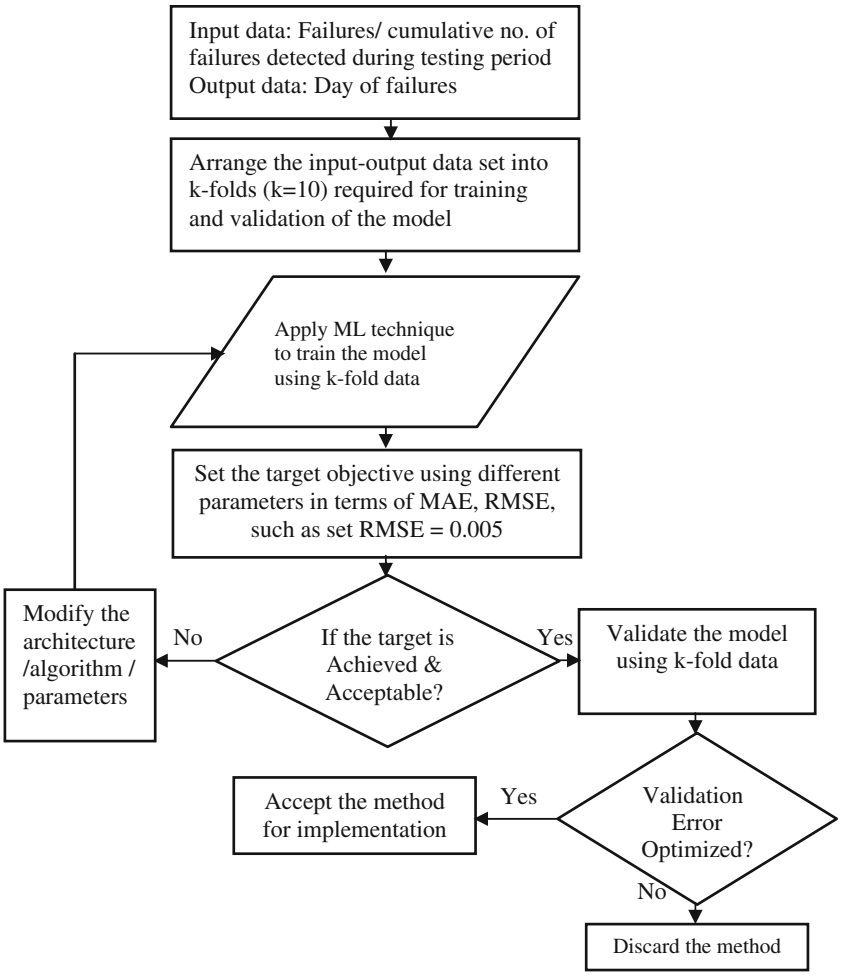

Fig. 2 Overview of software reliability prediction method using machine learning techniques

correlation coefficient between actual and predicted values, MAE, RMSE, and precision. The criteria for the goodness of-fit for the prediction of software reliability using various machine learning methods are described in Table 3 (Fawcett 2006; Hanley and McNeil 1982; Kapur et al. 2011; Lawson et al. 2003; Ohba 1984).

6 Results

In this section we present the summary of results for all 16 data sets using machine learning methods in terms of correlation coefficient, MAE, RMSE, *p* values and other parameters used in our study. The list of statistics shown in Tables 4, 5, 6, 7, 8, 9, and 10 summarize how accurately the machine learning methods are able to predict the software reliability of a product using 10-fold cross-validation. Figures 3, 4, 5, 6, and 7 are the graphical representation of performance analysis of machine learning methods (ANNs, SVMs, CCNN, DTs and FIS) for the prediction of software reliability.

6.1 Analysis results

Here, we discuss the analysis performed to find the relationship between ANNs and other machine learning methods (SVMs, CCNN, DTs and FIS) applied for software reliability prediction. The prediction capability of machine learning techniques are summarized and discussed in detail (Sect. 6.4).

6.2 Application in determination of software release instant

Quality of any software product mainly depends on how much time testing take place, what kind of testing methodologies are used, the amount of testing efforts put by testing team and complexity of software. More time developers spend on testing more errors can be removed leading to better reliable software. On the contrary, if testing time is short the software cost could be reduced but then customers may take a higher risk of buying unreliable software. However, it would

Table 3 Metrics used for model evaluation

Metrics	Definition
Correlation coefficient	Correlation coefficient measures the agreement of predictions with the actual class. This statistics shows that how closely actual and predicted values are correlated
MAE	Mean absolute error (MAE) is the quantity used to measure how close predictions are to the actual outcomes. MAE estimate the model's output for each observation to estimate whether the proposed model is biased and tend to over or under estimate. The predicted and observed values are the model's outcomes from n observations computed as follows: $MAE = \frac{1}{n}\sum_{i=1}^{n} \lvert\text{predicted} - \text{actual}\rvert$
RMSE	Root mean squared error (RMSE) is a frequently used measure of the difference between values predicted by a model and the values actually observed from the environment that has been modeled and computed as follows: $RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\text{predicted}-\text{actual})^2}{n}}$ where n is the number of observations for corresponding predicted and observed values of the model
Precision	Precision is defined as the number of classes that are predicted correctly, divided by the total number of classes
ROC curve	ROC curve defined as a plot of sensitivity on the y-coordinate versus its (1-specificity) on the x coordinate. It is an effective method of evaluating the performance of predicted models. Thus, Area Under the ROC Curve (AUC) is a combined measure of sensitivity and specificity
F-measure	F-measure is a way of combining recall and precision scores into a single measure of performance given as follows: F-measure = 2 × recall × precision/(recall + precision)
Sensitivity	Sensitivity also called recall rate, measures the proportion of actual positions which are correctly identified
Specificity	Specificity measures the proportion of negatives, which are correctly identified as the ratio of No. of true negatives to the (no. of true negatives + no. of false positives)
p value	p values are used for testing the hypothesis of no correlation. Each p value is the probability of getting a correlation as large as the observed value by random chance. If p is small, say less than 0.05, then the correlation R is significant
Slope in LR	In linear regression, we can't always draw a straight line that passes through every data-point, but we can find such a line that comes closer to most of the data points. Thus slope represents the estimated average change in Y (failure rate) when X (testing period) increases by one unit used in our study

increase the cost during maintenance phase since it is more expensive to fix an error during maintenance phase than during testing phase. Therefore it is essentially important to decide when to stop testing and release the software to customers based on cost and reliability criteria. The prediction of software reliability is a task where we try to predict the future failures trends of a software product using the present and past failure data. In our study, the failure data has been recorded in terms of number of failures, length of failure interval, and day of failures. The analyzed software reliability prediction results are given to the quality assurance and management team, who collectively decide appropriate time to release the software based on cost and reliability assessment meeting the requirements of customers, thereby leading to a better customer satisfaction management solution.

6.3 Observations

Based on the empirical results obtained from rigorous experiments conducted, some of the specific observations of our study are summarized as follows:

1. It is easy to design and construct models for software reliability growth of varying complexity for a given failure dataset using ANNs. However, the effectiveness of neural network based prediction models depend on the behavior of dataset that is basically of fluctuating nature. Therefore ANNs lack of generalization while dealing with real-life large data sets as shown in Tables 4, 5 and 6.
2. From Tables 4, 5, and 6, it can be easily depicted that ANNs using BPN, RBFN and Elman's network does not better fit in terms of p values and RMSE but able to generate encouraging results in terms of correlation coefficient and slope of liner regression.
3. Since ANN is able to approximate the continuous functions to a desired accuracy level through a multilayer perceptron. This implies that the approximated function of neural network based approach can also be employed in estimating cumulative failures observed by time t in software reliability modeling. However, ANNs model face the problem of over fitting the results and lack of learning algorithms from the dynamic environment which require larger sample size for training and testing the network together with more independent variables. Overfitting occurs usually

Table 4 Summary of predictions for different data sets using back propagation neural network

Data sets	Correlation coefficient	Slope of LR	*p* value	RMSE
1	0.9213	0.6561	0.0000	13.1466
2	0.8224	0.9681	0.0000	5.2801
3	0.9086	0.7502	0.0007	2.5072
4	0.8881	0.6616	0.0000	3.4367
5	0.9672	0.7486	0.0000	9.7447
6	0.9092	0.7687	0.0000	5.4920
14C	0.8845	0.8864	0.0001	2.8437
17	0.8611	0.7783	0.0002	3.0334
27	0.8928	0.6896	0.0000	2.9741
40	0.9214	1.0406	0.0000	5.4444
SS1A	0.9885	0.8737	0.0000	3.3393
SS1B	0.9710	0.8642	0.0000	7.5529
SS1C	0.9807	0.8848	0.0000	9.5419
SS2	0.9764	0.8698	0.0000	8.7359
SS3	0.9857	0.9986	0.0000	7.2242
SS4	0.9320	0.0176	0.1105	9.6954

Table 5 Summary of predictions for different data sets using radial basis function network

Data sets	Correlation coefficient	Slope of LR	*p* value	RMSE
1	0.7902	0.5389	0.0000	13.6042
2	0.8242	0.8985	0.0000	4.6012
3	0.9700	0.7958	0.0000	2.2353
4	0.9103	0.6141	0.0000	3.7371
5	0.9687	0.7775	0.0000	29.3936
6	0.9528	0.7693	0.0000	4.8882
14C	0.7705	0.7310	0.0034	2.9049
17	0.8654	0.8144	0.0001	2.9903
27	0.8654	0.5737	0.0000	3.7381
40	0.9541	1.0911	0.0000	4.8059
SS1A	0.9788	0.9318	0.0000	3.7925
SS1B	0.9647	0.9021	0.0000	19.8808
SS1C	0.9768	0.9158	0.0000	9.5273
SS2	0.9744	0.8257	0.0000	9.9873
SS3	0.9846	1.0077	0.0000	7.5430
SS4	0.9840	0.8957	0.0000	5.0622

when the parameters of a model are tuned in such a way that the model fits the training data well but it has poor accuracy when applied on separate unknown failure-data not used for training.

4. Based on the performance analysis achieved through SVMs, CCNN, DTs and FIS, it is quite encouraging to design SVM based models for software reliability growth of varying complexity for a given failure dataset.
5. The robustness and validity of machine-learning based models using SVM make it easier for real-world applications for predicting reliability accurately. Moreover, SVM is adaptive to the modeling nonlinear functional relationships, which are difficult to model with other machine learning methods such as CCNN and ANNs.
6. Machine-learning techniques such as SVM generalize well even in high dimensional spaces under small training datasets. Therefore, software reliability prediction models can be built much earlier with SVM than other conventional techniques with relatively good performance achieved and can be extensively applied in many fields of software reliability realistically.
7. The main advantage of modeling statistical software failure data is decision making about the software products. Thus, whether to release the system for deployment or continue further testing can be assessed through DTs. Therefore, machine learning methods such as SVM and DTs can be utilized as a powerful tool for reliability prediction than a conventional expert system while deciding the software release instance.
8. Comparing various statistics for the prediction of software reliability in terms of MAE and RMSE using various machine learning techniques, we found that accuracy of the SVM model is pretty close, suggesting that model will not break down with the unknown data also, or when future unseen failure data is applied to them.
9. Empirically, we conclude that SVM provides reliable performance and accurate results than other machine learning methods presented in our study. While, ANN is another viable alternative model which performs better in comparison with other machine learning techniques in practice. Moreover, CCNN is a supervised learning approach applied to create and install the hidden neurons for maximizing the magnitude of correlation between the existing and new neurons. This way, CCNN is capable of learning very quickly that could determine the size and topology automatically to minimize the residual error signals.
10. Thus machine learning methods presented in our study are able to model complex non-linear relationships and capable enough to approximate the desired measurable function. Therefore, machine learning is an attractive prospect for solving regression task without having to build an explicit model of the system.
11. Although, machine learning methods applied for software reliability prediction using DTs are rarely used for software reliability prediction in literature,

Table 6 Summary of predictions for different data sets using Elman network

Data sets	Correlation coefficient	Slope of LR	*p* value	RMSE
1	0.4871	0.0367	0.0006	12.8616
2	−0.6423	−0.0457	0.0013	6.7133
3	0.8459	0.2768	0.0002	3.3707
4	0.0922	0.0139	0.6758	7.3206
5	0.9817	0.3883	0.0000	12.2064
6	0.7591	0.1817	0.0000	8.3335
14C	0.9276	0.0240	0.0000	3.3898
17	0.7480	0.1368	0.0033	3.3374
27	0.8194	0.1984	0.0001	4.3665
40	0.5982	0.1534	0.0000	11.2035
SS1A	0.1102	0.0200	0.4765	13.1599
SS1B	−0.0733	−0.0124	0.3727	17.1219
SS1C	0.8190	0.0119	0.0000	30.6914
SS2	0.5156	0.1849	0.0000	24.7768
SS3	0.6311	0.0062	0.0000	31.0176
SS4	−0.0094	−0.0017	0.9359	22.5824

Table 7 Summary of predictions using SVM

Data sets	Evaluation parameters				
	Correlation coefficient	MAE	Training RMSE	Testing RMSE	No. of support vectors used
1	0.9979	1.9813	2.4943	2.6423	91
2	0.9965	0.9681	1.2991	2.5887	47
3	0.9877	1.3231	1.7212	2.5692	26
4	0.9882	1.9003	2.3417	2.5898	52
5	0.9995	5.2685	7.3104	13.8895	741
6	0.9928	1.9436	2.5205	3.6771	50
14C	0.9850	1.3036	1.8158	2.3397	35
17	0.9970	0.7425	0.8777	1.1929	15
27	0.9938	0.9821	1.3193	4.5056	23
40	0.9987	1.1498	1.4467	2.2218	54
SS1A	0.9988	1.1443	1.5311	1.9042	86
SS1B	0.9993	2.9369	3.8716	5.4320	313
SS1C	0.9987	3.0602	3.9734	5.1429	227
SS2	0.9987	2.2645	2.8062	3.1739	107
SS3	0.9992	2.4311	3.1103	3.3697	212
SS4	0.9989	2.0029	2.6230	3.1910	108

yet it is worthwhile and desirable to include these methods for discussion. However, to make a generalization of these methodologies, more similar studies need to be carried out by taking more parameters and updated failure data of real-life industrial projects.

6.4 Discussion

The performance measures of machine learning methods (ANNs, SVMs, CCNN, DTs and FIS) corresponding to RMSE, correlation coefficient, MAE, slope of LR, *p* values and other parameters used for estimation and prediction purposes can be summarized as follows.

The performance measure of RBFN and Elman networks in terms of RMSE, slope of LR and correlation coefficient values are higher than the corresponding values of the performance measures using BPN. However the performance measures of RBFN in terms of correlation coefficient and slope of LR is quiet encouraging which makes it better choice for prediction but lacks in generalization for larger data sets such as DS5, SS1B, SS1C and SS3. While the performance measures of SVM is the optimum choice for software reliability prediction in terms of MAE and training RMSE respectively. Alternatively, CCNN is another viable choice on the basis of training and testing RMSE. On the other hand, the accuracy of ANNs and FIS suffers from overfitting the results. However, ANNs using BPN and RBFN has been applied in many real-life applications but FIS performs very badly while validating the models. Although training results are fairly good but cannot be generalized due to less number of input–output membership functions and associated variables. Interestingly, the values of correlation coefficient statistics lies between 80 and 99 % for all five machine learning methods except Elman network model suggesting that the agreement of predictions with the actual class is closely related and can be used for estimation and prediction of actual classes in realistic environment. The best outcome of our study is the values of MAE and RMSE using SVM that is almost within the range of 0–1 only and for most of the cases it is just below 0.500 suggesting that the model will not break down under realistic conditions and shall be able to estimate and predict software reliability accurately. Therefore software practitioners can apply the proposed model to assess the software reliability of the software product that will facilitate the project managers to measure the acceptable level of reliable software at a specified severity level before the deployment within time and budget constraints. The precision and ROC-area of the model predicted with DTs is another positive outcome of our study which precisely covers 80–90 % area under ROC. However, the relative performances of DTs and ANNs (Elman network) model have been very inconsistent and discouraging particularly to small sized datasets due to the lack of adequate training and testing of the prediction model. Finally, the overall performance of model predicted using SVM, makes it the better choice in comparison to other machine learning techniques used in our study.

Table 8 Summary of predictions using CCNN

Data sets	Correlation coefficient	MAE	Training RMSE	Testing RMSE
1	0.9978	1.9188	2.5728	2.5799
2	0.9979	0.8082	0.9862	2.0899
3	0.9895	1.0356	1.5825	3.5197
4	0.9972	0.9172	1.1327	1.7751
5	0.9996	5.0218	6.3647	8.2574
6	0.9977	1.1608	1.4255	1.9633
14C	0.9971	0.4664	0.7782	3.5455
17	0.9974	0.6367	0.7900	1.5237
27	0.9968	0.7656	0.9394	1.7410
40	0.9981	1.4842	1.7825	2.2120
SS1A	0.9986	1.3032	1.6736	2.2259
SS1B	0.9794	14.3782	12.8095	13.6802
SS1C	0.9993	2.3560	2.9762	3.1020
SS2	0.9983	2.5025	3.1829	3.7161
SS3	0.9956	5.5567	7.4658	4.1969
SS4	0.9988	2.1166	2.7746	4.0151

Table 9 Summary of prediction accuracy using Naive Bayes

Data sets	Precision	Recall	F-measure	ROC area
1	0.927	0.926	0.926	0.977
2	0.730	0.778	0.744	0.932
3	0.618	0.579	0.574	0.904
4	0.434	0.566	0.484	0.893
5	0.986	0.986	0.996	0.994
6	0.775	0.767	0.765	0.947
14C	0.439	0.444	0.437	0.857
17	0.177	0.184	0.179	0.818
27	0.395	0.390	0.378	0.833
40	0.894	0.881	0.822	0.943
SS1A	0.675	0.679	0.648	0.937
SS1B	0.950	0.949	0.949	0.984
SS1C	0.947	0.946	0.946	0.960
SS2	0.938	0.938	0.937	0.964
SS3	0.968	0.968	0.967	0.981
SS4	0.934	0.934	0.933	0.962

6.5 Threats to validity

The present study for software reliability prediction using machine learning techniques has various limitations that are common with most of the other studies available in literature. Some of the specific limitations of our study are described as follows: First, the measures could not be evaluated over updated failure dataset of real-life industrial projects due to the lack of empirical software failure data

Table 10 Summary of predictions using FIS

Data sets	Evaluation parameters			
	Correlation coefficient	MAE	Training RMSE	Testing RMSE
1	0.9948	0.0104	2.6424	5.3262
2	0.9933	0.0097	1.2037	10.1067
3	0.9584	0.1490	1.9758	52.5764
4	0.9886	0.0344	1.5622	43.8102
5	0.9981	0.0263	6.6603	52.6511
6	0.9943	0.0531	1.4776	31.6250
14C	0.9768	0.0138	1.4832	14.7064
17	0.9984	0.0080	0.4129	34.6184
27	0.9900	0.0754	1.0979	31.0034
40	0.9970	0.0159	1.3464	57.0903
SS1A	0.9984	0.0055	1.2242	17.3331
SS1B	0.9988	0.5960	3.5984	6.8076
SS1C	0.9981	0.0554	3.2871	15.9369
SS2	0.9980	0.0218	2.3476	10.5509
SS3	0.9985	0.0846	2.9688	70.1806
SS4	0.9982	0.0048	2.2484	24.0991

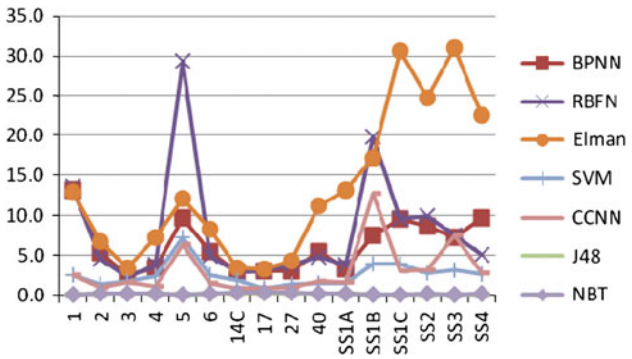

Fig. 3 The comparative performance measures of various machine learning methods

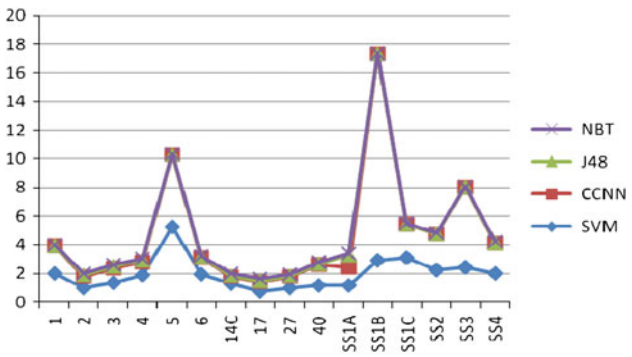

Fig. 4 The comparative performance measures of SVM, CCNN, and DTs using Naive Bayes

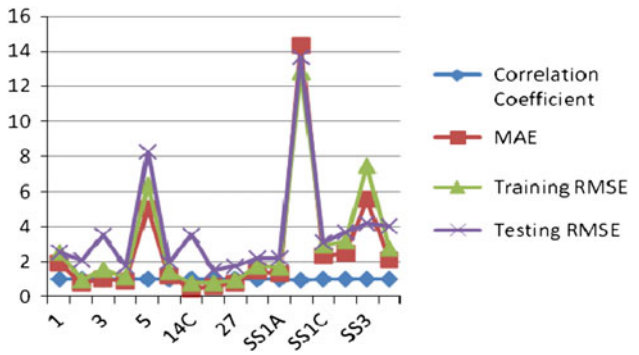

Fig. 5 The performance measures of CCNN model in terms of correlation coefficient, MAE and RMSE

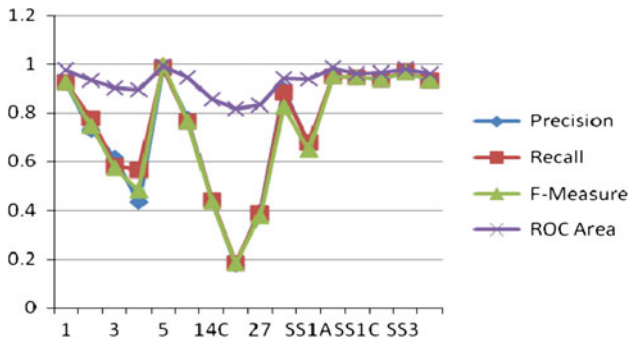

Fig. 6 The performance measures using Naive Bayes in terms of precision, recall, F-measure and ROC area

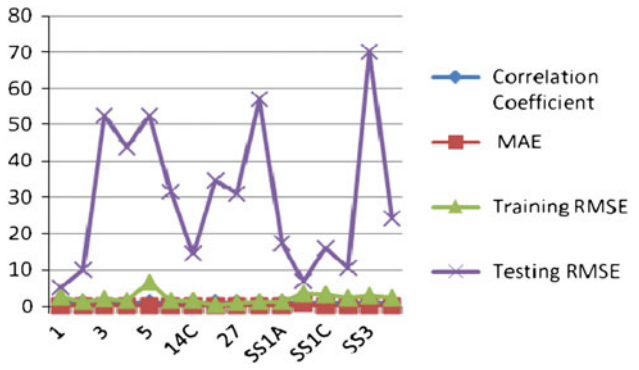

Fig. 7 The performance measures of FIS model in terms of MAE, training and testing RMSE

of modern computing system. So the prediction and assessment capability of our approach remains an open issue for the acceptance and deployment across different organizations. Hence, the generalization of the results achieved in our study is limited. Secondly, the performance and efficiency of our models for software reliability prediction using various machine learning methods depend on various factors such as (i) developing appropriate network architecture that could be generalized and customized in a realistic environment is still one of the major issues while modeling failure process for reliability improvement (ii) the representation of software failure data available through various resources is very old and sometimes outdated which is not updated frequently probably due to the competitive nature of business, fear of losing customers and other legal issues of software industry that need to be addressed separately before making such generalization. Thirdly, the effectiveness of these software reliability prediction models depends on the operational environment also. Therefore similar studies with large number of recent-failure datasets of real-life projects need to be carried out in order to establish the acceptability of the present models.

Finally, in spite of all these constraints the findings of our study provide the guidance for future research to assess the impact of past and present failure datasets for the prediction of software reliability using machine learning techniques. Hence further validations are required under different odd conditions to draw stronger conclusions for better quality prediction of the software products.

7 Conclusion

In this paper, we applied several machine learning methods namely ANNs (BPNN, RBFN, and Elman network), SVM, CCNN, DTs and FIS for the prediction of software reliability based on past failures of software products. The performances of various machine learning methods have been evaluated by using sixteen empirical databases extracted from DACS to predict failure intensity of the software. We empirically demonstrated that SVM model outperformed the model predicted using ANNs, CCNN, DTs and FIS in all datasets. CCNN shows very encouraging results and can be utilized as an alternative choice which outperformed the model predicted using BPNN, RBFN and Elman network model. Moreover, SVM and DTs can be applied for constructing software reliability growth models accurately by focusing on project's failure dataset of realistic environment.

However to make a generalization of our study more data based empirical studies which are capable of being verified by observation and experiment are required in near future. Thus present study of software reliability prediction using machine learning methods provide the guidance for future research in the field of assessing the impact of past and present failure data for realistic reliability prediction. In this paper the measures could not be evaluated over a large and updated failure dataset due to the lack of empirical software failure data of modern computing system in real-life scenario. Similar types of studies need to be carried out with different data sets to give generalized results across different organizations. We plan to replicate our study of software reliability prediction models by

introducing advanced machine learning techniques applied to a large category of failure datasets of real life industrial software projects. We also plan to focus on the cost benefit analysis also of our model that will help to determine whether a given software reliability prediction model would be economically viable in a realistic environment.

Acknowledgments The authors wish to thank all anonymous reviewers for their valuable suggestions and useful comments.

References

Aggarwal KK, Singh Y, Kaur A, Malhotra R (2006) Investigating the effect of coupling metrics on fault proneness in object-oriented systems. Softw Qual Prof 8(4):4–16

Aggarwal KK, Singh Y, Kaur A, Malhotra R (2009) Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: a replicated case study. Softw Process Improv Pract 14(1):39–62

Chen KC, Shiang Y, Liang TZ (2008) A study of software reliability growth from the perspective of learning effects. Reliab Eng Syst Saf 93(10):1410–1421

Eduardo OC, Aurora TR, Silvia RV (2010) A genetic programming approach for software reliability modeling. IEEE Trans Reliab 59(1):222–230

Fawcett T (2006) An introduction to ROC analysis. Pattern Recognit Lett 27:861–874

Fei X, Ping G, Lyu MR (2005) A novel method for early software quality prediction based on support vector machine. In: Proceedings of the 16th IEEE international symposium on software reliability engineering (ISSRE'05), Beijing, China: 213–222

Goel AL, Okumoto K (1979) Time-dependent fault detection rate model for software and other performance measures. IEEE Trans Reliab 28(3):206–211

Goel B, Singh Y (2009) An empirical analysis of metrics. Softw Qual Prof 11(3):35–45

Han J, Kamber M (2006) Data mining: concepts and techniques. Morgan Kaufmann, India

Hanley J, McNeil BJ (1982) The meaning and use of the area under a receiver operating characteristic ROC curve. Radiology 143:29–36

Jung Hua L (2010) Predicting software reliability with support vector machines. In: Proceedings of 2nd international conference on computer research and development (ICCRD'10), Kuala Lumpur, Malaysia: 765-769

Kapur PK, Garg RB, Kumar S (1999) Contributions to hardware & software reliability. World Scientific, Singapore

Kapur PK, Gupta A, Jha PC, Goyal SK (2010) Software quality assurance using software reliability growth modelling: state of the art. Int J Bus Inf Syst 6(4):463–496

Kapur PK, Pham Hoang, Gupta A, Jha PC (2011) Software reliability assessment with OR applications. Springer, London

Karunanithi N, Whitley D, Malaiya Y (1992) Prediction of software reliability using connectionist models. IEEE Trans Softw Eng 18(7):563–574

Kohavi R (1995) The power of decision tables. In: The eighth european conference on machine learning (ECML-95), Heraklion, Greece, pp 174–189

Lawson J, Wesselman S, Craig W, Scott D (2003) Simple plots improve software reliability prediction models. Qual Eng 15(3):411–417

Littlewood B (1979) Software reliability model for modular structure. IEEE Trans Reliab 28(3):241–246

Lyu M (2005) Handbook of Software reliability engineering. TMH, India.

Malhotra R, Singh Y, Kaur A (2009) Comparative analysis of regression and machine learning methods for predicting fault proneness models. Int J Comput Appl Technol 35(2):183–193

Malhotra R, Kaur A, Singh Y (2011) Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines. Int J Syst Assur Eng Manag 1(3):269–281. doi:10.1007/s13198-011-0048-7

Matlab fuzzy logic toolbox: tutorials on fuzzy inference system and ANFIS using MatLab. Available at http://www.mathworks.com. Accessed 14 Feb 2011

Mueller JA, Lemke F (1999) Self-organizing data mining: an intelligent approach to extract knowledge from data. Dresden, Berlin

Musa JD (1980) Software life cycle empirical/experience data, data & analysis center for software. Available at http://www.dacs.org. Accessed 17 Sep 2010

Musa JD (1998) More reliable, faster, cheaper testing with software reliability engineering. Softw Qual Prof 1(1):27–37

Musa JD (2005) Software reliability engineering: making solid progress. Softw Qual Prof 7(4):5–16

Norman F (1997) Application of software reliability engineering for NASA space shuttle. International Symposium on Software Reliability Engineering (ISSRE) 1:71–82

Ohba M (1984) Inflexion S-shaped software reliability growth models. Stochastic models in reliability theory. Springer, Berlin, pp 144–162

Pham H (2006) System software reliability. Springer, London

Phillip S (2003) DTReg predictive modeling software available at http://www.dtreg.com. Accessed 8 Jan 2011

Ping PF, Hong WC (2006) Software reliability forecasting by support vector machines with simulated annealing algorithms. J Syst Softw 79(6):747–755

Raj K, Ravi V (2008) Software reliability prediction using soft computing techniques. J Syst Softw 81:576–583. doi:10.1016/jss.2007.05.005

Ross Q (1993) C4.5: programs for machine learning. Morgan Kaufman, San Mateo

Scott E, Christian L (1991) The cascade-correlation learning architecture. Tech Rep. CMU-CS-90-100, School of computer science Carnegie Mellon University, Pittsburgh

Singh Y, Kumar P (2010a) A software reliability growth model for three-tier client–server system. Int J Comp Appl 1(13):9–16. doi:10.5120/289-451

Singh Y, Kumar P (2010b) Determination of software release instant of three-tier client server software system. Int J Softw Eng 1(3):51–62

Singh Y, Kumar P (2010c) Application of feed-forward networks for software reliability prediction. ACM SIGSOFT Softw Eng Notes 35(5):1–6

Singh Y, Kumar P (2010d) Prediction of software reliability using feed forward neural networks. In: Proceedings of computational intelligence and software engineering (CiSE'10), Wuhan, China: 1–5. doi:10.1109/CISE.2010.5677251

Vapnik V (1995) Nat Stat Learn Theory. Springer, New York

Weka: classification and regression software (2010). Available at http://www.cs.waikato.ac.nz. Accessed 17 Feb 2011

Witten IH, Frank E (2011) Data Mining: Practical machine learning tools and techniques with Java implementations, 3rd edn. Morgan Kaufman, Addison-Wesley, San Francisco

Xingguo L, Yanhua S (2007) An early prediction method of software reliability based on support vector machine. In: Proceedings

international conference on wireless communications, networking and mobile computing (WiCom'07), Hefei: 6075–6078

Yang B, Xiang L (2007) A study on software reliability prediction based on support vector machines. In: Proceedings of international conference on industrial engineering and engineering management (IEEM'07), Singapore, 1176–1180