



# Machine learning techniques for software vulnerability prediction: a comparative study

Gul Jabeen<sup>1,2</sup> · Sabit Rahim<sup>2</sup> · Wasif Afzal<sup>3</sup> · Dawar Khan<sup>4,5</sup> · Aftab Ahmed Khan<sup>2</sup> · Zahid Hussain<sup>2</sup> · Tehmina Bibi<sup>6</sup>

Accepted: 7 February 2022 / Published online: 4 April 2022

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

## Abstract

Software vulnerabilities represent a major cause of security problems. Various vulnerability discovery models (VDMs) attempt to model the rate at which the vulnerabilities are discovered in a software. Although several VDMs have been proposed, not all of them are universally applicable. Also most of them seldom give accurate predictive results for every type of vulnerability dataset. The use of machine learning (ML) techniques has generally found success in a wide range of predictive tasks. Thus, in this paper, we conducted an empirical study on applying some well-known machine learning (ML) techniques as well as statistical techniques to predict the software vulnerabilities on a variety of datasets. The following ML techniques have been evaluated: cascade-forward back propagation neural network, feed-forward back propagation neural network, adaptive-neuro fuzzy inference system, multi-layer perceptron, support vector machine, bagging, M5Rule, M5P and reduced error pruning tree. The following statistical techniques have been evaluated: Alhazmi-Malaiya model, linear regression and logistic regression model. The applicability of the techniques is examined using two separate approaches: goodness-of-fit to see how well the model tracks the data, and prediction capability using different criteria. It is observed that ML techniques show remarkable improvement in predicting the software vulnerabilities than the statistical vulnerability prediction models.

**Keywords** Software vulnerability · Machine learning · Prediction models

## 1 Introduction

The existing software systems have become larger and more complex. Software vulnerabilities have also increased rapidly, potentially causing an increasing number of serious security threats. Many researchers have modeled new vulnerability trends from different vulnerability datasets. The main goal of these studies is to find best vulnerability discovery model, to predict the number of vulnerabilities for the given software. Before releasing a product by software developers to the customers, predicting the frequency of disclosure of vulnerabilities is helpful for the customers as

well as vendor to help them allocate resources properly. In addition, such predictions can also satisfy not only the functional and technical requirements, but also provide useful information to evaluate the risk associated with a software [1]. After releasing, the developers must ensure that patches are available as soon as possible for the vulnerabilities that will be discovered.

Traditional software security research focuses on security models. These models contain access controls that include encryption schemes and intrusion detection systems. In literature, there exist two frequently used vulnerability prediction models which are focused on developing and building models using mathematical and vulnerability density functions. These models are further divided in to two main categories: code-based techniques and time based techniques. Source code static analysis techniques focus on discovering the relationship between code attributes and the number of vulnerabilities which includes: text analysis and source code mining [2–7] and most of recent researches incorporate

---

✉ Dawar Khan  
dawar.khan@siat.ac.cn

Extended author information available on the last page of the article.

the machine learning techniques that can be helpful in automating the detection process of software vulnerabilities and accelerate the code inspection process [8–14]. Predicting vulnerabilities at the code level requires access to the source code of the system or application and are thus, limited to only open source applications because the code is easily available for them. Also, the analysis of bugs or flaws at this level is done at a very low level of abstraction.

To overcome the above mentioned problems, numerous models are dedicated to predict security-related vulnerabilities based on historical vulnerabilities data in the software [1, 15–19]. The purpose of these models is not to identify vulnerabilities but to predict the number of vulnerabilities that are likely to be discovered in the future. These models use different approaches and make assumptions in their analytic formulation. Their parameters are defined explicitly and have physical interpretations. In order to determine the parameters, there are always a set of assumptions to be made. These assumptions generalize the model but their applicability becomes a critical issue because the VDMs can predict different vulnerability discovery rates using the same data. VDMs face challenges due to four assumptions: time, operational environment, independence and static code [1, 19–21]. Although VDMs are accurate in terms of curve fitting however, these might not fit well in prediction [19] and are often not powerful enough to take nonlinear data into consideration. Previous studies in different domains have shown that machine learning (ML) techniques are successful in predicting nonlinear data better than statistical methods. These techniques envisage past data as input and require few or no assumptions for modeling potentially complex software. It is quite natural for software practitioners and researchers to know that which particular method tends to work well for a given vulnerability dataset and up to what extent quantitatively. This paper attempts to address this need by exploring different machine learning techniques which fit the vulnerability data as well as predict the future [22]. There are several other problems associated with VDMs. Most of them have inherited Software Reliability Growth Models' (SRMGs) assumptions. Many of these assumptions of SRMGs are not applicable in vulnerability discovery problems because vulnerabilities are a special class of defects that can permit circumvention of the security measures, which makes VDMs' effectiveness uncertain [23]. Therefore, this paper attempts to address the issues related to VDMs by exploring the machine learning techniques' predictive behaviour.

In this paper, we present an empirical study of 9 machine learning methods: cascade-forward back propagation neural network (CFBPNN), feed-forward back propagation neural network (FFBPNN), adaptive-neuro fuzzy inference system (ANFIS), multi-layer perceptron (MLP), support vector

machine (SVM), bagging, M5Rule, M5P and reduced error pruning tree (RepTree) and 3 statistical vulnerability discovery models: Alhazmi-Malaiya (AML) model, linear regression and logistic regression model, for the prediction of software vulnerabilities in order to draw conclusions leading to widely accepted models. The performance of ML techniques is evaluated, both; based on goodness-of-fit and predictive capability, as excellent goodness of fit does not necessarily mean a superior predictive capability [21]. We make contributions on the following main questions:

1. How well machine learning techniques fit and predict the future trends of different vulnerability datasets?
2. What are the results of comparing the machine learning techniques with each other and most popular quantitative VDMs?

The rest of the paper is organized as follow. Section 2 describes the related work conducted on software vulnerability prediction. In Sections 5 & 6, machine learning techniques and software vulnerability prediction models are described. Section 7 includes the models' evaluation criteria and the results. Section 10 concludes the paper.

## 2 Related work

Prior research on software vulnerability prediction models is focused on developing and building models using mathematical and vulnerability density functions. These models are divided in to two main categories: Code attribute-based models and Time series-based models.

The code attribute-based models focus on discovering the relationship between code attributes and the number of vulnerabilities. Rahimi and Zargham proposed a method based on code properties to predict vulnerabilities that do not require historical data [16]. Lin et al. proposes a deep-learning based framework to extract the information applicability in vulnerability detection from the cross-domain datasets [14]. Ban et al. proposed a performance evaluation study that automatically extracts features from the source code using a deep learning algorithm, for software vulnerability detection [24]. In order to tap the potential of deep learning in a vulnerability detection system, a deep learning software vulnerabilities detection model called VulDeepEcker proposes the concept of code gadgets that is some intermediate representation between data dependency and control dependency [25]. Gupta et al. proposed a DeepFix algorithm which applies deep learning to generate fixes for simple syntax errors in student code [26]. Shar et al. analyzed the software vulnerabilities based on source code and is limited to open source applications only [27]. Recently,

advance code-based models have been proposed [28–32]. Many studies also targeted automated vulnerability identification using different machine learning techniques [33–38]. The techniques which require access to the source code are limited to open source applications only [39, 40]. Therefore, the researchers have focused on historical vulnerability data models called vulnerability discovery models (VDMs).

Recent vulnerability predictive techniques have modeled vulnerabilities based on time, which describe the relationship between the total number of vulnerabilities by calendar time and consider calendar time as the independent variable using vulnerability discovery models. Alhazmi and Malaiya have proposed two models for the process of vulnerability discovery using the data of windows: Windows 98 and Windows NT 4.0 [15]. They have compared different VDMS by fitting the data for major operating systems and found that AML fits better than other models in most cases. Rescorla found that the number of vulnerabilities follow an exponential decay curve [41]. Gul et al. proposed a multiple error iterative analysis model to enhance the predictive accuracy of existing VDMs [42]. Joh and Malaiya introduced and evaluated alternative S-shaped models based on the Weibull, beta, gamma, and normal distributions [21]. Zhu et al. proposed a prediction model for software vulnerability in which the probability and severity of vulnerability occurrences are determined by logistic and binomial distribution function [43]. They also examined the applicability of two models: logistic and linear model for the vulnerability discovery process on several versions of operating systems. Anand has a systematic approach for quantification of number of the discovered vulnerabilities to predict and scrutinising the loopholes [44]. Jhonstan et al. introduced multivariate methods to model vulnerability discoveries in web-browser software and present expert-judgment data-gathering techniques [45–47]. Shrivastava et al. investigated the quantitative vulnerability discovery models for predicting the total number of vulnerabilities detected, during the operational phase of the software [48]. Movahedi et al. tried to increase the accuracy of vulnerability discovery models using the clustering techniques [49, 50].

In the literature, very limited research has been done on the applicability of machine learning techniques to find vulnerability rates. Most of the techniques used code attributes to predict software vulnerabilities. Scandariato et al. use SVM to build a vulnerability prediction model, but their model consisted of sources code metrics [51]. Gul et al. proposed an integrated model to predict a number of software vulnerabilities by integrating artificial neural network and vulnerability discovery models [52]. Movahedi et al. introduced a neural network model for predicting the cumulative number of vulnerabilities to model the non-linearities

associated with vulnerability disclosure [22]. Catal et al. designed and implement a software vulnerability prediction web services which will be hosted on Azure cloud computing platform in machine learning techniques [53]. Sultana et al. design and performed experiments based on statistical tests to propose a set of software metrics to predict vulnerable classes and methods to classify java code as vulnerable or non-vulnerable java code, using supervised machine learning algorithms [54]. Shar et al. proposed supervised and semi-supervised learning to develop vulnerability models based on code attributes. The results show that these models can predict vulnerability discovery rate with high precision [27]. However, such models are valid only for the applications where the source code is easily available [42, 51]. In this paper, we present an empirical study based on machine learning techniques and also compare them with the popular statistical VDMs.

### 3 Data collection

We collected the vulnerability data from the publicly available National Vulnerability Database (NVD, <http://nvd.nist.gov>) maintained by National Institute of Standard and Technology (NIST) whose experts analyzed the vulnerabilities reported to the NVD and assigned proper attributes to the defects prior to the data entries [55]. The NVD is therefore considered a high-quality database and has been used by several researchers. In total, we collected six vulnerability datasets: Windows XP, Windows 8, Windows 7, Firefox, Internet Explorer and Windows Server 2008. These vulnerability datasets represent the major categories of software systems, namely, operating systems, web servers and web browsers. We collected the vulnerabilities of each application starting from the first day of the release or the earliest available data in the NVD. We aggregated all vulnerabilities for each application over a monthly period. Table 1 shows

**Table 1** Number of vulnerabilities for the software systems and observed period

S.No	Datastes Name	Data Collection Period	Total number of vulnerabilities
1	Windows XP	2001-2014	671
2	Window 7	2009-2017	499
3	Window 8	2015-2017	153
4	Firefox	2003-2017	623
5	Internet Explorer	2007-2017	801
6	Windows Server 2008	2008-2017	460

the descriptive statistics of all vulnerability datasets including name of software, the total number of vulnerabilities, data collection period.

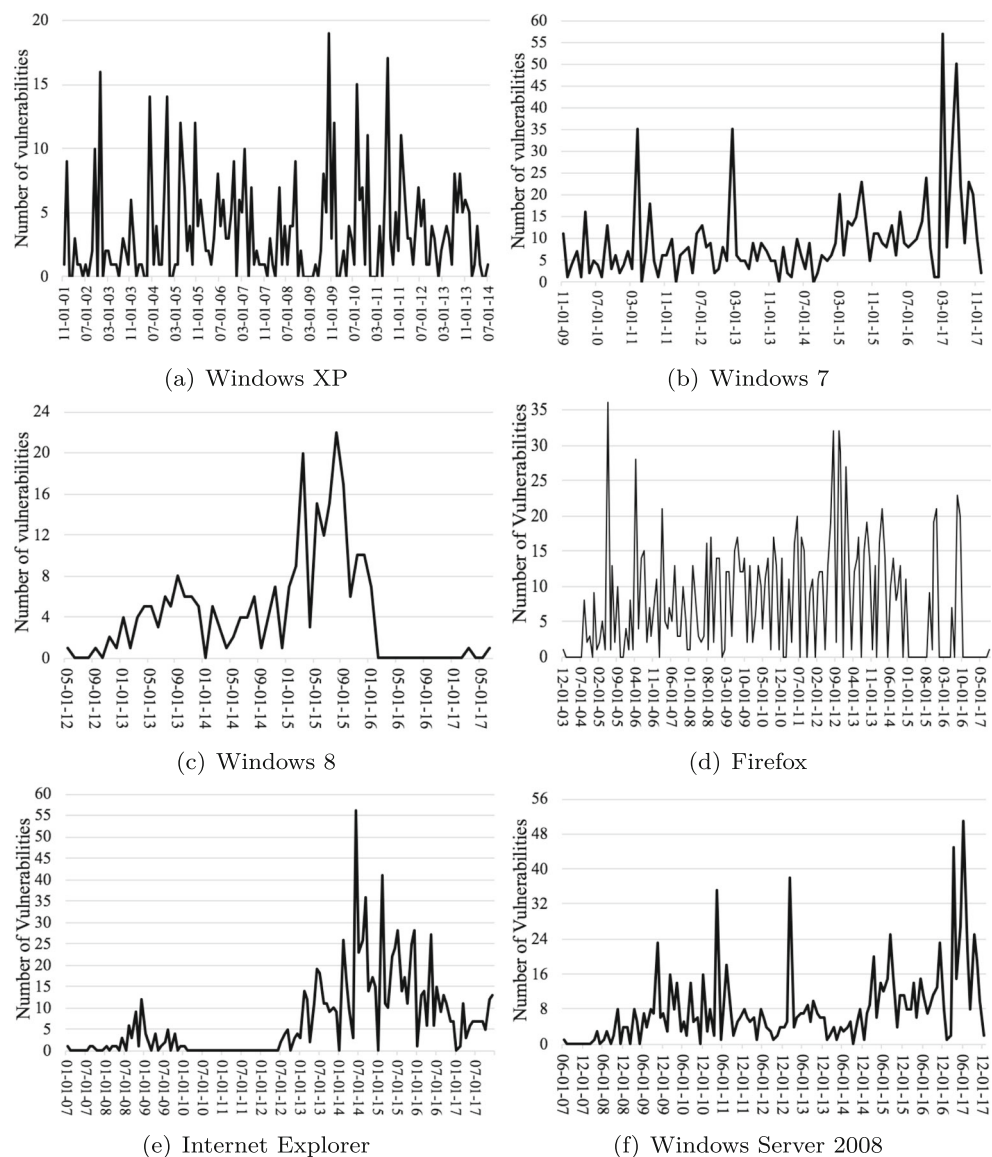
Figure 1 graphically presents the number of vulnerabilities aggregated by each month for the six vulnerability datasets. The Windows XP dataset includes all reported vulnerabilities over a period from 2001 to 2014, as shown in Figure 1(a). The Windows 7 dataset includes all reported vulnerabilities from 2009 to 2017, as shown in Figure 1(b). The Windows 8 dataset includes all reported vulnerabilities from 2015 to 2017, as shown in Figure 1(c). The Firefox dataset includes all reported vulnerabilities from 2003 to 2017, as shown in Figure 1(d). The Internet Explorer dataset includes all reported vulnerabilities from 2007 to 2017, as

shown in Figure 1(e). The Windows Server 2008 dataset includes all reported vulnerabilities from 2008 to 2017, as shown in Figure 1(f).

## 4 Dependent and independent variables

The dependent variable in our study is the vulnerability rate, which is used to study the number of vulnerability occurrences at each time interval. The number of vulnerabilities is aggregated by month, and time is measured in terms of month and year. Time is considered to be independent variable. In this paper, we predict the dependent variable based on the number of vulnerabilities to be detected using

**Fig. 1** Number of Vulnerabilities in Each Month with Calendar Time



different ML techniques. For the corresponding vulnerabilities, time in terms of months is chosen as the independent variable.

## 5 Statistical software vulnerability models

In this section, the three statistical vulnerability discovery models used in the paper are introduced, namely, the Alhazmi-Malaiya Logistic (AML), the Linear model, and the Logistic model. Following section introduces the models in detail.

### 5.1 Alhazmi-Malaiya model (AML)

The AML model is proposed by Alhazmi and Malaiya in [15] and is the only S-shaped model relying on capturing the underlying process of vulnerability discovery. It is based on the observation that the attention given to a newly released software increases at the initial phase, reaches its peak after sometime, and declines when the new competing versions are introduced. The model assumes that the rate of cumulative vulnerabilities governed by two factors on the right side of Equation 1. One of these factors decreases, as the remaining undetected vulnerabilities decline. The second factor increases with the time required to consider the rising share of the installed base. Assume that the vulnerability discovery rate is given by the differential equation:

$$\frac{d\Omega}{dt} = A\Omega(B - \Omega) \quad (1)$$

where  $\Omega$  is the cumulative number of vulnerabilities and  $t$  is the calendar time.  $A$  and  $B$  are empirical constants determined from the recorded data. By solving the differential equation, the following equation is obtained:

$$\Omega = \frac{B}{BC^{-ABt} + 1} \quad (2)$$

where  $C$  is a constant introduced in solving Equation 1.

The AML model has been tested for goodness-of-fit by numerous studies and it is found to provide good fit. Some studies also tested the predictive power of the AML model and found that it performs well in most of the situations [21].

### 5.2 Linear model

The Linear model was used by Alhazmi et al. in [56] and attempts to model the relationship between a single input variable and an output variable. A complex model is comprised of dozens of input variables. The model has been found to fit well when the learning phase is short and when the vulnerabilities are shared with the successive versions of the software. We have used a uni-variate linear regression

method to represent the relationship among dependent and independent variables. The formula of the linear model is represented as follows:

$$\Omega(t) = u + vt \quad (3)$$

where  $v$  and  $u$  are, respectively, the slope and constant of the equation (are also called regression coefficients),  $t = 0$ , and  $\Omega = 0$ ,  $\Omega(t)$  increases indefinitely as the system grows linearly. We use Akaike criterion for model selection, and implemented through Weka class [57].

### 5.3 Logistic regression model

The Logistic regression (LR) is a standard statistical method for analyzing a data having one or more independent variables. The AML model is based on the logistic regression model, the latter of which has been used in several earlier studies to predict fault-prone and vulnerability-prone entities [58–60]. It is a more flexible probability model to predict software vulnerabilities because the probability of logistic regression has a flexible S-shaped curve. The general form of logistic regression is as follows:

$$P = \frac{1}{1 + e^{-(\beta_0 + \sum_{i=1}^l \beta_i x_i)}} \quad (4)$$

where LR computes the probability  $P$  that an entity is vulnerability-prone for given metric value,  $x_i$ 's are the independent variables and  $\beta$  is a parameter that is estimated numerically.

## 6 Machine Learning (ML) techniques used for vulnerability prediction

In this section, we explain nine well-known and widely used ML techniques: CFBPNN, FFBPNN, ANFIS, MLP, SVM, Bagging, M5Rules, M5P and RepTree. These are used to predict software vulnerabilities based on past quantitative vulnerability data.

### 6.1 Feed-Forward Back Propagation Neural Network (FFBPNN)

The most popular training method for the feed-forward neural network is the back-propagation learning algorithm which provides a way to train multi-layer feed-forward neural networks. A neural network consists of a number of elements called neurons, and these neurons are grouped to form a layer. Each neuron has a number of inputs and a single output. A single-layered FFBPNN has one layer of sigmoid functions, followed by an output layer of linear neurons. The input layer, with a transfer function, permits the network model to learn both linear and nonlinear



relationships between the input and output variable vectors. The operation of the feed-forward neural network is represented as follows:

$$net_i = b_i + \sum_{j=1}^n x_j w_{ij} \quad \text{and} \quad Y_i = f(net_i) \quad (5)$$

where  $n$  is the number of input element, that is,  $x_1, x_2, \dots, x_n$ ;  $w_{ij}$  is a set of connecting links associated with weights  $w_{i1}, w_{i2}, \dots, w_{in}$ ; and  $Y_i$  is the output function of the previous layer of network. The activation function  $f()$  is used to process the input signals and generate the final output of the neurons. The *newff* function of MATLAB Toolbox has been used to train and test the network to create an FFBPNN [61].

The cumulative time of vulnerabilities ( $x_i$ ) is the input to the FFBPNN and the output of the FFBPNN is the predicted number of software vulnerabilities ( $Y_i$ ), where  $i$  is the software vulnerability occurrences time sequence index. We have constructed feedforward network with one hidden layer of size 10 and specified the variable learning rate resilient back-propagation algorithm such as 'trainrp'.

## 6.2 Cascade-Forward Back Propagation Neural Network (CFBPNN)

The CFBPNN is a type of neural network that consists of several layers of neurons, with each layer being biased, and including an input connection to each of previous layer. By using the MATLAB neural network toolbox, weights and biases are initialized for each layer. The CFBPNN is similar to the FFBPNN, as it uses a back-propagation algorithm to update weights. However, the difference of this layer is that the input values are computed after every hidden layer is back-propagated to the input layer and the weights adjusted successively.

In this study, the CFBPNN is applied to vulnerability occurrences data. Thereby, the neurons in the input layer are the lags of vulnerability time data, whereas the output is the number of the software vulnerabilities. It constitutes the direct relationship between the input and output node. To create a CFBPNN, the *newcf* MATLAB function used to train and test the network [61]. A cascade-forward network has been constructed with one hidden layer of size 10, Tan-sigmoid transfer function was used to achieve an optimized status and used the training function as 'trainrp'.

## 6.3 Adaptive Neuro Fuzzy Inference System (ANFIS)

The ANFIS technique works similarly to neural networks. The ANFIS is a supervised method that combines the advantages of artificial neural networks (ANNs) and fuzzy

inference systems. It can construct an input-output mapping based on both human knowledge and the stipulated data pairs [62]. We have explored the capability of the ANFIS for software vulnerability prediction based on historical vulnerability data of different software. The ANFIS is a fuzzy system whose function parameters have been tuned using neuro-adaptive learning methods, similar to methods used in training neural networks. The ANFIS combines the low-level computational power of neural networks with the great reasoning power capability of fuzzy inference system (FIS) [63]. The FIS basic models steps are given as follows:

- Step 1: Identification of input variables (vulnerability time) and output variable (cumulative vulnerabilities).
- Step 2: Development of fuzzy profile of input/output variables.
- Step 3: Definition of the relationships between vulnerability time and cumulative variables using the fuzzy inference system.

The FIS structure was generated using MATLAB ANFIS toolbox, which calculates the membership function parameters such that these parameters best fit with the associated FIS using the given vulnerability input/output training data. In this study a gaussian membership function is used for input data, the linear type of sugeno-fuzzy model is used to set membership function for output variables. Back-propagation gradient descent function and least square method are used to combine the hybrid algorithm as ANFIS network learning algorithm, thus making error tolerance limit to 0.

## 6.4 Multi-Layered Perceptron (MLP)

A multi-layer perceptron is a deep artificial neural network comprised of more than one perceptron. It works the same way as an FFBPNN wherein the back propagation algorithm is used in the form of gradient descent. An MLP is composed of an input layer to receive input data and an output layer that predicts input, and in between those two layers, an arbitrary number of hidden layers is present; these last are the true computational engines of an MLP. A nonlinear function such as sigmoid function is associated with each node, except for the input nodes. An MLP network learns a specific target function and adjusts weights properly using a general method of linear optimization (a gradient function). For this, the derivative of the errors function concerning the network weight is measured. The network weights are changed because of error decrease. The square Euclidean distance is used to compute the error between the actual output and desired output of a network.

An MLP algorithm follows the following steps:

- Step 1: The input and the desired output are presented first.
- Step 2: Each layer calculates the actual output, and the results are then passed to the next layer as input. Finally, the final output is calculated.
- Step 3: The weight from the output is adapted, and work is resumed, but backward.

In this study, the function MLP from the Weka class library has been used to implement the MLP to train with the vulnerability dataset. The vulnerability time data is used as input to the MLP along with the number of vulnerability occurrences. Back-propagation algorithm is used to train the neural network and sigmoid is used as an activation function. Number of hidden layers are three. Default values are used for the the rest of the parameters defined in Weka.

### 6.5 Support Vector Machine (SVM)

As support vector machine (SVM) is a pattern classifier and learning system, which constructs an N-dimensional hyperplane that separates the dataset into two optimal categories [64]. A basic SVM is linear, but the machine can also be used for nonlinear data by using a kernel function to map the non-linear data into linear feature space. The linear model can represent nonlinear data when it is constructed in the new space. Support vector machines can be used as alternative training methods for polynomial, and radial basis functions and MLP networks that use a kernel function. Support vectors are the vectors near the hyperplane and find the oriented hyperplane so that the margin between the support vector is maximized. The SVM model uses an optimization method to identify support vectors  $s_i$ , weights  $\alpha_i$ , and biases  $b$ , which are used to classify vectors  $x$  according to the equation below:

$$c = \sum \alpha_i k(s_i, x) + b, \quad (6)$$

where  $k$  is the kernel function and a dot product. If  $c \geq 0$ , then  $x$  is classified as a member of the first group instead of the second group. The training process may be slow, but the accuracy is increased because of the capability of SVMs for non-linear decision-bound modeling.

In this study, the regression optimizer was used based on the SMO algorithm with a radial basis function (RBF) kernel for the prediction process. The  $c$  parameter is specified as 1.0 and batchSize 100. The Weka class library has been used for implementing support vector machine to six vulnerability datasets. The vulnerability trained data, which is defined as the data to develop the regression model. We have used the vulnerability time variable as the predictor variable and vulnerability number as the observed variable. The SVM finds a function that has most error deviation from the observed lateral displacement for all the training data.

### 6.6 Bagging

The Bagging method is based on the idea that predictors can be rendered more stable by repeated bootstrapping and averaging the bootstrap samples, hence reducing variance [65]. Bagging is a simple and compelling ensemble method and is a general procedure that can be used to reduce the variance of high variance algorithms. Advantages of bagging have been found in cases wherein predictors are unstable, such as when small perturbations of the training set produce marked changes in model training. A significant application of bagging is in prediction using a random forest in which case bagging is used to improve the high variance found in decision trees and to select the most frequent class from the predicted group of classes.

Given as vulnerability training dataset of size  $n$ , bagging generates  $m$  new training sets, each size  $n'$ , by sampling from new training datasets uniformly and with replacement. It is a kind of sample which is known as a bootstrap sample. Then,  $m$  models are fitted using the above  $m$  bootstrap samples and combined by averaging the output. RepTree is used as a classifier, size of each bag as a percent is 100 and the preferred number of instances batch bag set as 100. Rest of the parameters are also selected as default values in Weka.

### 6.7 M5Rules

M5rules is a technique based on Quinlan's M5 algorithm [66]. This technique uses the divide and conquer method for problems based on regression to generate a decision list. With each iteration, it builds a model tree using an M5 algorithm and makes the best leaf into a rule. The M5Rule algorithm produces propositional regression rules in an If-Then rule format using the routines from M5Model trees to generate a decision list [67]. The model developed by M5 is a multivariate linear model.

M5Rules uses a tree learner over the vulnerability training samples to train a pruned tree. At each stage, the instances covered by the best rule are removed from the vulnerability training data before generating the next tree. The batch size is specified as 100 and the minimum number of instances as 4.0.

### 6.8 M5P

M5P is a binary regression tree model based on Quinlan's M5 algorithm [66]. The last nodes are the linear regression functions that can generate continuous numerical attributes. It generates M5 model trees, combining a conventional decision tree with the incorporation of linear functions. A decision tree induction algorithm is introduced first to build a tree, after which a splitting criterion is applied along

with each branch to minimize the intra-subset variation in class values. This procedure is applied to the class values of each of the instance. To overcome the issue of over-fitting, backward pruning is applied to each leaf. The attribute that is selected for splitting maximizes the expected error reduction of the node. The standard deviation (SD) reduction, that is, the expected error reduction, is calculated in (7):

$$SDReduction = sd(t) - \sum \frac{|t_i|}{|t|} * sd(t_i) \quad (7)$$

where  $t_i$  corresponds to the vulnerability datasets ( $t_1, t_2, , t_3, \dots$ ) that result from splitting the node according to the selected vulnerability attributes. However, a linear regression model predicts future vulnerability numbers at the leaves. M5 model trees are explained in [68].

In this study, the function *M5P* from the Weka class library was used to conduct the experimental implementation on vulnerability datasets. All the parameters were used as default in Weka.

## 6.9 Reduced error pruning Tree (RepTree)

The RepTree method was proposed by Quinlan [66], and in it decision trees use pruning methods to reduce the complexity of tree structures without decreasing the accuracy of classification. RepTree is a rapid decision tree which builds a decision tree or a regression tree to perform the pruning with backward fitting [69] and only sorts numeric values only and produces sub-optimal trees under the constraint that a sub-tree (S) can only be pruned if it does not contain any other sub-tree (S) with a lower classification error. The constraint that the S contains no sub-tree with the same property ensures reduced error pruning in the bottom-up induction, and each node is visited only once to evaluate the need to prune it. The fundamental advantage of this method is its linear computational complexity. However, from the cases in the training set where the tree was constructed, this method requires a separate test set, and this method may lead to over-pruning if the test set is smaller than the training. In this study, the function *Reptree* from the Weka class library was used to conduct the experimental evaluation with six vulnerability datasets. All the parameters were used as default in Weka.

## 7 Methodology

This section explores various ML techniques (CFBPNN, FFBPNN, ANFIS, MLP, SVM, Bagging, M5Rules, M5P and RepTree) as well as statistical techniques (the AML model, the Linear model, the Logistic model) to predict software vulnerabilities. The vulnerability dataset for every

software (Windows XP, Windows 7, Windows 8, Firefox, Internet Explorer and Window Server 2008) is aggregated by month. We have divided the entire dataset into two parts: training and testing. The training dataset is applied to the techniques for predicting the cumulative number of software vulnerabilities and is also used to check the goodness-of-fit of any model. The training part of the dataset is also employed using a 10-fold cross-validation procedure, which is an alternative procedure that allows more of the data to be used for fitting and testing [22]. By using 10-fold validation, the entire dataset is divided randomly into 10 subsets, and each time one of the subsets is used as training data, and the other (10 - 1) subsets are used to validate the prediction model for software vulnerability. Cross-validation is considered an efficient method that maximizes the repeated utilization of past vulnerability data on the same set of data. The following steps are used, to perform our experiments:

- Step 1: The dependent and independent variables are selected (the cumulative number of vulnerabilities is used as the dependent variable and time, measured in months, is used as an independent variable).
- Step 2: The entire set of vulnerability data is divided into two parts: training and testing. The training dataset is used to train the ML techniques to predict the software vulnerabilities. For each software, we have used data from the last 20 months to test the predictive power of ML techniques
- Step 3: A 10-fold cross validation method is used to divide, the data into 10 folds. The nine parts are used for training, and the tenth part is used for validation.
- Step 4: After dividing training data in to 10-folds, the ML techniques are applied for training the models.
- Step 5: After training, the predicted vulnerabilities are recorded for the techniques.
- Step 6: The statistical efficacy measures are estimated based on the predicted results for all the selected datasets.
- Step 7: Finally, the empirical assessment of all techniques is performed to determine the best technique.

## 7.1 Model evaluation criteria

Quantitative models are generally evaluated in terms of applicability using goodness-of-fit measures. However, even if the goodness-of-fit for a model is extremely satisfactory for the available data values, it is possible that the models may not predict future values well. Since the main use of models is to produce accurate predictions, the predictive capability is the most appropriate metric with which to compare alternative models [52]. In this study, both evaluation criteria are selected to compare not only different



ML techniques themselves but also compare them with well known statistical quantitative VDMs.

### 7.1.1 Goodness-of-fit analysis

Here we use four goodness-of-fit measures that are commonly used to examine the performance of VDMs. The criteria are as follows:

- **Chi-Square ( $\chi^2$ ) test:** Used to compare the observed distribution with the expected probability distribution. The  $\chi^2$  is used to determine how well each model fits the vulnerability dataset. It is calculated as follows:

$$\chi^2 = \sum_{i=1}^n \frac{(a_i - p_i)^2}{p_i} \quad (8)$$

where  $a_i$  and  $p_i$  are the actual and predicted values at  $i^{th}$  time point. For the fit to be acceptable, the  $\chi^2$  value should be less than the corresponding  $\chi^2$  critical value for the given alpha level and the degree of freedom. Emran and Ye proposed a means of applying a  $\chi^2$  test to real datasets from the computer intrusion detections [70].

- **P-value test:** The  $P$ -value represents the probability that a value of the statistic is at least as high as the value of  $\chi^2$  in (8). The alpha value is selected as  $\alpha = 5\%$ . The null hypothesis for this study is that the actual distribution is described well by the fitted model. However, if the  $P$ -value of the  $\chi^2$  test is below 0.05, then the fit will be considered not statistically significant. A  $P$ -value closer to 1 shows better fitting for the test data.  $P$ -value criterion is used by many researchers to test model fit [15] [71].
- **R-Square ( $R^2$ ):** The  $R^2$ , which is the square of correlation between the dependent and independent variables, is the statistical measure that shows how close the data fit to the regression line [72] and is calculated using (9):

$$R^2 = 1 - \frac{\sum_{i=1}^n (a_i - p_i)^2}{\sum_{i=1}^n (a_i - p_i)} \quad (9)$$

where  $a_i$  and  $p_i$  are the actual and predicted values at  $i^{th}$  time point. The  $R^2$  values near to 1 indicate an extremely good fit, whereas lesser values show a poor or no fit. This measure is used by researchers to evaluate their model fitting capabilities such as [50] [52].

- **Correlation Coefficient (CC):** The correlation coefficient measures the agreement of predictions with the actual values thereby showing how closely the actual and predicted values are correlated. There are several types of correlation coefficient; we have used Pearson

correlation which is commonly used in linear regression. The formula of CC is shown in (10).

$$CC(x, y) = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sqrt{\sum (x - \bar{x})^2 \sum (y - \bar{y})^2}} \quad (10)$$

The correlation coefficient values lie between +1 and -1. A correlation coefficient of +1 indicates a perfect fit (meaning that variables are strongly related to each other) while the -1 correlation coefficient indicates a non-fitting model. Movahedi et al. used the correlation coefficient to find the metrics and the number of vulnerabilities in each file [50].

### 7.1.2 Predictive power analysis

- **Mean Absolute Error (MAE):** Used to measure how close the predicted values are to the actual values, estimating the output for each input to estimate whether the proposed method is biased or tends to over or underestimate. It is calculated as follows:

$$MAE = \frac{1}{n} \sum_{i=1}^n |p_i - a_i| \quad (11)$$

where  $a_i$  and  $p_i$  are the actual and predicted values of the model's outcome observations. The MAE is used as the model performance evaluation function in [52] and [73].

- **The Root Mean Square Error (RMSE):** The root mean square error (RMSE) is a frequently used measure of the predicted and actual values observed from the environment. It is computed as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (p_i - a_i)^2}{n}} \quad (12)$$

where actual output is  $a_i$  and predicted output is  $p_i$ . The RMSE is preferred because it has the same scale as the data, and it is a popular means with which to forecast accuracy because of its theoretical relevance to statistical modeling [44].

- **Average Error (AE):** The average error (AE) is a predictive measure of how well a model makes predictions throughout the test phase [22]. It is denoted as follows:

$$AE = \frac{1}{n} \sum_{i=1}^n \left| \frac{p_i - a_i}{a_i} \right|, \quad (13)$$

where  $n$  is the total number of time interval,  $p_i$  is the predicted value and  $a_i$  is the actual value. The AE is always positive.

- **Average Bias (AB):** The average bias (AB) is the general bias of the model, evaluating the tendency of over and underestimation. It is denoted as follows:

$$AE = \frac{1}{n} \sum_{i=1}^n \frac{p_i - a_i}{a_i} \quad (14)$$

where  $n$  is the total number of time interval (one per month),  $p_i$  is the predicted value and  $a_i$  is the actual value. The AB can be positive or negative depending on whether the model is prone to over or underestimation [22].

## 7.2 Models' result analysis with vulnerability datasets

We evaluate the nine ML techniques and three well-known vulnerability prediction models as mentioned above on six datasets. The data and the fitted curves for all models using six datasets are shown in Fig. 2. The comparison criteria for goodness-of-fit has been conducted to see how well the models fit to the software vulnerability datasets by using correlation coefficient,  $R^2$ , the Chi-square ( $\chi^2$ ) and the  $P$ -value on six vulnerability datasets. We next compare the models using their prediction capabilities and examine how well the ML techniques and statistical models can predict the number of vulnerabilities in advance. The predictive capability of VDMs are also more important than model fitting since the main use of a model is to predict the future trends rather than only analyzing the past performance of predictive models. In the following section, each model's prediction capability is evaluated to assess how well the ML techniques and statistical models predict the future trend. The following section shows the goodness-of-fit and predictive power comparison separately.

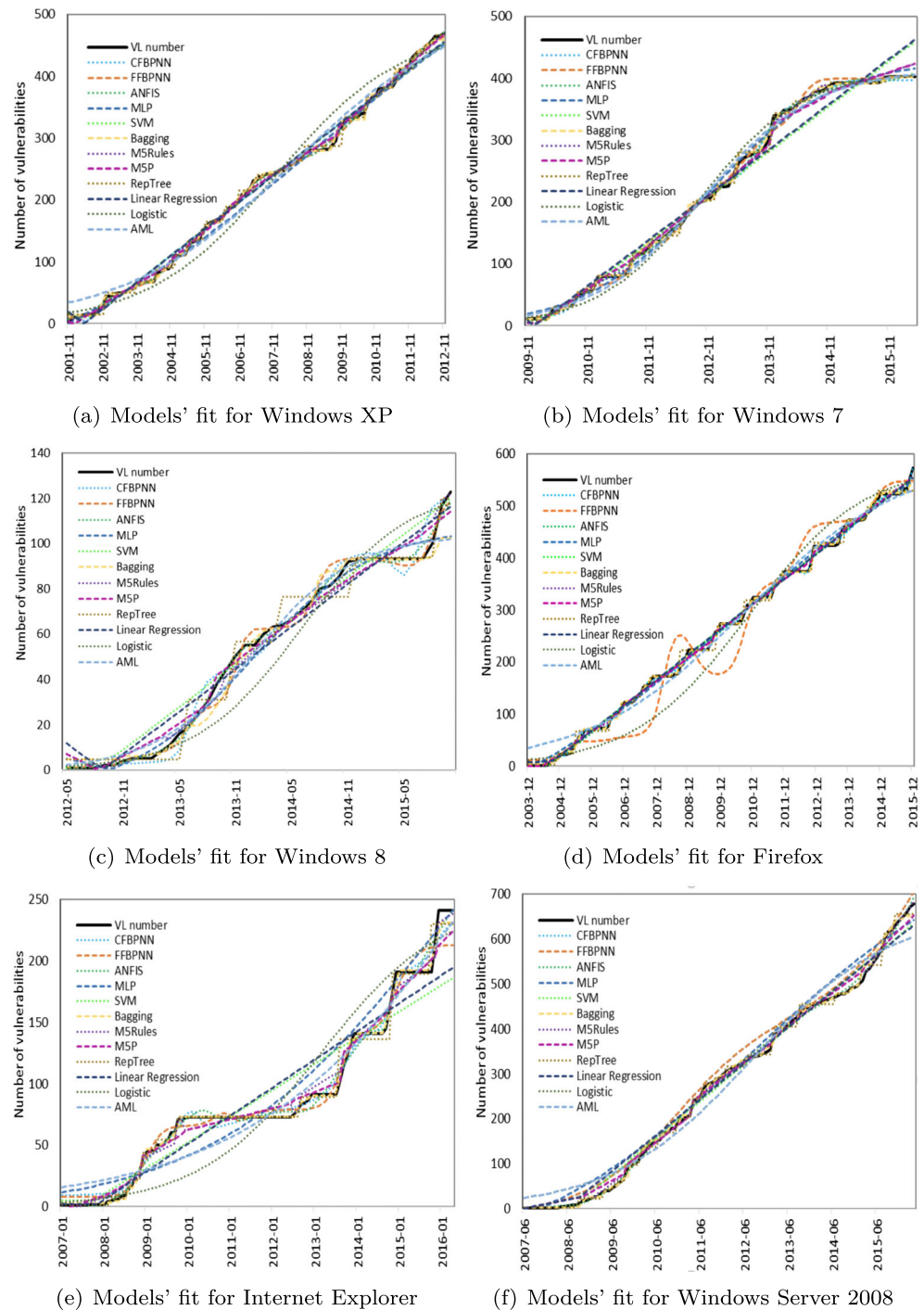
**Windows XP** For fitting power comparison of Windows XP, all data points are used to train and fit the models except last 20 data points, which are used to test the models. Fig. 2 shows the graph of the fitting comparison of all models based on Windows XP. The ML and statistical models can be seen to yield a good fitting. To see how well the models fit to Windows XP vulnerability data, the goodness-of-fit is evaluated using the comparison criteria: CC,  $R^2$ , Chi-square ( $\chi^2$ ) and  $P$ -value, as seen in Table 2. The bold numbers means the results of best ML techniques and other models in the columns. The results show that for Windows XP, the CC is within the ranges of 0.99–0.9999 in most of the predictions, except linear and logistic model, with a value of 0.9824, and 0.9686 respectively. We can see that  $R^2$  values are also close to 1, except in the linear and logistic models, which shows low fit. For Chi-square goodness-of-fit test, we selected ad alpha level of 0.05. Among them, FFBPNN,

ANFIS, Bagging and M5PRules shows significant good  $\chi^2$  values for Windows XP. However, the  $\chi^2$  values of linear regression, logistic regression, and AML are greater than the  $\chi_c^2$ , and therefore the fit is considered non-significant. The  $P$ -value in Table 2 shows that fits for same models are not accepted since the  $P$ -values are less than 0.05. For linear regression and logistic regression and AML, the  $P$ -values are less then 0.05. However, it can be misleading because the plot for Windows XP in Fig. 2a for every model fits the dataset well. Also the  $R^2$  and  $CC$  values are close to 1 which means very good fitting. FFBPNN, ANFIS, Bagging, M5Rules and RepTree show best fitting because the  $\chi^2$  test with a  $P$ -value is equal to 1 and also fits well because their  $CC$  and  $R^2$  values are more close to 1.

For the predictive power comparison, the last 20 data points of Windows XP are used to test the models. The predictive power of the models is evaluated using the comparison criteria: RMSE, MAE, AB and AE. From Table 2, it can be seen that the RMSE and MAE for the predictions based on the ANFIS, MLP, M5PRules and M5P show the smallest values of all models, within the range of 5 - 20. CFBPNN and SVM show RMSE and MAE in between the range of 20 - 40, which are considered as second highest predictive power models. Average bias values for Windows XP indicates that it has consistently negatively biased across all models. ANFIS, MLP, M5P and M5Rules have the lowest AE, within the rage of 0 - 0.02 for Windows XP vulnerability dataset. The normalized error values for each vulnerability dataset are plotted in Fig. 3. For Windows XP, Fig. 3a further demonstrates the predictive ability of all models, showing that predicted errors of the ANFIS, MLP, M5Rules and M5P are close to 0 percent error line.

**Windows 7** For the fitting power comparison of windows 7, all data points are used to fit the model, except the last 20 data points, which are used to test the models. The fitting result of windows 7 is shown in Fig. 2b. The ML and statistical models are shown to fit well with the Windows 7 data. The goodness-of-fit is further evaluated using the comparison criteria: CC,  $R^2$ ,  $\chi^2$  and  $P$ -values, as seen in Table 3. The bold numbers means the results of best ML techniques and other statistical models in the columns. From Table 3, we observe that the values of  $R^2$  and  $CC$  for most of the ML techniques are significantly close to 1. The Chi-square values for Windows 7 in Table 3 shows that the linear regression, logistic regression, and AML have less Chi-square values than Chi-critical ( $\chi_2 > \chi_{critical}^2$ ). The fit was judged to be poor for all of three statistical models with the  $P$ -value very close to zero, which is significantly less than 0.05. Only the FFBPNN, ANFIS, bagging and M5Rule techniques fit the given dataset very well with  $P$ -value being 1. The results can be further verified by predictive power comparison.

**Fig. 2** Models' fit for 9 ML Techniques and Three Statistical VDMs



For the predictive power comparison, the last 20 data points of Windows 7 are used to test the models. The predictive power of models is evaluated using the comparison criteria: RMSE, MAE, AB and AE. Table 3, demonstrate that the RMSE and MAE values for the predictions based on the ANFIS and FFBPNN are smallest among models. Average bias values for Windows 7 indicates mixed bias, showing overestimation for some models underestimation for others. The AE values for Windows 7 show that ANFIS

and FFBPNN have the lowest AE, within the range of 0 - 0.03. The normalized error values for each vulnerability dataset are plotted in Fig. 3. For Windows 7, Fig. 3b also demonstrate that the predicted errors of the FFBPNN and ANFIS, are close to the zero percent error line.

**Windows 8** In this section, we examine fitting power of models using another vulnerability dataset, Windows 8. For the goodness-of-fit power comparison, all data points of

**Table 2** Comparison of Machine Learning Techniques and Statistical Models' Descriptive and Predictive Power for Windows XP

	Goodness-of-fit comparison				Predictive power comparison			
	Correlation Coefficient	R-Square	Chi-square $\chi_c^2 = 159$	P-value	RMSE	MAE	AB	AE
CFBPNN	0.9970	0.9941	92.0	0.7802	29.69	33.01	-0.0232	0.0544
FFBPNN	<b>0.9995</b>	0.9953	<b>24.3</b>	<b>1</b>	41.30	38.31	-0.0696	0.0696
ANFIS	<b>0.9995</b>	<b>0.9991</b>	<b>24.9</b>	<b>1</b>	<b>6.79</b>	<b>5.77</b>	-0.0031	<b>0.0084</b>
MLP	0.9971	0.9977	116.5	0.8302	<b>16.39</b>	<b>13.78</b>	-0.0368	<b>0.0168</b>
SVM	0.9970	0.9979	115.0	0.8443	24.24	28.41	-0.0265	0.0265
Bagging	<b>0.9995</b>	<b>0.9991</b>	<b>26.9</b>	<b>1</b>	78.61	73.40	-0.0891	0.1091
M5Rules	<b>0.9992</b>	0.9984	<b>35.9</b>	<b>1</b>	<b>11.54</b>	<b>13.50</b>	0.0166	<b>0.0166</b>
M5P	0.9987	0.9975	158.1	0.0606	<b>12.90</b>	<b>11.17</b>	0.0088	<b>0.0105</b>
RepTree	<b>0.9990</b>	0.9986	72.7	<b>1</b>	44.18	38.60	-0.0748	0.0748
Linear Regression	0.9842	0.9687	364.6	1.141E-23	55.69	56.29	-0.0232	0.0732
Logistic	0.9882	0.9746	488.6	2.133E-42	48.81	25.95	-0.0898	0.0898
AML	0.9927	0.9973	159.0	0.0547	40.13	38.28	-0.0749	0.0649

Windows 8 are used to fit the models except for the last 20 data points, which are used for testing the models. The fitting results of Windows 8 are shown in Fig. 2c. The ML and statistical models are shown to fit the Windows 8 data well. Table 4, indicates that the CC and  $R^2$  values of most of the ML techniques are close to 1. However, MLP, Reptee and statistical models show less Chi-square values than Chi-critical ( $\chi_2 > \chi_{critical}^2$ ). CFBPNN, FFBPNN, ANFIS and bagging outperforms than all ther ML techniques with best fitting results.

For the predictive power comparison, the last 20 data points of Windows 8 are used to test the models. The predictive power of models is evaluated using the comparison criteria: RMSE, MAE, AB, and AE. From Table 4, it can be seen that RMSE and MAE for the predictions based on the

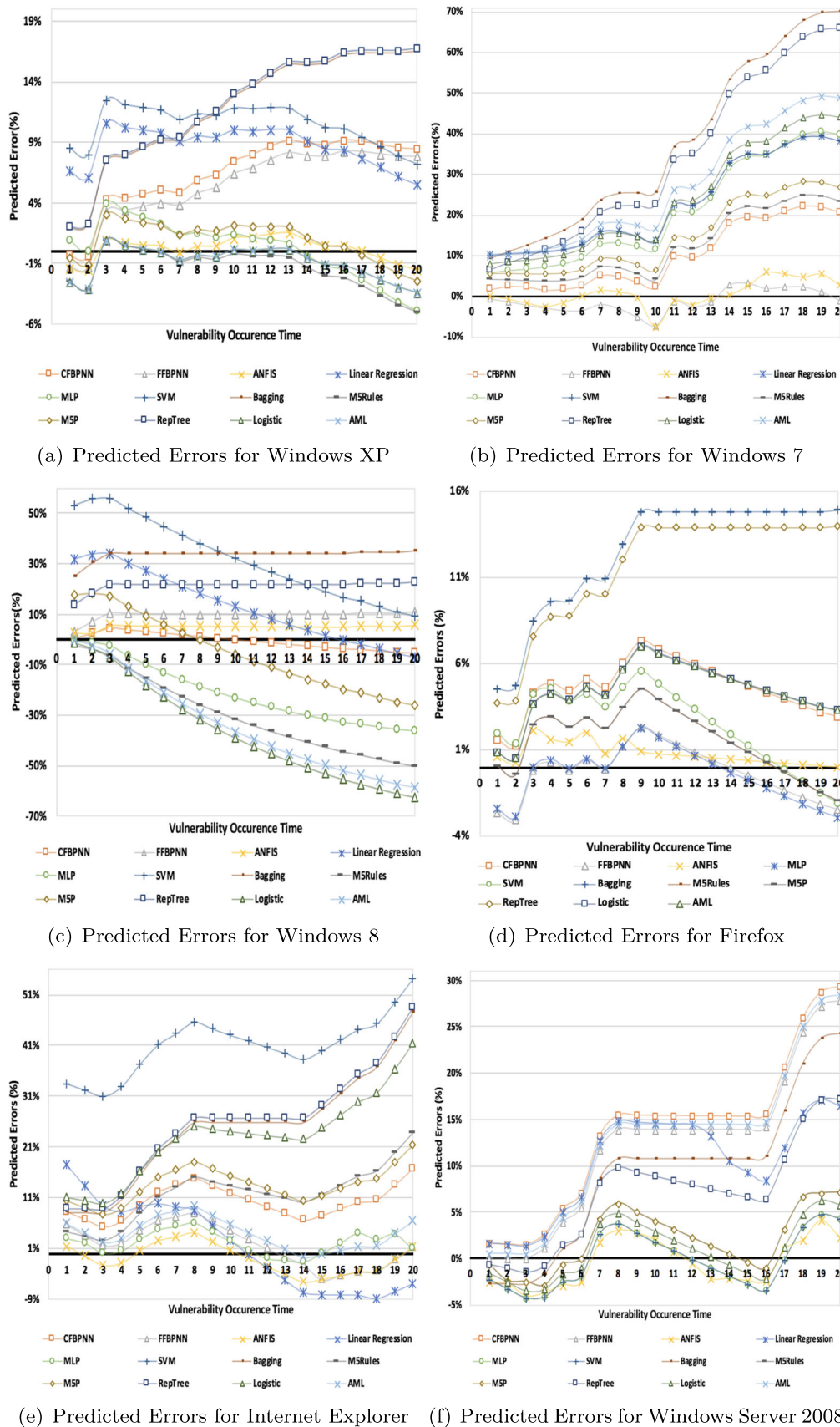
FFBPNN and ANFIS shows smallest values among all the models. However, the second class of models which show good predictive capabilities are CFBPNN, SVM, MLP, M5P, RepTree and linear regression which lie in the range of 25-50. Average bias values for Windows 8 indicate that it also has mixed bias, with some models showing over-estimation and others underestimation. The AE values for Windows 8 show that FFBPNN and ANFIS have the lowest AE values. The normalized error values for each vulnerability dataset are plotted in Fig. 3. Figure 3c also demonstrates that the predicted errors of the FFBPNN and ANFIS are close to zero percent error line.

**Firefox** In this section, we examine another vulnerability dataset (Firefox). For goodness-of-fit power comparison, all

**Table 3** Comparison of Machine Learning Techniques and Statistical Models' Descriptive and Predictive Power for Windows 7

	Goodness-of-fit comparison				Predictive power comparison			
	Correlation Coefficient	R-Square	Chi-square $\chi_c^2 = 99.6$	P-value	RMSE	MAE	AB	AE
CFBPNN	0.9933	0.9877	94.32	0.1047	88.11	67.13	-0.1056	0.0861
FFBPNN	0.9950	<b>0.9989</b>	88.4	0.1968	<b>21.34</b>	<b>18.23</b>	-0.0934	<b>0.0264</b>
ANFIS	<b>0.9991</b>	<b>0.9980</b>	23.7	<b>1</b>	<b>19.83</b>	<b>17.78</b>	-0.0224	<b>0.0245</b>
MLP	0.9909	0.9819	85.8	0.2731	58.40	63.67	-0.0441	0.0617
SVM	0.9929	0.9851	90.0	0.1665	99.67	88.01	0.1746	0.1286
Bagging	<b>0.9984</b>	<b>0.9988</b>	32.7	0.9999	189.70	217.09	-0.0945	0.2055
M5Rules	<b>0.9990</b>	<b>0.9980</b>	21.1	<b>1</b>	99.8	93.78	0.0170	0.1234
M5P	0.9973	0.9946	36.29.0	0.9999	102.38	93.06	0.0182	0.1231
RepTree	0.9965	0.9930	80.73	0.394	206.7	176.2	-0.0923	0.2345
Linear Regression	0.9868	0.9738	151.44	1.26E-13	151.44	132.9	0.1822	0.1822
Logistic	0.9856	0.9719	363.63	1.93E-38	143.06	137.49	-0.0816	0.1821
AML	0.9858	0.9719	350.77	3.08E-36	140.14	134.65	-0.0749	0.2014





**Fig. 3** Illustration of Predicted Errors for 9 ML Techniques and Three Statistical VDMs



**Table 4** Comparison of Machine Learning Techniques and Statistical Models' Descriptive and Predictive Power for Windows 8

	Goodness-of-fit Comparison				Predictive Power Comparison			
	Correlation Coefficient	R-Square	Chi-square $\chi_c^2 = 58.1$	<i>P</i> -value	RMSE	MAE	AB	AE
CFBPNN	0.9912	0.9822	32.11	0.8651	<b>45.95</b>	<b>36.70</b>	-0.1853	0.1440
FFBPNN	<b>0.9991</b>	<b>0.9984</b>	<b>11.41</b>	<b>1</b>	<b>22.26</b>	<b>22.59</b>	-0.1792	<b>0.0877</b>
ANFIS	<b>0.9990</b>	<b>0.9981</b>	<b>6.30</b>	<b>1</b>	<b>12.37</b>	<b>12.52</b>	0.0094	<b>0.0487</b>
MLP	0.9806	0.9743	72.70	0.003	<b>30.76</b>	<b>37.63</b>	-0.2856	0.2983
SVM	0.9856	0.9814	<b>38.30</b>	0.634	42.02	48.67	0.0290	0.2026
Bagging	<b>0.9997</b>	<b>0.9989</b>	<b>6.00</b>	<b>1</b>	63.06	63.88	-0.3140	0.2140
M5Rules	0.9940	0.9883	<b>33.50</b>	0.8222	148.97	125.10	0.0742	0.4208
M5P	0.9833	0.9819	<b>34.90</b>	0.7733	<b>38.83</b>	<b>46.83</b>	-0.0049	0.1516
RepTree	0.9882	0.9765	75.90	0.0011	<b>44.73</b>	<b>44.05</b>	-0.1943	0.1743
Linear Regression	0.9814	0.9701	76.60	0.0009	<b>30.12</b>	<b>37.21</b>	0.0006	0.1215
Logistic	0.9810	0.9790	129.0	9.042E-11	187.12	230.20	-0.1737	0.7375
AML	0.9883	0.9775	125.8	3.6E-10	187.80	200.65	-0.3058	0.6058

data points are used to fit the models, except for last 20 data points, which are used to test the models. Figure 2d shows the graph of the fitting comparison of all models based on Firefox. All the ML and statistical models are seen to yield a good fit. Table 5 demonstrate, that CC and  $R^2$  values of most of the ML techniques are close to 1 with higher values. The ANFIS, MLP, Bagging, M5Rules, M5P and Reptree show the best fit, with Chi-square test producing *P*-values more close to 1 and they also fit well because their *CC* and  $R^2$  values are close to 1. The CFBPNN, FFBPNN and SVM and all statistical models show *P*-values less than 0.05, which indicates an extremely poor fit.

For the predictive power comparison, the last 20 data points of Firefox are used to test the models. The predictive

power of models is evaluated using the comparison criteria: RMSE, MAE, AB, and AE. Table 5 demonstrates that the RMSE and MAE for the predictions based on the ANFIS, CFBPNN and MLP models show the smallest values among all models. SVM, M5Rules, M5P and linear regression show the second class of models having lower RMSE and MAE values in the range of 20-50. Average bias values indicate a consistently negative bias across all models. The AE values for Firefox show that the FFBPNN, MLP, ANFIS, M5Rules and M5P models have the lowest AE values, within the range of 0 - 0.02 for the Firefox vulnerability dataset. The normalized error values for each vulnerability dataset are plotted in Fig. 3. For Firefox, Fig. 3d further demonstrates the predictive ability of all

**Table 5** Comparison of Machine Learning Techniques and Statistical Models' Descriptive and Predictive Power for Firefox

	Goodness-of-fit Comparison				Predictive Power Comparison			
	Correlation Coefficient	R-Square	Chi-square $\chi_c^2 = 174.1$	<i>P</i> -value	RMSE	MAE	AB	AE
CFBPNN	0.9979	0.9958	203.65	1.E-02	<b>19.02</b>	<b>15.34</b>	-0.0519	<b>0.0121</b>
FFBPNN	0.9973	0.9947	422.23	3.E-109	52.96	47.71	-0.0942	0.0371
ANFIS	<b>0.9994</b>	<b>0.9989</b>	<b>120.28</b>	<b>0.93</b>	<b>12.56</b>	<b>9.85</b>	-0.0179	<b>0.0078</b>
Multiple Layer Perceptron	0.9985	0.9971	169.14	0.084	<b>19.39</b>	<b>16.16</b>	-0.0435	<b>0.0128</b>
Support Vector Machine	0.9929	0.9918	194.1	0.0037	40.17	35.25	-0.0141	0.0238
Bagging	<b>0.9997</b>	<b>0.9995</b>	<b>122.14</b>	<b>0.91</b>	146.19	141.16	-0.1067	0.1101
M5Rules	<b>0.9991</b>	<b>0.9985</b>	<b>127.14</b>	<b>0.85</b>	29.61	25.22	-0.0104	<b>0.0197</b>
M5P	<b>0.9992</b>	<b>0.9984</b>	<b>156.56</b>	<b>0.24</b>	29.61	25.22	-0.0179	<b>0.0198</b>
RepTree	<b>0.9991</b>	<b>0.9983</b>	161.11	0.1704	137.52	132.15	-0.1244	0.1030
Linear Regression	0.9953	0.9906	401.99	5E-26	38.91	31.89	-0.0162	0.0251
Logistic	0.9950	0.9981	623.66	1.4E-10	57.43	54.02	-0.0750	0.0421
AML	0.9964	0.9931	192.08	0.0054	57.10	53.96	-0.0967	0.0421

models and shows that predicted errors of the FFBPNN, ANFIS, MLP, M5Rules, and M5P models close to zero percent error line.

**Internet explorer** In this section, we examine another vulnerability dataset, Internet Explorer. For the goodness-of-fit power comparison, all data points are used to fit the models except for last 20 data points, which are used to test the models. Fig. 2e shows the graph of the fit comparison of all models based on Internet Explorer data. Most of the ML and statistical models can be seen to yield a good fit. To determine how well the models fit to the Internet Explorer vulnerability data, goodness-of-fit is evaluated using the comparison criteria: correlation coefficient,  $R^2$ , Chi-square ( $\chi^2$ ) and  $P$ -value in Table 6. The result shows that the CC and  $R^2$  values are more close to 1 in most of the predictions. For the Chi-square goodness-of-fit test, we choose alpha level as 0.05. Almost most of the ML techniques show Chi-square values less than Chi-critical ( $\chi_{critical}^2$ ), and therefore the fits are considered significant except CFBPNN. The  $P$ -value shows that fits for CFBPNN, Linear Rergrssion, Logistic model and AML are not accepted since the  $P$ -values are less than 0.05. The  $CC$  and  $R^2$  values also show that these models fit poorly. The plot of Internet Explorer in Fig. 2e depicted the fitting of all models. ANFIS and Bagging models show best fit because the Chi-square test produce a  $P$ -value close to 1 and they also fit well because their  $CC$  and  $R^2$  values are more close to 1.

For the predictive power comparison, the last 20 data points of Internet Explorer are used to test the models. From Table 6, it can be seen that RMSE and MAE for the predictions based on the FFBPNN, ANFIS and MLP show smallest values among all models, in between the range of 5 - 20. Average bias values for Internet Explorer indicates

that it has a consistently negative bias across all models. The AE suggest that the FFBPNN, ANFIS and MLP models performed better than other models. Bagging, which shows good fitting power, has low predictive capability. Fig. 3e demonstrates the predictive ability of all models and shows that predicted errors of the FFBPNN, ANFIS and MLP models close to the zero percent error line.

**Windows server 2008** For the fitting power comparison of Windows Server 2008 (WS2008), all data points are used to fit the models except for last 20 data points, which are used to test the models. Figure 2f shows the graph of the fitting comparison of all models based on WS2008. All the ML and statistical models can be seen to yield a good fit. Fig. 2d shows the fitting values of each model. To see how well the models fit to WS2008 vulnerability data, the goodness-of-fit is also evaluated using the comparison criteria: correlation coefficient,  $R^2$ , Chi-square ( $\chi^2$ ) and  $P$ -value in Table 5. The results show that the CC is above 0.99 in most of the predictions except the Logistic and AML model. We can see that  $R^2$  values for ML techniques are also more close to one. All the statistical models have greater Chi-square values than ( $\chi_{critical}^2$ ), therefore the fit is considered non-significant. The  $P$ -value also shows that fits for same models are not accepted since the  $P$ -values are less than 0.05. The FFBPNN, ANFIS, and bagging models show the best fit because the Chi-square test produce a  $P$ -value equal or more close to 1 and the also fit well because their  $CC$  and  $R^2$  values are close to 1 which means very good fit.

For the predictive power comparison of WS2008, the last 20 data points of WS2008 are used to test the models. The predictive power of the models is also evaluated using the comparison criteria RMSE, MAE, AB and AE. Table 7 shows that the RMSE for the predictions based

**Table 6** Comparison of Machine Learning Techniques and Statistical Models' Descriptive and Predictive Power for Internet Explorer

	Goodness-of-fit Comparison				Predictive Power Comparison			
	Correlation Coefficient	R-Square	Chi-square $\chi_c^2 = 137.7$	$P$ -value	RMSE	MAE	AB	AE
CFBPNN	0.9950	0.9900	142.85	0.302	28.01	25.56	-0.1852	0.0909
FFBPNN	0.9989	0.9978	135.16	0.076	<b>19.12</b>	<b>17.62</b>	-0.2572	<b>0.0195</b>
ANFIS	<b>0.9992</b>	<b>0.9986</b>	<b>88.25</b>	<b>0.959</b>	<b>11.58</b>	<b>9.61</b>	0.0116	<b>0.0263</b>
Multiple Layer Perceptron	0.9916	0.9932	119.86	0.312	<b>15.36</b>	<b>13.17</b>	-0.0340	<b>0.0181</b>
Support Vector Machine	0.9911	0.9814	136.0	0.0685	39.39	35.95	-0.2392	0.2154
Bagging	<b>0.9991</b>	<b>0.9983</b>	<b>60.8</b>	<b>1</b>	135.92	129.80	-0.1922	0.1560
M5Rules	0.9971	0.9941	125.55	0.1977	55.04	44.90	-0.1020	0.0566
M5P	0.9977	0.9956	121.0	0.2717	142.44	136.62	-0.1178	0.1643
RepTree	0.9977	0.9956	104.96	0.6925	142.44	136.62	-0.1972	0.1643
Linear Regression	0.8370	0.7068	250.0	2E-12	251.81	243.17	-0.2549	0.3149
Logistic	0.7823	0.6781	2128.2	0	275.86	273.67	-0.1828	0.3689
AML	0.7687	0.7383	1331.9	5E-207	57.76	46.93	-0.0370	0.0592

**Table 7** Comparison of Machine Learning Techniques and Statistical Models' Descriptive and Predictive Power for Windows Server 2008

	Goodness-of-fit Comparison				Predictive Power Comparison			
	Correlation Coefficient	R-Square	Chi-square $\chi^2_c = 132.1$	P-value	RMSE	MAE	AB	AE
CFBPNN	0.9984	0.9970	130.77	0.0590	42.76	34.85	-0.1167	0.0441
FFBPNN	<b>0.9990</b>	<b>0.9982</b>	<b>70.91</b>	<b>0.9971</b>	<b>28.09</b>	<b>25.93</b>	-0.1047	0.0307
ANFIS	<b>0.9992</b>	<b>0.9985</b>	<b>46.18</b>	<b>1</b>	<b>9.89</b>	<b>8.50</b>	0.0049	<b>0.0209</b>
Multiple Layer Perceptron	0.9961	0.9923	130.0	0.0611	177.32	152.66	-0.0812	0.1703
Support Vector Machine	0.9976	0.9954	127.87	0.0825	<b>13.58</b>	<b>11.74</b>	0.0027	<b>0.0261</b>
Bagging	<b>0.9994</b>	<b>0.9989</b>	<b>43.5</b>	<b>0.9999</b>	222.22	191.04	-0.1242	0.2131
M5Rules	0.9989	0.9981	120.34	0.1821	<b>16.29</b>	<b>15.04</b>	-0.0193	<b>0.029</b>
M5P	0.9981	0.9979	129.22	0.0726	<b>16.89</b>	<b>15.51</b>	-0.0098	<b>0.029</b>
RepTree	0.9989	0.9979	85.1	0.9346	218.52	186.73	-0.1093	0.2078
Linear Regression	0.9927	0.9858	313.6	4E-22	159.79	140.46	0.0171	0.15789
Logistic	0.9896	0.9795	874.53	2E-120	237.96	214.28	-0.1112	0.2427
AML	0.9897	0.9795	744.0	5E-96	237.95	214.27	-0.1436	0.242

on the ANFIS, SVM, M5PRule and M5P models shows the smallest values among all models, within the range of 10 - 20. The MAE for the predictions based on the CFBPNN, FFBPNN and ANFIS models also show smallest values among all models. Average bias values for WS2008 indicate that it has mixed bias, sometimes predicting more and sometimes predicting fewer than actual number of vulnerabilities. the ANFIS, SVM, M5PRule and M5P models also have the lowest AE within the range of 0 - 0.03 for WS2008 vulnerability dataset. Fig. 3f demonstrate the predictive ability of all models that predicted errors of the CFBPNN, FFBPNN, ANFIS, SVM, and M5Rules models close to zero percent error line.

## 8 Discussion

This study examines the applicability of nine ML techniques: CFBPNN, FFBPNN, ANFIS, MLP, SVM, bagging, M5Rule, M5P, and RepTree to predict the software vulnerabilities of six different vulnerability datasets using two separate approaches: goodness-of-fit, to see how well the models track the data, and predictive capability, to see how well the model can predict future vulnerabilities. In general, ML techniques are much better at predicting software vulnerabilities than statistical vulnerability prediction models. The fit of ML techniques has been examined using CC,  $R^2$ , Chi-square and P-value, while predictive capability has been examined using the RMSE, MAE, AB and AE for the major operating systems (Windows XP, Windows 8, Window 7), web browsers (Firefox, Internet Explorer) and a server (Windows Server 2008). The fitting results show that ML techniques fit better than statistical VDMs in most of the cases. The results of goodness-of-fit show that most

of the ML techniques fit well with all the datasets, though some models show low predictive capability.

Based on the results obtained from the rigorous experiments that were conducted, the following observations can be made:

1. The fitting capability of the ANFIS model yields best fitting results for all the datasets. Tables 2–7 show that the ANFIS produces CC and  $R^2$  equal to 1 and the higher the correlation and  $R^2$  values, the better the fitting power of a model. ANFIS shows Chi-square values less than Chi-critical ( $\chi^2 < \chi^2_{critical}$ ) for every dataset and fits better in terms of P-value of  $\chi^2$  for every dataset. Therefore, the fit of the ANFIS is considered highly significant. Compared to other ML techniques and statistical models, the ANFIS shows good predictive results in terms of the RMSE (within the range of 5 to 15) and the MAE (within the range of 5 to 12) for the testing part of all of the vulnerability datasets. The AE values also show that the errors found in most of the predictions for the ANFIS is quite low compared to the errors calculated for the other techniques. Therefore, the ANFIS performs relatively well and is considered the best predictor in all of the vulnerability datasets.
2. The model that fits the second-best is bagging, which shows consistent fitting results with all of the datasets. The CC and  $R^2$  are also close to 1 (0.99) for all vulnerability datasets. Bagging also shows p-values from Chi-square tests greater than 0.05. However, although bagging produced remarkable fitting results, the predictive capability is very low for all of the datasets
3. FFBPNN technique also performs relatively well. It has the third best fitting and the second best predictive

capability. The correlation coefficient and  $R^2$  are always close to one for all the vulnerability datasets. These models fit the given datasets very well with most of the  $P$ -values very close to 1, except for Firefox. It shows good predictive results in terms of RMSE and MAE (within the range of 5 to 40) for testing part of most of the vulnerability datasets, except for Firefox. The AE values also show that the errors found in most of the predictions for FFBPNN are quite low in comparison to the errors given by the other mentioned techniques, except ANFIS. Therefore, FFBPNN is the second best predictor in terms of RMSE, MAE and AE.

4. M5P and M5Rules techniques also performed relatively well and their fitting capability was higher than those of ANFIS and bagging. They have the fourth best fitting and the second best predictive capabilities. The correlation coefficient and  $R^2$  are always close to 1 for all the vulnerability datasets. These models fit the given datasets extremely well, with most of the  $P$ -values very close to 1, except for those with respect to the two software systems. Compared to the other ML techniques and statistical models, the M5P and M5Rules models show good predictive results in terms of the RMSE and MAE (within the range of 5 to 50) for the testing part of most of the vulnerability datasets, except for that of Windows and Internet Explorer, where both models show RMSE and MAE values (greater than 50). The AE values also show that the errors found in most of the predictions for M5Rules and M5P are quite low compared to the errors calculated by the other techniques, except the ANFIS model. Therefore, M5Rules and M5P are considered the third best predictors in terms of RMSE, MAE and AE.
5. The SVM model also performed well in terms of fitting and predicting for some of the vulnerability datasets. It does not fit better in terms of  $P$ -values for some of the software systems but is able to generate encouraging results in terms of CC and R-square for all the datasets. The SVM has the lowest RMSE (in the range of 5 to 50), MAE (within the range of 10 to 50) and AE (within the range of 0 to 0.03) for most of the datasets, except Window 7 and Internet Explorer.
6. MLP also perform well in terms of fitting; it fits well in terms of  $P$ -values as well as CC and  $R^2$ , except for only one dataset. The predictive capability for two datasets shows low predictive errors.

Based on the performance analysis of ML techniques, used for the vulnerability data assessment, we found these techniques can be simultaneously operated on both quantitative and qualitative (code attribute) data. In this study, we also compared our results with well-known statistical vulnerability discovery models. In a comparison, we found that

the accuracy of most of the ML techniques is higher than the statistical VDMs and that ML techniques have a better ability to model the non-linear software vulnerability data. Empirically, we conclude that ML techniques provide more reliable performance and more accurate predictive results than statistical vulnerability prediction models. Machine learning techniques such as ANFIS, FFBPNN, M5Rules, M5P, MLP and SVM were found to be extremely useful for predicting number of software vulnerabilities in the future, even in high dimensional spaces and in small training datasets. They have the ability to model complex non-linear relationships and the capability to approximate any measurable function. Therefore, it is concluded that software vulnerability assessments and predictions using ML techniques can provide better results than conventional statistical models. Most of the ML techniques are rarely used for the quantitative assessment of software vulnerability prediction in the literature, and it is thus both important and desirable to include these models in software vulnerability discovery.

## 9 Threats to validity

The present study for software vulnerability prediction using ML techniques has several limitations. The first is that this study could not evaluate more datasets. It is important to conduct a similar type of study using more vulnerability datasets of different systems such as other operating systems, android applications and others. The size of the datasets is also important, because most of the ML techniques require large training datasets as input/output to correct learning, which is a computationally intensive process [74, 75], and different sizes of datasets are therefore required to validate the techniques. It is also important to get the data from different sources to check its validity or correctness. The generalization of the results achieved is limited because of the limited number of vulnerabilities used. The results evaluated in this paper used the Weka (an open source tool) and MATLAB tools. While Weka is freely available, MATLAB might not be available to other researchers to replicate our findings. The validity of the results of this study can be improved by conducting more repeated studies across different types and sizes of datasets.

## 10 Conclusion

In this paper, we applied ML techniques (cascade-forward back propagation neural network, feed-forward back propagation neural network, adaptive-neuro fuzzy inference system, multi-layer perceptron, support vector machine, bagging, M5Rule, M5P and reduced error pruning tree)

and statistical techniques (Alhazmi-Malaiya model, linear regression and logistic regression model) to the prediction of software vulnerabilities based on past vulnerability data. The performance of techniques has been evaluated using four goodness-of-fit and predictive power criteria. We have empirically demonstrated that the ANFIS model outperformed the other ML techniques for all of the vulnerability datasets. For each of the six datasets, the results show that the ANFIS predictions not only have the best goodness-of-fit results but also have best predictive capabilities. In addition, most of the ML techniques show satisfactory goodness-of-fit, namely, FFBPNN, SVM, MLP, M5Rules and M5P and bagging. Of these, FFBPNN, M5Rules and M5P show a high predictive capability. The comparison of statistical models with ML techniques shows that most of the ML techniques perform better and show significant results.

In future, we plan to improve the generalizability of our study by conducting experiments on more data from diverse software systems and expanding the range of deep learning algorithms (CNN,LSTM) and statistical techniques.

**Acknowledgments** This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No. 957212 and the ECSEL Joint Undertaking (JU) under grant agreement No 101007350. D. Khan was supported in part by NSFC (No.62150410433), Shenzhen Basic Research Program (JCYJ20180507182222355) and CAS-PIFI (No. 2020PT0013 ). We are thankful to the anonymous reviewers for their valuable comments and suggestions.

## References

- Kansal Y, Kumar P, Uday K (2018) Coverage based vulnerability discovery modeling to optimize disclosure time using multiattribute approach. June 2017, pp 1–12. <https://doi.org/10.1002/qre.2380>
- Goseva-Popstojanova K, Tyo J (2018) Identification of security related bug reports via text mining using supervised and unsupervised classification. In: 2018 IEEE International conference on software quality, reliability and security (QRS), IEEE, pp 344–355
- Şahin CB, Dinler OB, Abualigah L (2021) Prediction of software vulnerability based deep symbiotic genetic algorithms: Phenotyping of dominant-features. Appl Intell, pp 1–17
- Zeng J, Nie X, Chen L, Li J, Du G, Shi G (2020) An efficient vulnerability extrapolation using similarity of graph kernel of pdgs. In: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), IEEE, pp 1664–1671
- Dam HK, Tran T, Pham TTM, Ng SW, Grundy J, Ghose A (2018) Automatic feature learning for predicting vulnerable software components. IEEE Trans Softw Eng
- Piran A Vulnerability Analysis of Similar Code
- Morrison PJ, Pandita R, Xiao X, Chillarege R, Williams L (2018) Are vulnerabilities discovered and resolved like other defects ?. <https://doi.org/10.1007/s10664-017-9541-1>
- Chakraborty S, Krishna R, Ding Y, Ray B (2021) Deep learning based vulnerability detection: Are we there yet. IEEE Trans Softw Eng
- Kaloutsosoglou I, Siavvas M, Tsoukalas D, Kehagias D (2020) Cross-project vulnerability prediction based on software metrics and deep learning. In: International Conference on Computational Science and Its Applications, Springer, pp 877–893
- Li Z, Zou D, Xu S, Jin H, Zhu Y, Chen Z (2021) Sysevr: A framework for using deep learning to detect software vulnerabilities. IEEE Transactions on Dependable and Secure Computing
- Bhatt N, Anand A, Yadavalli VenkataSS (2021) Exploitability prediction of software vulnerabilities. Qual Reliab Eng Int 37(2):648–663
- Ban X, Liu S, Chen C, Chua C (2019) A performance evaluation of deep-learned features for software vulnerability detection. Concurrency and Computation: Practice and Experience 31(19): e5103
- Lin G, Wen S, Han Q-L, Zhang J, Xiang Y (2020) Software vulnerability detection using deep neural networks: A survey. Proc IEEE 108(10):1825–1848
- Lin G, Zhang J, Member S, Luo W, Pan L, Vel OD, Montague P, Xiang Y, Member S (2019) Software Vulnerability Discovery via Learning Multi-domain Knowledge Bases. IEEE Trans. Dependable Secur. Comput. PP(c):1. <https://doi.org/10.1109/TDSC.2019.2954088>
- Alhazmi OH, Malaiya YK (2005) Quantitative Vulnerability Assessment of Systems Software. Reliability and Maintainability Symposium, 2005. Proceedings. Annual, pp 615–620. <https://doi.org/10.1109/RAMS.2005.1408432>, <https://www.dropbox.com/s/pjc8a97q5vjomgp/Quantitativevulnerabilityassessmentofsystemssoftware.pdf?dl=0>
- Rahimi S, Zargham M (2013) Vulnerability Scrying Method for Software Vulnerability Discovery Prediction Without a Vulnerability Database . IEEE Trans Reliab 62(2):395–407. <https://doi.org/10.1109/TR.2013.2257052>
- Joh HC, Malaiya YK (2017) Periodicity in software vulnerability discovery, patching and exploitation. Int J Inf Secur 16(6):673–690. <https://doi.org/10.1007/s10207-016-0345-x>
- Wang X, Ma RUI, Li B, Tian D, Wang X (2019) E-WBM : An Effort-Based Vulnerability Discovery Model. IEEE Access 7:44276–44292. <https://doi.org/10.1109/ACCESS.2019.2907977>
- Anand A, Bhatt N, Alhazmi OH (2021) Modeling Software Vulnerability Discovery Process Inculcating the Impact of Reporters. pp 709–722
- Liu B, Shi L, Cai Z, Li M (2012) Software vulnerability discovery techniques: A survey. Proc. - 2012 4th Int. Conf. Multimed. Secur. MINES 2012, pp 152–156. <https://doi.org/10.1109/MINES.2012.202>
- Joh H, Malaiya YK (2014) Modeling Skewness in Vulnerability Discovery. Qual Reliab Eng Int, September 2013. <https://doi.org/10.1002/qre.1567>
- Movahedi Y, Cukier M, Gashi I (2019) Vulnerability prediction capability: A comparison between vulnerability discovery models and neural network models. Computers & Security 87:101596
- Anand A, Bhatt N, Aggrawal D (2020) Modeling Software Patch Management Based on Vulnerabilities Discovered. 27(2), pp 1–15. <https://doi.org/10.1142/S0218539320400033>
- Ban X (2018) A performance evaluation of deep-learned features for software vulnerability detection. November, 1–10. <https://doi.org/10.1002/cpe.5103>
- Li Z, Zou D, Xu S, Ou X, Jin H, Wang S, Deng Z, Zhong Y (2018) Vuldeepecker: A deep learning-based system for vulnerability detection. Network and Distributed System Security Symposium



26. Gupta R, Pal S, Kanade A, Shevade S (2017) Deepfix: Fixing common c language errors by deep learning. In: Thirty-First AAAI Conference on Artificial Intelligence
27. Shar LK, Briand LC, Tan HK, Member S (2015) Web Application Vulnerability Prediction Using Hybrid Program Analysis and Machine Learning. *IEEE Transactions on Dependable and Secure Computing* 12(6):688–707. <https://doi.org/10.1109/TDSC.2014.2373377>
28. Shar LK, Briand LC, Tan HBK (2015) Web application vulnerability prediction using hybrid program analysis and machine learning. *IEEE Transactions on Dependable and Secure Computing* 12(6):688–707
29. George TK, Jacob KP, James RK (2018) Token based detection and neural network based reconstruction framework against code injection vulnerabilities. *Journal of Information Security and Applications* 41:75–91
30. Akram J, Liang Q, Luo P (2019) Vcpr : Vulnerable code is identifiable when a patch is released (hacker's perspective). In: 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST), IEEE, pp 402–413
31. Akram J, Mumtaz M, Gul J, Luo P (2019) Droidmd: An efficient and scalable android malware detection approach at source code level. *Int J Inf Comput Secur*, 11(1). <https://doi.org/10.1504/IJICS.2019.10020453>
32. Akram J, Luo P (2021) Sqvdt: A scalable quantitative vulnerability detection technique for source code security assessment. *Software: Practice and Experience* 51(2):294–318
33. Li X, Wang L, Xin Y, Yang Y, Chen Y (2020) Automated vulnerability detection in source code using minimum intermediate representation learning. *Appl Sci* 10(5):1692
34. Saccante N, Dehlinger J, Deng L, Chakraborty S, Xiong Y (2019) Project achilles: A prototype tool for static method-level vulnerability detection of java source code using a recurrent neural network. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW), IEEE, pp 114–121
35. Partenza G, Amburgey T, Deng L, Dehlinger J, Chakraborty S (2021) Automatic identification of vulnerable code: Investigations with an ast-based neural network. In: 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), IEEE, pp 1475–1482
36. Hanif H, Nasir MHN, Ab Razak MF, Firdaus A, Anuar NB (2021) The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches. *J Netw Comput Appl*, p 103009
37. Semasaba AOA, Zheng W, Wu X, Agyemang SA (2020) Literature survey of deep learning-based vulnerability analysis on source code. *IET Softw* 14(6):654–664
38. Zheng W, Gao J, Wu X, Liu F, Xun Y, Liu G, Chen X (2020) The impact factors on the performance of machine learning-based vulnerability detection: A comparative study. *J Syst Softw* 168:110659
39. Geng J, Luo P (2016) A novel vulnerability prediction model to predict vulnerability loss based on probit regression. *Wuhan University Journal of Natural Sciences* 21(3):214–220
40. Roumani Y, Nwankpa JK, Roumani YF (2015) Time series modeling of vulnerabilities. *Computers & Security* 51:32–40
41. Rescorla E (2005) Is finding security holes a good idea?. 3(1)
42. Jabeen G, Rahim S, Sahar G, Shah AA, Bibi T (2020) An optimization of vulnerability discovery models using multiple errors iterative analysis method: An optimization of vulnerability discovery models. *Proceedings of the Pakistan Academy of Sciences: A. Physical and Computational Sciences* 57(3):47–60
43. Zhu X, Cao C, Zhang J (2017) Vulnerability severity prediction and risk metric modeling for software. *Appl Intell* 47(3):828–836
44. Anand A, Das S, Aggrawal D, Klochov Y (2017) Vulnerability discovery modelling for software with multi-versions. In: *Advances in reliability and system engineering*. Springer, pp 255–265
45. Johnston R, Sarkani S, Mazzuchi T, Holzer T, Eveleigh T (2019) Bayesian-model averaging using mcmcbytes for web-browser vulnerability discovery. *Reliability Engineering & System Safety* 183:341–359
46. Johnston RA (2018) A multivariate bayesian approach to modeling vulnerability discovery in the software security lifecycle. Ph.D. Thesis, The George Washington University
47. Johnston R, Sarkani S, Mazzuchi T, Holzer T, Eveleigh T (2018) Multivariate models using mcmcbytes for web-browser vulnerability discovery. *Reliability Engineering & System Safety* 176:52–61
48. Shrivastava AK, Kapur PK, Anjum M (2019) Vulnerability discovery and patch modeling: State of the art. *Reliab Eng*, pp 401–419
49. Movahedi Y (2019) Some guidelines for risk assessment of vulnerability discovery processes. Ph.D. Thesis, University of Maryland, College Park
50. Movahedi Y, Cukier M, Andongabo A, Gashi I (2019) Cluster-based vulnerability assessment of operating systems and web browsers. *Computing* 101(2):139–160
51. Scandariato R, Walden J, Hovsepian A, Joosen W (2014) Predicting vulnerable software components via text mining. *IEEE Trans Softw Eng* 40(10):993–1006. <https://doi.org/10.1109/TSE.2014.2340398>
52. Jabeen G, Ping L, Akram J, Shah AA (2019) An integrated software vulnerability discovery model based on artificial neural network. In: *SEKE*, pp 349–458
53. Catal C, Akbulut A, Ekenoglu E, Alemdaroglu M (2017) Development of a software vulnerability prediction web service based on artificial neural networks. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Springer, pp 59–67
54. Sultana KZ, Anu V, Chong T-Y (2021) Using software metrics for predicting vulnerable classes and methods in java projects: A machine learning approach. *Journal of Software: Evolution and Process* 33(3):e2303
55. Houmb SH, Franqueira VNL, Engum EA (2010) Quantifying security risk level from CVSS estimates of frequency and impact. *J Syst Softw* 83(9):1622–1634. <https://doi.org/10.1016/j.jss.2009.08.023>
56. Alhazmi OH, Malaiya YK, Ray I (2007) Measuring, analyzing and predicting security vulnerabilities in software systems. *Computers and Security* 26(3):219–228. <https://doi.org/10.1016/j.cose.2006.10.002>
57. Machine learning group. <http://www.cs.waikato.ac.nz>
58. El Emam K, Melo WL, Machado JC (2001) The prediction of faulty classes using object-oriented design metrics. *J. Syst. Softw.* 56(1):63–75. [https://doi.org/10.1016/S0164-1212\(00\)00086-8](https://doi.org/10.1016/S0164-1212(00)00086-8)
59. Chowdhury I, Zulkernine M (2011) Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. *J Syst Archit* 57(3):294–313. <https://doi.org/10.1016/j.sysarc.2010.06.003>
60. Liu MY (2006) Empirical Relation between Coupling and Attackability in Software Systems : A Case Study on DOS. In: *ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, Ottawa, Canada, pp 57–64
61. Demuth H (2009) *Neural Network Toolbox*. The MathWorks Inc., Natr
62. Jang J-R (1993) ANFIS : Adaptive-Ne twork-Based Fuzzy Inference System. *IEEE Trans. Syst. Man. Cybern.*, 23(3)

63. Tyagi K (2014) An adaptive neuro fuzzy model for estimating the reliability of component-based software systems. *Applied Computing and Informatics* 10(1-2):38–51. <https://doi.org/10.1016/j.aci.2014.04.002>
64. Lo J (2010) Early Software Reliability Prediction Based on Support Vector Machines with Genetic Algorithms. In: 5th IEEE Conference on Industrial Electronics and Applications, pp 2221–2226
65. Breiman L (1996) Bagging predictors. *Mach Learn* 24(2):123–140. <https://doi.org/10.1007/BF00058655>
66. Quinlan JR (1992) LEARNING WITH CONTINUOUS CLASSES 2. Constructing Model Trees. In: *Proceedings AL'92*, vol 92, pp 343–348
67. Duggal H, Singh P (2012) Comparative study of the performance of m5-rules algorithm with different algorithms
68. Alsultanny Y (2020) Machine learning by data mining reptime and m5p for predicating novel information for pm10
69. Galathiya A, Ganatra A, Bhensdadia C (2012) Improved Decision Tree Induction Algorithm with Feature Selection, Cross Validation, Model Complexity and Reduced Error Pruning, vol 3. <http://ijcsit.com/docs/Volume3/Vol3Issue2/ijcsit2012030227.pdf>
70. Emran SM, Ye N (2002) Robustness of chi-square and canberra distance metrics for computer intrusion detection. *Qual Reliab Eng Int* 18(1):19–28
71. Rathore SS, Kumar S (2021) An empirical study of ensemble techniques for software fault prediction. *Appl Intell* 51(6):3615–3644
72. Fonticella R (1998) The Usefulness of the R2 Statistic. *Society* 23:56–60
73. Yasasin E, Prester J, Wagner G, Schryen G (2020) Forecasting it security vulnerabilities—an empirical analysis. *Computers & Security* 88:101610
74. Amin A, Grunske L, Colman A (2013) An approach to software reliability prediction based on time series modeling. *J Syst Softw* 86(7):1923–1932. <https://doi.org/10.1016/j.jss.2013.03.045>
75. Afzal W, Torkar R, Feldt R (2012) Resampling methods in software quality classification. *Int J Softw Eng Knowl Eng* 22(02):203–223

**Publisher's note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Gul Jabeen** was born in Gilgit-Baltistan, Pakistan. She received the B.Sc. degree in computer science from Karakoram International University, Gilgit, Pakistan, the M.S. degree in computer science from International Islamic University, Islamabad, and the Ph.D. degree from Tsinghua University, Beijing, China, in 2019. Her research interests include software and information security, and prediction modeling.



**Sabit Rahim** is working as Assistant Professor in Department of Computer Sciences, Karakoram International University Gilgit, Pakistan; He received his MS from Hamdard University Pakistan and PhD from University of Science and Technology Beijing China. His research interest is machine learning; Cloud computing; IoTs, GIS and ICT in education in rural areas of developing countries. He is also member of provincial ICT policy for education of Gilgit-Baltistan, Pakistan.



quality assurance, evidential assessment of software engineering literature and software metrics.

**Wasif Afzal** is a Professor of Software Engineering at Mälardalen University, Sweden. He likes to do empirical research in software engineering in general and within software verification, validation and testing in particular. His core topics of research include prediction and estimation in software engineering, application of AI techniques to solve software engineering problems, decision-making based on software analytics, software testing and



2018. He got 2018-Excellent International graduate Award of the UCAS. His research interest includes computer graphics, visualization, Virtual and Augmented Reality and computer applications.

**Dawar Khan** is working as Postdoc Fellow with Shenzhen Institute of Advance Technology, Chinese Academy of Sciences, Shenzhen, China; under the distinguished fellowship of the CAS-PIFI. He is also an Assistant Professor with Department of Information Technology, The University of Haripur. Before this he was an Assistant Professor with Nara Institute of Science and Technology, Nara, Japan. He earned his PhD from Institute of Automation, CAS in



**Aftab Ahmed Khan** was born in Gilgit-Baltistan, Pakistan. He received the B.S. (equivalent) degree in information technology from Pir Mehr Ali Shah Arid Agriculture University, Rawalpindi, the M.Sc. degree from the Technical University of Berlin, Germany, and the Ph.D. degree in 3D geoinformatics (indoor navigation) from the Technical University Munich, Germany, in 2015. From 2013 to 2015, he was a Research Associate with the Technical University

of Munich. Since 2016, he has been an Assistant Professor with the Computer Science Department, Karakoram International University, Gilgit, Pakistan. He is the author of two book chapters and more than 10 peer-reviewed articles. His research interests include 3D geoinformatics and machine learning using earth science information).



**Tehmina Bibi** is working as Lecturer in Institute of Geology, University of Azad Jammu and Kashmir, Muzaffarabad, Pakistan. She received her M.Phil in Geology from University of Peshawar Pakistan and PhD from Universiti Teknologi Malaysia, Malaysia in Geoinformatics. Her research interest is Natural Hazards; Risk Assessment; disaster risk reduction and future modelling through IoTs in mountainous area of Asian countries. She is also member of




Pakistan Association of Petroleum Geoscientists (PAPG).



**Zahid Hussain** received the B.Sc. (Hons) degree in Mathematics from University of Sindh, Jamshoro. M.S. degree in Mathematics from Blekinge Institute of Technology, Sweden, in 2010. Ph.D in applied mathematics from Chung Yuan Christian University, Taiwan in the year 2019 with specialized in fuzzy mathematics and its generalization. His research interests include fuzzy sets, intuitionistic fuzzy sets, and hesitant fuzzy sets with their

applications in multicriteria decision making, pattern recognition, and fuzzy clustering.

## Affiliations

Gul Jabeen<sup>1,2</sup> · Sabit Rahim<sup>2</sup>  · Wasif Afzal<sup>3</sup>  · Dawar Khan<sup>4,5</sup>  · Aftab Ahmed Khan<sup>2</sup> · Zahid Hussain<sup>2</sup> · Tehmina Bibi<sup>6</sup>

Gul Jabeen  
jgl14@mails.tsinghua.edu.cn

Sabit Rahim  
sabit.rahim@kiu.edu.pk

Wasif Afzal  
wasif.afzal@mdu.se

Aftab Ahmed Khan  
aftab.ahmed@kiu.edu.pk

Zahid Hussain  
zahid.hussain@kiu.edu.pk

Tehmina Bibi  
tehmina.bibi@ajku.edu.pk

<sup>1</sup> Tsinghua University, Beijing, China

<sup>2</sup> Department of Computer Science, Karakoram International University, Gilgit, Pakistan

<sup>3</sup> Mälardalen University, Västerås, Sweden

<sup>4</sup> Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China

<sup>5</sup> Department of Information Technology, The University of Haripur, Haripur, Pakistan

<sup>6</sup> Institute of Geology, University of Azad Jammu and Kashmir, Muzaffarabad, Pakistan