SN Computer Science (2024) 5:595
https://doi.org/10.1007/s42979-024-02942-x

ORIGINAL RESEARCH

Advancing Software Vulnerability Scoring: A Statistical Approach with Machine Learning Techniques and GridSearchCV Parameter Tuning

Birendra Kumar Verma[1,2] · Ajay Kumar Yadav[1]

Received: 17 January 2024 / Accepted: 30 April 2024
© The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd. 2024

Abstract
The growing complexity, diversity, and importance of software pose a significant threat to computer system security due to exploitable software vulnerabilities. Important infrastructure systems, including banking, electricity, healthcare, and the military, are at risk of loss due to these vulnerabilities that permit unwanted access. This study investigates statistical features that contribute to improved outcomes, even though existing approaches primarily utilize natural language processing for vulnerability descriptions. We present an innovative scoring method that incorporates six well-known machine learning techniques: Linear Regressor, Decision Tree Regressor, Random Forest Regressor, K Nearest Neighbors Regressor, AdaBoost Regressor, and Support Vector Regressor into its framework. An assessment is conducted on 159,979 vulnerabilities obtained from the National Vulnerability Database using six metrics: explained variance, mean absolute error, mean squared log error, R-squared, root mean squared error, and mean squared error. GridSearchCV and tenfold cross-validation have validated the Random Forest Regressor as superior, yielding an accuracy of 0.9486. This approach demonstrates promise in proactive risk management across multiple sectors, including healthcare, energy, defense, and finance, by integrating machine learning techniques and statistical features.

Keywords Statistical technique · GridSearchCV · Software vulnerability scoring

Introduction

The software has evolved to become ubiquitous, complex, and incredibly varied. Although some of the world's brightest minds are engaged in software coding, even the most skilled programmers can make mistakes that leave their products vulnerable to attack. Attackers can oppress these faults or weaknesses in software design to obtain system privileges. As a result, software vulnerabilities provide an entry point for attackers into the system. A list of known security vulnerabilities in various software programs and open-source libraries is compiled in CVE. A special number called a CVE ID is used for the unique identification of each vulnerability in this list, which can be used by anyone eager to learn more about a specific vulnerability. CVE IDs are assigned by both MITRE and CNAs, which are a group of vendors and researchers from many different countries worldwide. The vulnerability is then examined by the US NIST, whose Common Vulnerability Scoring System provides both a severity score and eight widely used security attributes of the vulnerability. The analyst of the NVD uses the formula specified in CVSS v3.1 to assign a Base Score to each identified vulnerability. Software vulnerability distributions by severity are shown in Fig. 1 National Vulnerability Database for 2001–2022 (NVD).

The distribution of CVSS scores with low, medium, high, and critical (CVSS version 3.1 range of 9.0–10.0) severity vary over time, with roughly sixteen times more distribution in 2022 than in 2001.

In addition, the number of reported vulnerabilities has risen steadily over the past 5 years, from 14,645 in 2017–20,156 in 2021; this represents an increase of 136.63%. The MSRC found that high-severity software vulnerabilities climbed by 41.7% in 2nd half of 2015, accounting for 41.8

✉ Birendra Kumar Verma
birendraverma74@gmail.com

Ajay Kumar Yadav
ajaykyadav@banasthali.in

[1] Banasthali Vidyapith, Jaipur, Rajasthan, India

[2] JSS Academy of Technical Education, Noida, UP, India

Published online: 28 May 2024

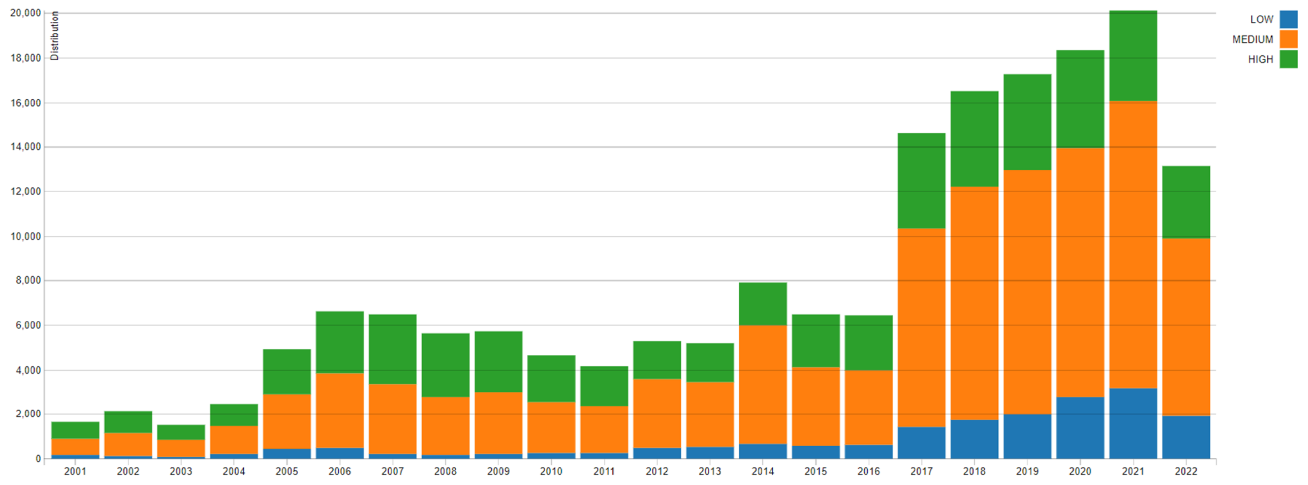


Fig. 1 Time-varying vulnerability severity distribution

percentage of the greatest in 3 years. The CVE database recorded a total of 20,161 vulnerabilities in 2021, up from 18,375 in 2020 and 17,308 in 2019. The number of reported vulnerabilities increased by roughly 58% between 2019 and 2020. Comparing 2021 to the previous year, there was a 65% rise in third-party component vulnerabilities. In addition, Skybox Security [1] revealed that the number of reported vulnerabilities has been on the rise. From 2020 to 2021, the number of vulnerabilities rated high or critical rose from 1100 to 1200, with Microsoft coming in second place for the disclosure of such vulnerabilities. It is claimed in the study that software flaws like information disclosure and denial-of-service attacks can have serious consequences for software architectures. A hacker can leverage these flaws to get even more access to a compromised system, even if the company has released an update to address the flaws. From 2012 to 2021, the distribution of vulnerability types Fig. 2 depicts an increase in no and percentage of unique vulnerabilities, such as miscellaneous and yet-to-be-assigned CWE (labeled "MISCELLANEOUS").

Figure 3 CWE-787, showed significant changes from 2016 to 2021, while CWE-79, CSS, and CWE-No Info-Insufficient Information showed the majority of changes from 2012 to 2021

Holm et al. [2], tested CVSS scores with 384 experts covering over 3000 vulnerabilities. The CVSS Base Score and expert evaluations conflict by -0.38, with a variation of 4.46 (on a scale from 0 to 10). The experts proposed many CVSS changes to explain this disparity. In a different piece of research, Khazaei et al. [3] were able to get the CVSS Base Score from the OSVDB database and the vulnerability reports from the CVE database. Next, they developed a feature vector for nine vulnerability reports for key terms. They used linear discriminant analysis to handle the large

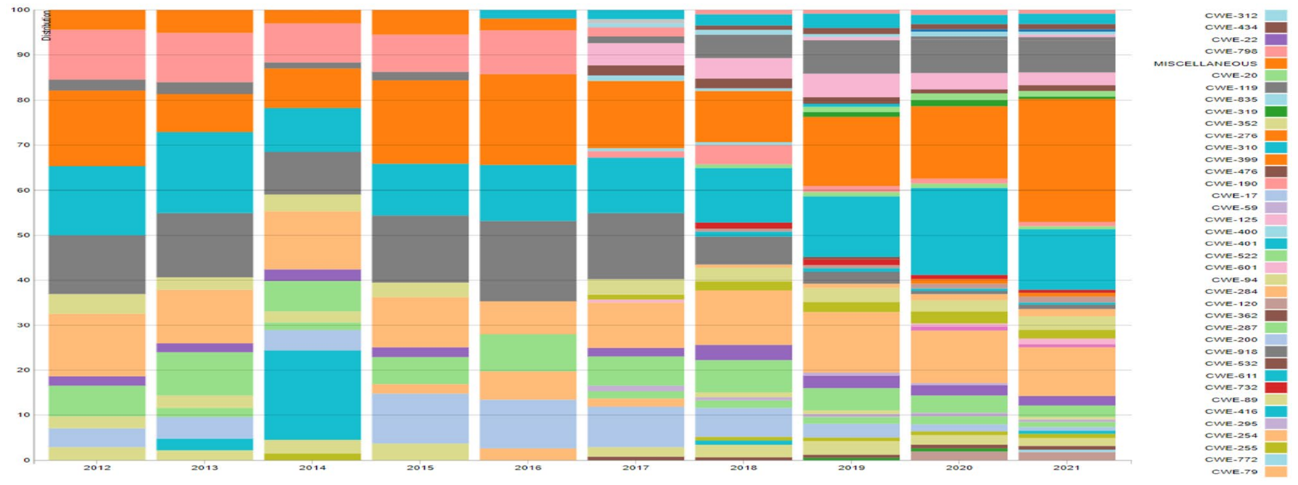


Fig. 2 The cumulative yearly number of CWE-identified vulnerabilities

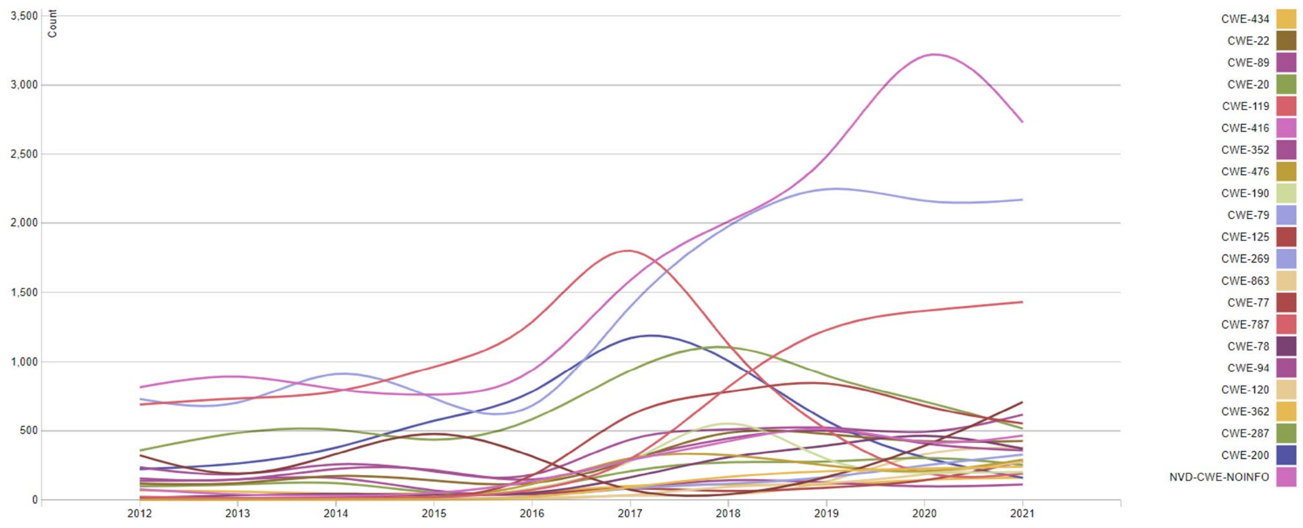


Fig. 3 The evolution of CWE assignments over time

Table 1 Fundamental statistical features

Attack complexity	Attack vector	Availability impact	Confidentiality impact
Integrity impact	Privileges required	Scope	User interaction
Vector string	Exploitability score	Impact score	Base severity
Access complexity	Access vector	Authentication	Availability impact. 1
Confidentiality impact. 1	Integrity impact. 1	Obtain all privilege	Obtain other privilege
Obtain user privilege	User interaction required	Vector string. 1	Exploitability score. 1
Impact score	Severity	Published date	Last modified date

dimensionality of the feature space. To determine how critical a software vulnerability is, they employed three machine learning methods. Spanos et al. [4] propose a multi-target method for identifying vulnerability traits and calculating severity scores. Their methodology first forecasts the characteristics of the vulnerability and then uses those predicted characteristics to determine the severity ratings for the vulnerability. The results of the experiments show that the classification accuracy was significantly improved. Despite expanding the study and increasing our understanding of vulnerabilities, they examined 23 months' worth of CVE numbers given by MITRE and discovered that NIST releases CVSS on average 132 days far ahead than it does for CVE numbers [5, 6]. Haipeng Chen et al. [7] Critical infrastructure systems, including energy, military, and healthcare, are being affected by the ongoing discovery of significant vulnerabilities and the increasing financial consequences that follow from these issues. Researchers in academia have been using new ML &DL methods to try to make more accurate models for scoring software vulnerabilities [3–7] Researchers are looking at vulnerabilities and scoring software vulnerabilities by mining vulnerability descriptions more [8]. This is because CVSS has problems with time delay and dependence on subject knowledge [3]. A short paragraph covers each vulnerability in great depth. This paragraph describes the type of vulnerability, the affected products and vendors, the versions that are affected, and the different statistical factors that an attacker needs to exploit the vulnerability, such as availability, integrity, attack complexity, privilege requirements, and more, as depicted in Tables 1, 2 and 3.

Table 2 Marginalized Database Metric Accuracy

Metric	NVD	X-Force	OSVDB	CERT-VN	CISCO
Availability	90	88	83	75	81
Confidentiality	90	88	81	67	84
Access comply	84	84	24	76	76
Integrity	92	90	84	73	85
Authentication	99	95	95	96	84
Access vector	99	98	97	85	97
Average	93	91	77	79	85

It also lists the key code components or inputs that are involved. Some results have been reported in earlier studies, but there are still problems in the following areas:

Table 3 Dataset Description

	Row0	Row1	Row2	………	Row159976	Row159977	Row159978
(CVE)IDs	CVE-2021–0204	CVE-2021–0205	CVE-2021–0206	………	CVE-1999–0001	CVE-1999–0002	CVE-1999–0003
Attack complexity	LOW	LOW	LOW	………	LOW	NaN	NaN
Attack vector	LOCAL	NETWORK	NETWORK	……….	NETWORK	NaN	NaN
Availability impact	HIGH	HIGH	HIGH	………	HIGH	NaN	NaN
Confidentiality impact	HIGH	NONE	NONE	………	HIGH	NaN	NaN
Vector string	CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:N/I:N/A:L	CVSS:3.1/AV:A/AC:P/PR:R/UI:N/S:N/C:Y/I:NA/A:M	………	NaN	NaN	NaN
Exploitability score	1.8	3.9	3.9	………	3.9	NaN	NaN
Impact score	5.9	3.6	4	………	5.9	NaN	NaN
Base severity	HIGH	HIGH	HIGH	………	CRITICAL	NaN	NaN
Vector string.1	AV: L/AC: L/Au:N/C:C/I:C/A:C	AV:N/AC:M/Au:N/C:N/I:N/A:P	AV:P/AC:N/Au:Y/C:A/I:P/A:N	………	NaN	NaN	NaN
Exploitability score.1	3.9	10	8.6	………	10	10	10
impact score	6.4	2.9	2.9	………	6.4	10	10
Scope	UNCHANGED	UNCHANGED	CHANGED	………	UNCHANGED	10	10
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
Description	A sensitive information disclosure vulnerabil-ity…	When the "Intrusion Detection Service" (IDS) f…	Snoopy before 2.0.0 has a security hole in exe…	…………	ip_input.c in BSD-derived TCP/IP imple-mentation…	Buffer overflow in NFS mounted gives root access…	Execute commands as root via buffer over-flow i…

(i) When assessing software vulnerabilities, don't think about the various statistical features, including but not limited to confidentiality, availability, exploitability, attack complexity, privilege requirements, and so on.
(ii) When picking classifiers, don't think about how the features that were taken from independent features depend on each other.
(iii) When looking for vulnerabilities in software, don't think about the specific score but rather the class to which software vulnerabilities adhere.

The recent fast growth of wrong access control and other software disruptions shows the need for software vulnerability scoring models that can handle changing technologies and attack methods in infrastructure systems that are cyber-enabled. As a result, this paper's main objective is to present a software vulnerability scoring method that addresses the mentioned issues. To summarize, this study presents the following contributions:

1. We first identified the fundamental statistical features of distinct software vulnerabilities and encoded them into their different categorical features.
2. We implemented principal component analysis to extract high-level attributes from a wide variety of categorical features and reduce dimensionality.
3. Our innovative software vulnerability assessment approach improves the accuracy and reliability of vulnerabilities retrieved from large datasets, such as CVE and NVD, by using statistical features and machine learning techniques.

Related Work

This section provides context for the research by reviewing relevant prior studies. The second version of CVSS was suggested and evaluated by Peter Mell and Karen [9] for its strengths and weaknesses. Using large datasets, experiments were conducted on both CVSS v1 and CVSS v2, and their characteristics were compared. Some system modifications simplified things without significantly affecting scoring, while others helped get us closer to our goals. Wang et al. [10] came up with a way to measure the effects of system flaws by capturing and using the basic features of vulnerability analysis, security mechanisms, and other related topics. The OVM has been filled with rules and programs that find out what they know to make it a reliable and efficient security system for all vulnerabilities. To forecast when an app might be vulnerable again, Su Zhang et al. [11] used a data mining strategy on NVD data. They also employed a machine learning algorithm to make predictions. In addition to suggesting potential new applications for the NVD data, the authors offer arguments as to why their present method failed to produce an appropriate prediction model for TTN vulnerability. They used the WEKA tool to execute various ML approaches and evaluate their effectiveness; the findings were CC = 0.1974; RMSE = 493.6879; RRSE = 694.7868; and CC = 0.1974. Researchers Xiao et al. [12] looked into security holes in popular deep learning frameworks such as Torch and TensorFlow. In contrast to deep learning models, which have very small code sizes [13], these frameworks are quite complex and reliant on numerous external components. These frameworks can be exploited by attackers to execute Denial of Service (DOS) assaults that cause the program to freeze, crash, or potentially compromise the system. This paper improves the security of frameworks for deep learning. One method for anticipating when a vulnerability will be exploited was developed by Chen et al. [7] and is called Vulnerability Exploit Scoring and Timing. They predict its characteristics and score. Despite using the Flask framework as its foundation, this approach enhances its performance by utilizing alternative deep-learning frameworks. Manual procedure-based vulnerability characteristic assignment was taken to a new level by [4]. They did this by making a model that used both text analysis and multi-target categorization techniques. This model provides the estimated vulnerability score and vulnerability characteristics. The dataset has 99,091 records, and the results made vulnerability scores and prediction accuracy much better. In addition to using Bayesian networks for classification, this model also made use of Decision Trees, Random Forests, and Boosting. The RF technique is significantly more effective than the other two methods. Using the F-measure, the Random Forest model does much better than the other two methods in five out of six cases. DT is the least accurate classification method five out of six times. Calculations of the CVSS and WIVSS [14] were carried out with the help of a Random Forest classifier. The estimated values from projected attributes are close to the actual mean values: 5.22 for the CVSS and 4.98 for the WIVSS versus 5.5 and 5.95, respectively. To address the issue of software vulnerabilities discovered on NVD sharing the same taxonomy, [15] developed a vulnerability classification system that was based on text clustering. The bisecting means, the sample mean, and the bathos clustering methods all use the Cluster Overlap Index, or COI, to calculate the clustering results. Only 45 vulnerabilities were selected from a total of 40,000 records, as indicated by the Descriptor Dominance Index. To transform their works into research on vulnerability taxonomies, they make use of these taxonomies of vulnerabilities. On NVD, the Simple K Mean, Bisecting K Means, and Kohonen Map text cluster operations are carried out; the cosine vector distance type is used, and the maximum number of iterations that can be performed is fifty. The Bisecting K Means algorithm is a straightforward partition clustering algorithm that uses bisecting. However, it has the maximum overlap indices, which indicates that the majority of clusters overlap. The Simple K Mean algorithm is the best clustering algorithm out of the three because it has the smallest average distance within a centroid and the largest average distance between centroids. It also has the lowest overlap indices. Wijayasekara et al. [16] constructed a binary classification model using the TF-IDF model as part of their text-mining strategy. The experiment reveals that the proposed approach can classify vulnerabilities. The studied classifiers had low accuracy. They advised researchers to improve feature extraction. Using 5 ML methods or apps, including 27,248 security vulnerabilities. Chen et al. [17] described a way to rate the seriousness of a vulnerability using IG feature selection and TF-IGM (term frequency-inverse gravity moment). They used a large dataset with 27,248 security flaws to build the framework. To improve its vulnerability classification accuracy, the TF-IGM model incorporates a unique statistical technique known as feature selection. The most powerful machine learning algorithms, such as KNN, RF, SVM, NB, and DT, were used to make the proposed classification from all of the old vulnerability reports that were found. Anjum et al. [18], proposed a method for determining the relative attack vulnerability of sensitive network protection services. The fuzzy Best Worst Method is used to prioritize the vulnerabilities identified. SQL Injection (SQLI), Information Gain (IG), and Gain of Privileges are extremely serious vulnerabilities that must be addressed as soon as possible. It was also observed that fuzzy BWM offers significantly greater precision and consistency than crisp BWM. Ruohonen et al. [5], looked into the time between when CVEs are added to NVD and when the data that goes with their CVSS scores is added. The findings of a regularized regression analysis conducted on more than 80,000 retrieved vulnerabilities. CVSS content has little statistical impact on time delays, which are driven by a diminishing annual trend. In addition to such results, the report presents insights on empirical methodological misrepresentations. This helps to fill in some gaps in the previous research. Deep learning was used by Vishnu et al. [19], to come up with an idea for a way to classify vulnerabilities. A self-attention deep neural network (SA-DNN) model and a text mining technique are used in the proposed system to classify vulnerabilities based on how they are written. 134,091 vulnerability reports from the CVE details website are used to test how well an SA-DNN-based system for classifying vulnerabilities works. Experiments showed that the proposed method is effective and that the SA-DNN model is superior to the SVM model, the GCNN model, the CNN-Bi LSTM model, and other deep learning techniques.

Dataset Description

The data used in this study came from the NVD, which is thought to be the largest and most complete database of vulnerabilities. The NVD has been compiling vulnerability records since 2002, and it now contains more than 159,979 such records. The Computer Security Division of the National Institute of Standards and Technology and the United States Computer Emergency Readiness Team (US-CERT) of the Department of Homeland Security worked together to make the National Vulnerability Database (NVD). Each entry in the NVD that describes a vulnerability contains a significant amount of information about that vulnerability. This includes the CVE number, exploitability score, vector string, impact score, severity, description, and so on, which is the practical explanation of the vulnerability. A CVSS score can be written as a number or as a “vector string,” a textual representation of the score’s values. The used schema is a string of 8 dimensions, with each dimension representing a different vulnerability metric: (CVSS: 3.1/AV: N/AC: L/PR:H/UI: N/S: U/C: L/I: L/A: N). The severity of the vulnerability is reflected in the ranking of the characteristic values. When it comes to rating and prioritizing vulnerabilities, the information technology security community universally agrees on the CVSS. The fact that CVSS was chosen as the official score for the NVD shows how well-known the system is. CVSS was first released in 2005, and its final vulnerability score is based on nine different factors. The most recent update to the CVSS, version 3.1, was made available for download in June 2019. CVSSv2.0 and CVSSv3.1 are now used concurrently by NVD. Both were chosen because they were included in the comprehensive NVD dataset used for this research. The base metrics produce a score ranging from 0 to 10, which can then be modified by including extra scores for the temporal and environmental metrics. The final score will be in the range of 0–10. The Base Score remains constant regardless of the passage of time or the user’s context. When a vulnerability is successfully exploited, the impact metrics record the worst-case scenario for the affected component as mentioned in the equation and the scope metric is a way to figure out if a weakness in one system or part can affect another system. The severity of a security vulnerability grows when exploit code can be easily obtained by the public and used by inexperienced attackers. A vulnerability’s ‘attack Vector’ is a descriptive term for the type of access that an adversary would need to exploit it. When compared to a vulnerability that can be used from far away over the Internet, one that requires physical access to the intended system is much harder to use. A vulnerability's Attack Complexity score reflects the difficulty with which an attacker can exploit it under arbitrary circumstances. This usually involves some sort of participation on the part of the user or a predetermined setup for the intended system. The Privileges Required metric specifies the minimum required permissions for an attacker to exploit the flaw. If no special permissions are needed, then the Base Score will be at its highest. The User Interaction metric shows how much a vulnerability can be taken advantage of without the help of other users. The formula for the base metrics of CVSS v3.1 relies on the sub-function for the ISS, the impact, and the exploitability, each of which is defined below.

$$\begin{aligned}\text{ISS} &= 1-\left[(1-\text{Integrity})\times(1-\text{Confidentiality})\right.\\ &\quad \left.\times(1-\text{Availability})\right]\end{aligned} \tag{1}$$

$$\begin{aligned}\text{Impact} &= \text{If the scope does not changeISS}\\ &\quad *\ 6.42\text{If Scope Changed }(\text{ISS}-0.029)\\ &\quad *\ 7.52-3.25*(\text{ISS}-0.02)^{15}\end{aligned} \tag{2}$$

$$\begin{aligned}\text{Exploitability} &= 8.22*\text{Attack Vector}\\ &\quad *\ \text{Attack Complexity}\\ &\quad *\ \text{privileges Required}*\text{UI}\end{aligned} \tag{3}$$

where UI is User Interaction

$$\begin{aligned}\text{Base Score} &= \text{If Impact} <= 00,\\ &\quad \text{else If the scope does not change Roundup}\\ &\quad (\text{Minimum }[(\text{Impact }+\text{Exploitability}),\ 10])\\ &\quad \text{If Scope Changed Roundup}\\ &\quad (\text{Minimum }[1.08*(\text{Impact }+\text{Exploitability}),\ 10])\end{aligned} \tag{4}$$

When conducting research, why did we choose NVD as our dataset of choice? prioritizing vulnerabilities based on severity can help the industry save time and money. For vulnerability management purposes, many private-sector industries rely on CVSS internally and fix the ones that present the greatest danger to their systems. Therefore, reliable vulnerability data is essential for setting priorities [20]. When one takes a look at the findings of the Bayesian analysis of the CVSS that was carried out by [21], it is very clear that NVD achieves excellent results. Using only one metric eliminates the need to worry about database accuracy.

The dataset has been taken from the website [22] in the form of a JSON file. Python script has been used to convert it into a CSV file. Currently, the CSV file is being processed in the Panda's data frame. There are 32 columns and 159,979 rows to start. Each of the 32 columns has its own set of unique (CVE) IDs, such as CVE-2021–0202, and its own set of unique categorical features, such as attack complexity, attack vector, availability impact, confidentiality impact, and so on. In a pre-processing step, every categorical feature has been changed into separate columns such as attack complexity changed into attack Complexity High and attack complexity is Low because it has two categories High and Low. Similarly, every mandatory feature has been changed into the category they have.

Methodology

We are working with statistical data, and our target feature is continuous. As described in the section on the system architecture, we are doing regression analysis of different regressors on predetermined datasets.

Data Pre-Processing

We used the NVD dataset, which included information from the years 2001–2021 and is displayed in Fig. 1. In the first step of the statistical method, we look at how the dataset is set up. This includes counting the number of rows and columns and figuring out what kind of data it is. In its initial state, it consists of 159,979 rows and 32 columns, of which 26 columns have an object datatype and 6 columns have a float 64 datatype**.** The primary terms included are "CVE," "Attack Complexity," "Attack Vector," "Availability Impact," and others. The dataset includes CVSS2.0 and CVSSv3.1 values, which explains why nearly identical values appear for features like 'Availability Impact" and 'Availability Impact. 1', 'Confidentiality Impact' and 'Confidentiality Impact. 1', and so on. Table 3 with the columns 'Vector String' and 'Vector String. 1' could also be used to confirm this. Twenty-six of the thirty-two features have categorical values, such as the feature 'attack complexity', which can be either 'low' or 'high'. In the same way the 'Confidentiality Impact' has three categories that are 'HIGH', 'LOW', and 'NONE'. Certain features have been removed because we are using statistical information because they do not contribute. When some rows in a dataset have missing values, the median values in that column are substituted. Because the machine learning model can only understand binary numbers and not text, categorical features are turned into binary numbers by using dummy variables. One hot encoding can also be used to turn categorical variables into binary ones and zeros, but it has a serious problem called multicollinearity. There is a total of 159,979 rows and 54 columns in the dataset, and a heatmap is immediately available in Table 3 to show the degree of association between the various features. We used the NVD information to score the severity of software vulnerabilities. The remaining columns serve as independent variables in the method we proposed. To simplify the data without losing key characteristics, we used PCA to discover patterns in the input layer's fifty-four independent features and then distilled the variables down to their most important properties. Principal component analysis has given us thirty different features we can use to build regression models. The approach that has been suggested is based on the idea that we can tell more about the different types of vulnerabilities by looking at their scores (Fig. 4).

Multi-Collinearity of Vulnerability Attributes

In a multiple regression model, multicollinearity occurs when an independent variable has a high degree of correlation with another or more of the independent variables. The multi-collinearity of vulnerability attributes, as represented in Fig. 5, represents the linear relationships between independent attributes. Figure 5 shows color codes from blue to red that show how different vulnerability traits are related. Blue codes show weak relationships, while red codes show the strongest relationships. As an example, the exploitability score contains values of 1, 0.79, and 0.65 for exploitability score, exploitability score.1, Attack Vector Network, respectively, means these attributes are highly correlated undermining the statistical significance as depicted in Fig. 5, column 1.

Optimal Parameter Selection

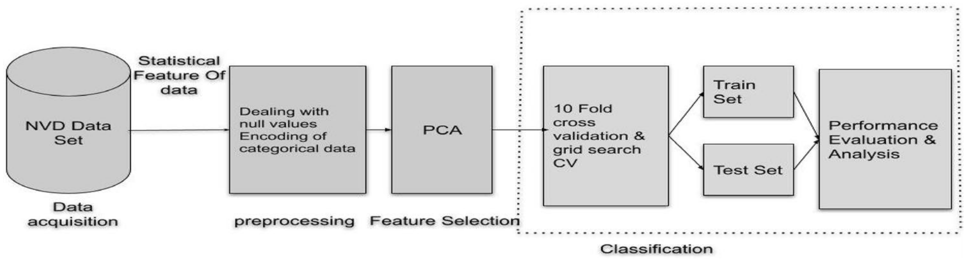
As was talked about in the preprocessing module about getting the data ready for analysis, the next analysis will use thirty features. We used a method called "tenfold cross-validation" to see how well our machine-learning model did with data it had never seen before [17]. This method divides the information set into 10 folds, using nine folds for training and one onefold for validation each time. After ten iterations, the results are averaged to come up with a final value. GridSearchCV is also part of the model and is used to find the best parameter in the grid of parameters given before making predictions. The accuracy of a model's predictions depends a lot on the values of its hyperparameters. Since it is impossible to know in advance which hyperparameter values will yield the greatest results, we must try them all to see which ones work best. Looping through the predefined hyperparameters is made easier with the assistance of this function, which also assists with fitting the model to the training set. In conclusion, we can choose the best parameters by looking at the list of hyperparameters. We gave the GridSearchCV function some predefined values for its hyperparameters, as was already said. The way we did this was by defining a dictionary that lists each hyperparameter and the possible values for that parameter. GridSearchCV goes through all of the possible combinations of the dictionary's values one at a time before using the Cross-Validation method to check how well the model works for each of these combinations. Every possible combination of hyperparameters can have its profit or deficit calculated using this function. Subsequently, we can choose the alternative that provides the most optimal performance.

Proposed Classification Models

Random Forest Regression

We use the Random Forest technique for supervised learning to perform predictive modeling. It is a group learning method for regression called ensemble learning. The name of the algorithm hints at how unpredictable it is. Sample data for each iteration is picked at random, just like in the bagging algorithm. Random subspace approach for building each classification tree's target variable only two parameters, the of trees and the no of target variables, need to be established and tuned [23]. To fine-tune the prediction, we tried out several different tree sizes but ultimately settled on 700.

Fig. 4 System architecture


NVD Data Set
Data acquisition
Statistical Feature Of data
Dealing with null values Encoding of categorical data
preprocessing
PCA
Feature Selection
10 Fold cross validation & grid search CV
Train Set
Test Set
Performance Evaluation & Analysis
Classification

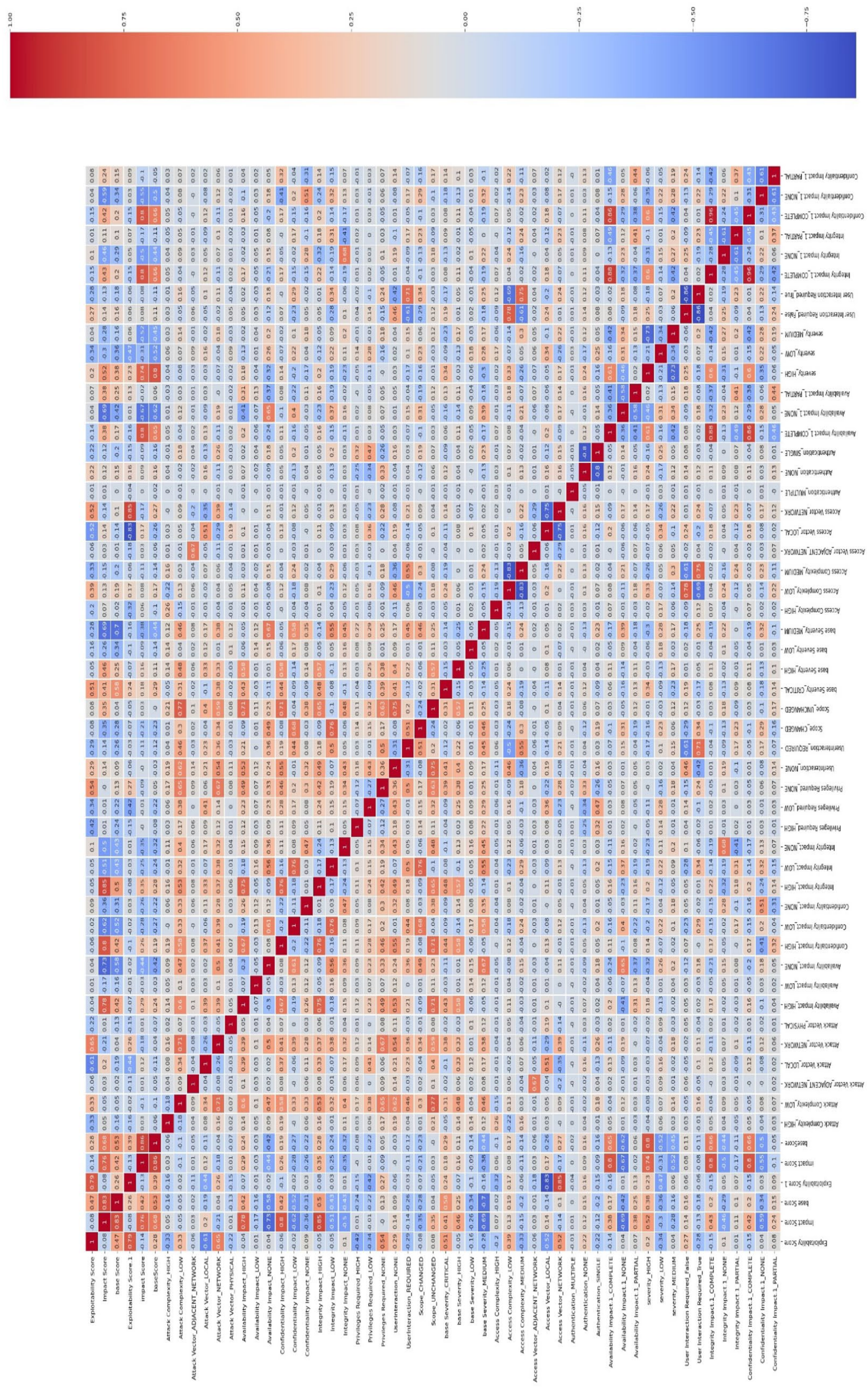


Fig. 5 Multi-collinearity of vulnerability attributes

Decision Tree Regression

As its name suggests, the Decision Tree regressor takes the form of a tree graph, and it is one of the most widely used classifiers [24]. It serves as the foundation of many regressors, including AdaBoost and R. F. The DT is also great at working with data having different categories, and it can easily deal with both single data points and large gaps in the data. As a result, the parameter tuning was done based on the results that were averaged after 10 rounds of cross-validation.

Linear Regression

In linear regression, the sigmoid function classifies data into distinct categories. This makes linear regression a generalized linear model. The objective of linear regression analysis is to establish a mathematical model that represents the linear relationship between the input variables (X) and the investigated outcome (Y) [25]. To fine-tune the parameters, we used a tenfold method on the data responsible for training. Because of this, the tuning was done based on the average results from ten rounds of cross-validation. Can tune other metrics like MSE and negative mean squared error, as an example.

K-NN Regression

It employs a learning approach that is independent of parameters and applies to problems involving classification and regression [24]. A lazy learner algorithm stores the dataset and uses it during classification, which is why it gets its name. The K-Nearest Neighbors approach is designed to identify the most likely main class for a set of test cases based on a subset of K training examples. It compares the training set to the test set using a similarity metric like the Minkowski metric. This classifier was chosen because it can classify unknown instances based on their closest neighbors and can be set up quickly and easily. To figure out how well the regressor works, GridSearchCV and stratified tenfold cross-validation are used.

Support Vector Regression

It's a powerful statistical machine-learning technique with wide-ranging practical applications. In actual use cases, it is primarily employed for classification jobs; however, it can also provide useful support in regression circumstances. The support vector machine was chosen because [24] showed that it outperformed the Naive Bayes classifier. To do this, we chose a good value for C and used the radial basis function (RBF). Parameter tuning was based on 10 rounds of cross-validation results.

AdaBoost Regression

AdaBoost is a sequential ensemble method that uses random training subsets to make several learners with less skill [26]. Weights are used in each training to learn each hypothesis. To determine the extent to which the hypothesis is incorrect for the entire dataset, the weights indicate the importance of individual cases. After individual iterations, features are recalculated to give greater importance to instances that were misclassified by the most recent hypothesis. It frees up the algorithm's resources to be applied to more challenging learning problems. The most important part of the algorithm is reweighting the incorrectly classified examples. In regression, the results are not true or false, but rather a real value error. This is different from classification, where the results are either true or false. When the Ada Boost classifier is used, the error is compared to a prediction error threshold.

Metrics for Performance Evaluation

To determine which Regressors were the most effective, we utilized the following metrics: [27].

Principal Component Analysis

PCA condenses large datasets without compromising data quality. For large data sets with many features, principal component analysis (PCA) is invaluable. The base principle is to reduce the number of dimensions in data while maintaining as much information as possible [28].

Explained Variance

Importantly, it measures how much the explained variance of the model's predictions accounts for the observed variation. This means that the anticipated value differs from the expected value. Since there is a big chance of losing substantial information during dataset reconciliation, it is critical to have a solid grasp of this idea.

Negative Mean Squared Error

Using the "negative mean squared error" score in the test set will produce -ve results. For example, if mean squared error is 5, it would be interpreted as -5. Similarly, a Mean Squared Error (MSE) value of 9 would correspond to a negative value of -9. The cross-validation function's goal of maximizing scores motivates this strategy. Scorer objects are usually designed to guarantee consistent, non-zero results.

R- Squared (R^2)

An important component of determining the efficiency of a regression-based ML model is the R^2 value [25]. A measure of the dataset's usefulness is how much of the discrepancy between the predictions it can account for. Discrepancy occurs when the model's predictions differ from the dataset samples. A score of 1 for R2 indicates that the model performs well with fresh data, whereas values of 0 and 1 indicate that the model fails to accurately predict future data. We calculate an MSLE, an MAE, an R2, an RMSE, and an MSE using Eqs. 5–9.

$$\text{Mean Squared Log Error (MSLE)}L(a,\hat{a}) = \frac{1}{n}\left(log\left(a_i+1\right) - log\left(\hat{a}_i+1\right)\right)^2 \tag{5}$$

where $\hat{a}_i$ is predicted value

$$\text{R} - \text{Squared}\left(\text{R}^2\right) = 1 - \frac{\Sigma_i\left(a_i-\hat{a}_i\right)^2}{\Sigma_i\left(a_i-\overline{a}_i\right)^2} \tag{6}$$

$$\text{Mean Absolute error (MAE)} = \frac{1}{n}\sum_{i=1}^{n}\left|a_i-\hat{a}_i\right| \tag{7}$$

where a_i is the true value and $\hat{a}_i$ is predicted one

$$\text{Mean Squared Error (MSE)} = \frac{\sum_{i=1}^{n}\left(a_i-\hat{a}_i\right)^2}{n} \tag{8}$$

$$\text{Root Mean Squared Error (RMSE)} = \sqrt{\frac{\sum_{i=1}^{n}\left(a_i-\hat{a}_i\right)^2}{n}} \tag{9}$$

Experimental Results and Analysis

Figure 10, shows an R^2 value comparison of the Linear, DT, RF, KNN, Ada Boost, and SVR for Train, Test, 10-Fold, and GridSearchCV Results. Linear, DT, RF, KNN, Ada Boost, and SVRs' tenfold cross-validation outcomes for NMSE are depicted in Fig. 9. Linear regression, decision trees, random forests, k-nearest neighbors, AdaBoost, and support vector

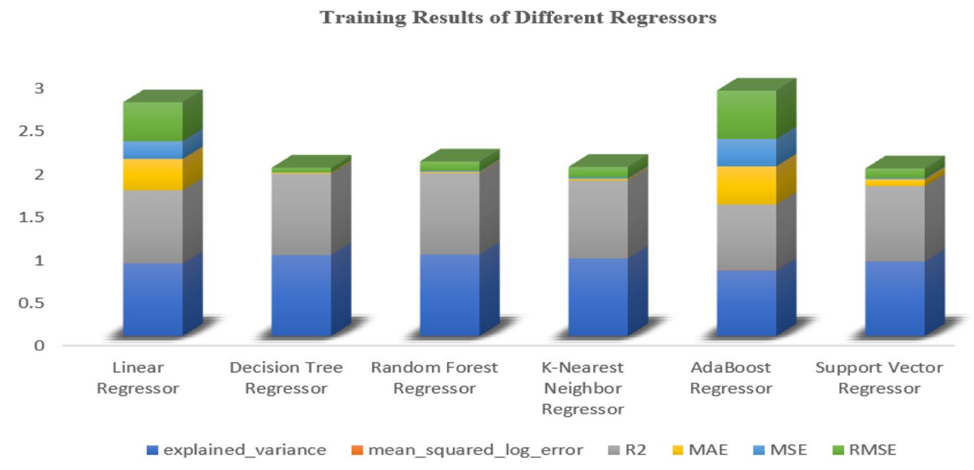


Fig. 6 Training results of different regression models based on metrics such as explained variance, mean squared logarithmic error, R-squared, mean absolute error, mean squared error, and root mean squared error

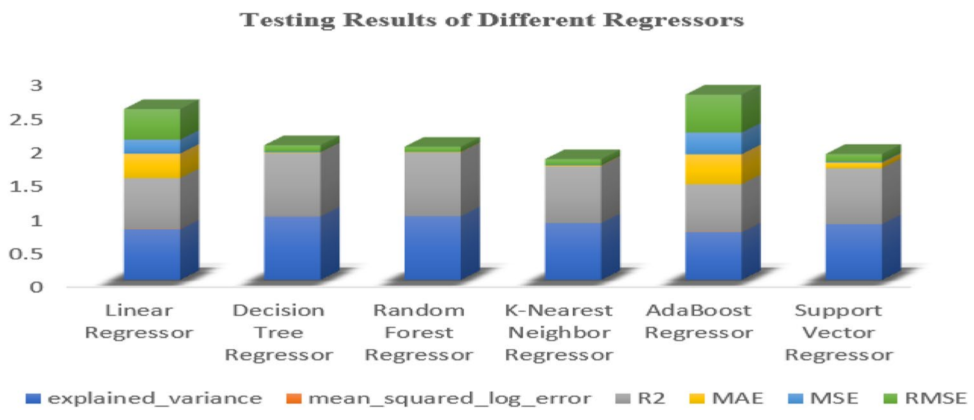


Fig. 7 Testing results of different Regression Models based on metrics such as explained variance, mean squared logarithmic error, R-squared, mean absolute error, mean squared error, and root mean squared error

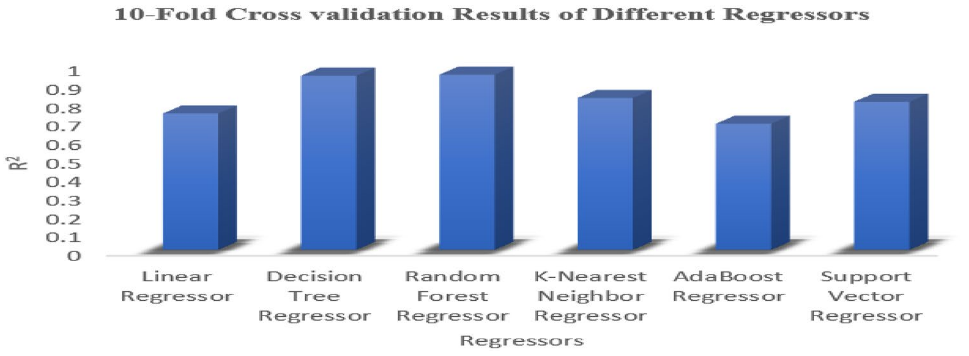


Fig. 8 Results of tenfold cross-validation for various regressors along with their corresponding R squared values

Regressors were evaluated using their R-squared values, and the results of a tenfold cross-validation are shown in Figs. 6, 7 and 8 display the training and testing results for the linear, decision tree, random forest, K-nearest neighbors, Ada boost, and support vector regression models, respectively. This study presents its findings using the following measures: an MSLE, an R-squared, an MAE, an MSE, and an RMSE, and explains variance. Linear regression, decision tree, random forest, k-nearest neighbors, Ada boost, and SVR are evaluated along with their respective levels of performance in Table 4. This evaluation method utilizes metrics such as explained variance, mean squared logarithmic error, R-squared, mean absolute error, mean squared error, and root mean squared error. The testing set comprises twenty percent of the data, while the training set consists of eighty percent. We use several different measures, such as RMSE, MAE, MSLE, MSE, and explained variance, to evaluate the effectiveness of each regression model. In this section, we outline and assess the additional findings.

Evaluation

A mean squared log error was only 0.0002. As for the most stringent R^2 measure, the outcomes mirror that for the RF, DT, KNN, SVR, Linear, and Ada Boost models, with values of 0.9459, 0.9427, 0.8442, 0.8297, 0.7537, and 0.7083, respectively. Here is an analysis of the individual regression results that were made so that a more complete evaluation of the suggested method can be made. Checking the mean error is essentially what we're doing when we compare the R^2 score to the Explained Variance score; if the two are equal, then the mean error is also equal to zero. The biased variance is utilized by the explained variance to calculate an explanation percentage. Both scores should be equivalent if the predictor's mistake is truly random.

By analyzing Table 4 and Fig. 7, we can observe the specific values of Explained Variance and R2 for the linear, DT, RF, KNN, Ada Boost, and SVR models. These values are as follows: 0.7538, 0.9427, 0.9459, 0.8442, 0.7092, and 0.7537, 0.9427, 0.9459, 0.8442, 0.7083, and 0.297. According to the experiments' results, it appears that RF, DT, and KNN do not exhibit any type of bias because they have a similar score of Explained Variance and R^2 of 0.9427, 0.9459, and 0.8442, respectively. On the other hand, the linear, Ada Boost, and SVR models are biased because they have a distinct score of Explained Variance and R^2 of (0.7538, 0.7537), (0.7092, 0.7083), and (0.8298, 0.8297), respectively. The performance evaluation was based on a total of six different metrics. The Random Forest regressor did well in five of them, but it didn't do as well with an MAE metric, and the decision tree did better with this particular metric. The majority of an MSEs and an RMSEs for all regressors are found within the intervals (0.0055, 0.3198) and (0.0744, 0.5655), respectively, as shown by the results.

Figure 8 displays the tenfold cross-validation findings for the linear, decision tree, random forest, K-nearest neighbors, AdaBoost, and support vector regression models, expressed in terms of R^2 values. This shows that all of these regressors do well in tenfold cross-validation, but the Random Forest regressor did especially well on both the testing set and the validation set. In addition to this, we used a different score metric known as the negative mean squared error. We found that the decision tree performed better than all of the other regressors when we analyzed the values of the negative mean squared error shown in Fig. 9. negative mean squared error for the RF, DT, KNN, SVR, linear, and Ada Boost regression models are -0.0052, -0.0045, -0.0065, -0.0053, -0.2021, and -0.2688, respectively.

Random Forest outperforms Linear, DT, KNN, Ada boost, and SVR in R^2 values for train, test, tenfold, and GridSearchCV results, as depicted in Fig. 10. The GridSearchCV result for Random Forest is 0.9486, while its train score is 0.9494, its test score is 0.9459, and its tenfold score is 0.9461. Therefore, it can be concluded that the RF performed well across five of the six metrics used for performance assessment, based on the data shown in Table 4 and Figs. 8 and 10. The outcomes of GridSearchCV and tenfold cross-validation further supported these conclusions.

Table 4 The results are based on a split of 80% for training and 20% for testing

Evaluation Metrics	Training results of different regressors						Testing results of different regressors					
	Linear Regressor	D. T Regressor	R. F Regressor	K-N N Regressor	A. B Regressor	S. V Regressor	Linear Regressor	D. T Regressor	R. F Regressor	K-NN Regressor	A.B Regressor	S. V Regressor
Explained Variance	0.8465	0.9441	**0.9494**	0.906	0.7641	0.8705	0.7538	0.9427	**0.9459**	0.8442	0.7092	0.8299
Mean Squared Log Error	0.0038	**0.0001**	0.0002	0.0004	0.0045	0.0035	0.0038	0.0002	**0.0001**	0.0002	0.0046	0.0002
R^2	0.8465	0.947	**0.9494**	0.906	0.7632	0.8701	0.7537	0.9427	**0.9459**	0.8442	0.7083	0.8297
MAE	0.3646	**0.0115**	0.0125	0.0184	0.4441	0.0814	0.3661	**0.0074**	0.0076	0.0103	0.4459	0.0819
MSE	0.2058	**0.0029**	0.0119	0.0153	0.3176	0.0129	0.2067	0.0097	**0.0055**	0.0077	0.3198	0.0136
RMSE	0.4536	**0.0538**	0.109	0.1236	0.5635	0.1138	0.4547	0.0985	**0.0744**	0.0876	0.5655	0.1168

Best results are in bold

GridSearchCV Results and Analysis of Different Regressors

Linear Regression

As mentioned in the experimental design section, cross-validation using GridSearchCV is used to assess the accuracy of the Linear Regression thiry different features can be used as predictors. When using RFE to construct the model, we must specify the desired feature set. A feature-elimination method is then executed. To specify the range of hyperparameters to tune, use the syntax hyperparameters = n features to select a list {range (1, 30)}. The following data was returned by the function GridSearchCV: RFE (estimator = Linear Regression ()) {'n_ features _to_ select': 29}, best_ params_ (12), mean_ score _time (0.0062) s and best_ score obtained (0.7364), Optimal Number of Features selected is 12. Even though GridSearchCV gives back 21 predictors, we checked the results by making a graph between the test score and the training score as depicted in Fig. 11 and Table 5.

Decision Tree Regression

Cross-validation using GridSearchCV is used to assess the accuracy of the DT using folds = K Fold as mentioned in the experimental design section. Use hyperparameters With five folds for each of the 260 candidates, 1300 fits were obtained. The following data was returned by the function GridSearchCV: RFE 'max_ depth': 10, 'max_ features': 16, mean_ score _time (0.0079) s and best_ score obtained (0.9459). Even though GridSearchCV gives back 22 predictors, we checked the results by making a graph between the mean_ test_ score and mean_ train_ score as depicted in Fig. 12 and Table 5.

Random Forest Regression

As mentioned in the section on experimental design, cross-validation using GridSearchCV is used to assess the accuracy of the Random Forest Regression. The estimator = Random Forest Regressor scoring set to R^2 are the parameters passed to the GridSearchCV function. When the model was fitted with five folds for each of the 156 candidates, a total of 780 fits were obtained. The following data was returned by the function GridSearchCV: RFE 'max_ depth': 11, 'max_ features': 16, mean_ score _time (0.0219) s and best_ score obtained (0.9486). Even though GridSearchCV gives back 22 predictors, we checked the results by making a graph between the mean_ test_ score and mean_ train_ score as depicted in Fig. 13 and Table 5.

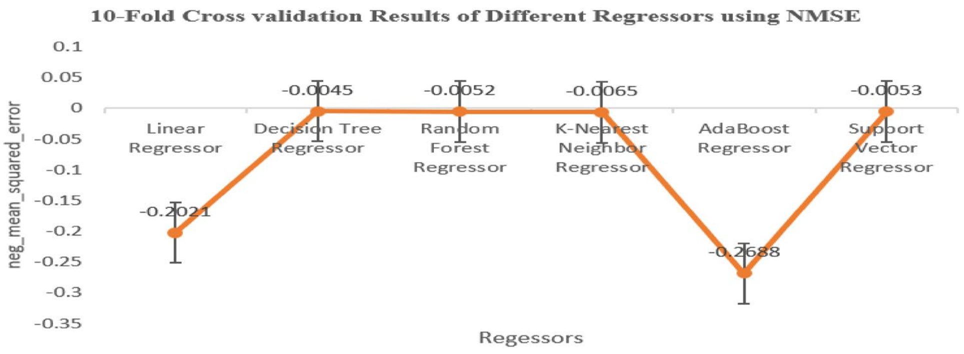


Fig. 9 Results of tenfold cross-validation for various regressors along with their corresponding negative mean squared error values

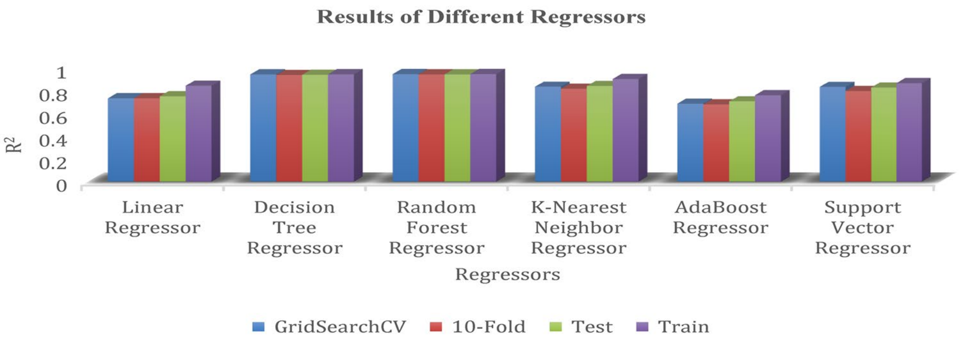


Fig. 10 Evaluation of R^2 values for different regressors by comparing the outcomes of Train, Test, 10-Fold Cross-Validation, and GridSearchCV

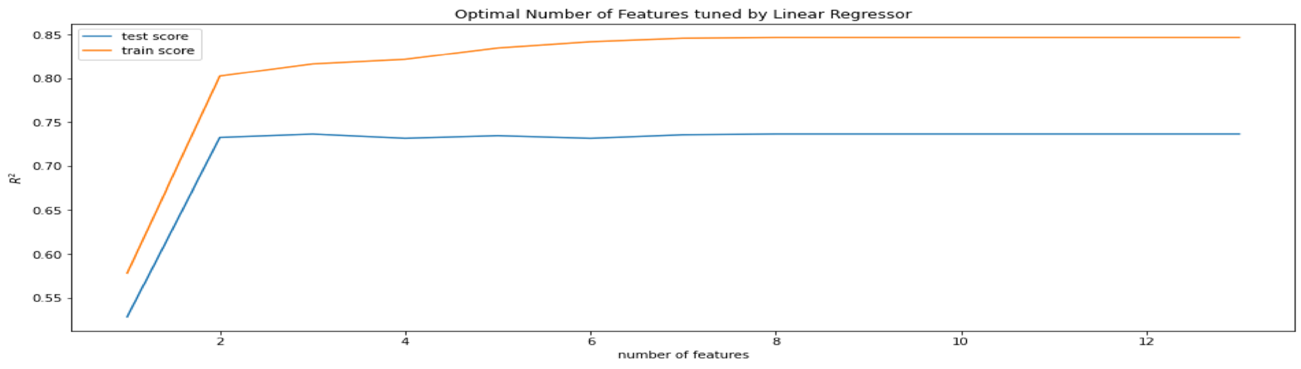


Fig. 11 Optimal value of R^2 for linear regressor (Train and Test)

Table 5 simulation parameter used in the experiment for GridSearchCV validation

Regressors	Feature selection algorithm	Best parameter	Mean score time	Best score obtained
Linear regressor	Recursive feature elimination	12	0.0062 s	0.7364
Decision tree regressor	Recursive feature elimination	16	0.0079 s	0.9459
Random forest regressor	Recursive feature elimination	16	0.0219 s	0.9486
K-Nearest neighbor regressor	Recursive feature elimination	21	0.1833s	0.8377
AdaBoost regressor	Recursive feature elimination	100	0.1827s	0.6877
Support vector regressor	Radial basis function	2	0.0246S	0.8358

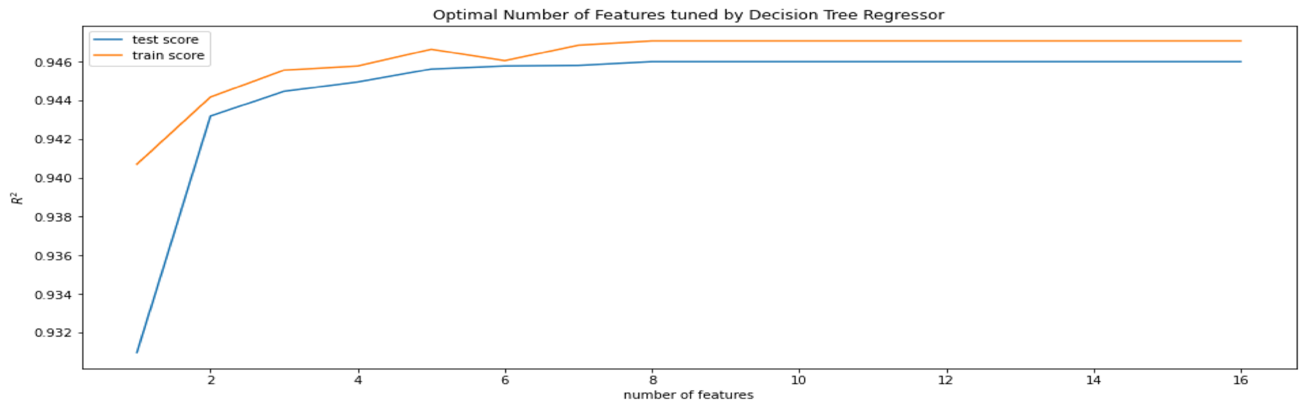


Fig. 12 Optimal value of R^2 for decision tree regressor (Train and Test)

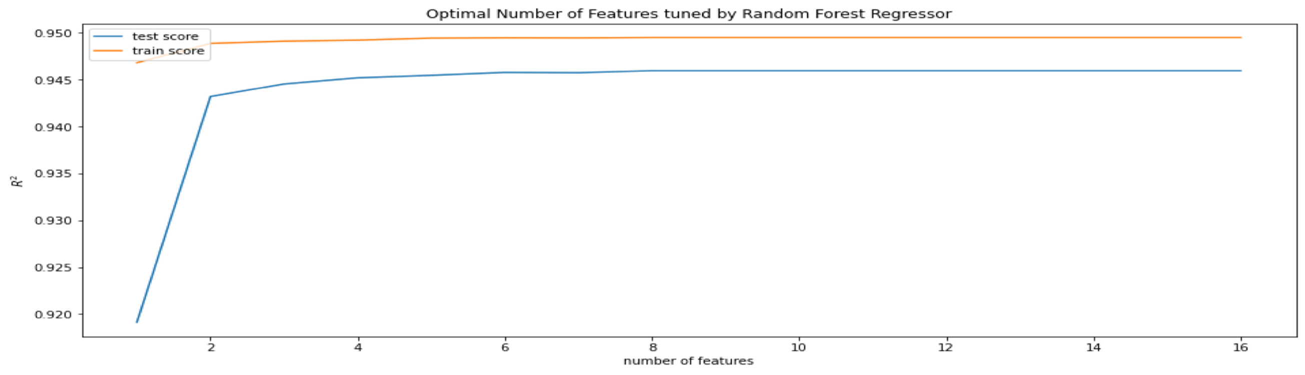


Fig. 13 Optimal value of R^2 for random forest regressor (Train and Test)

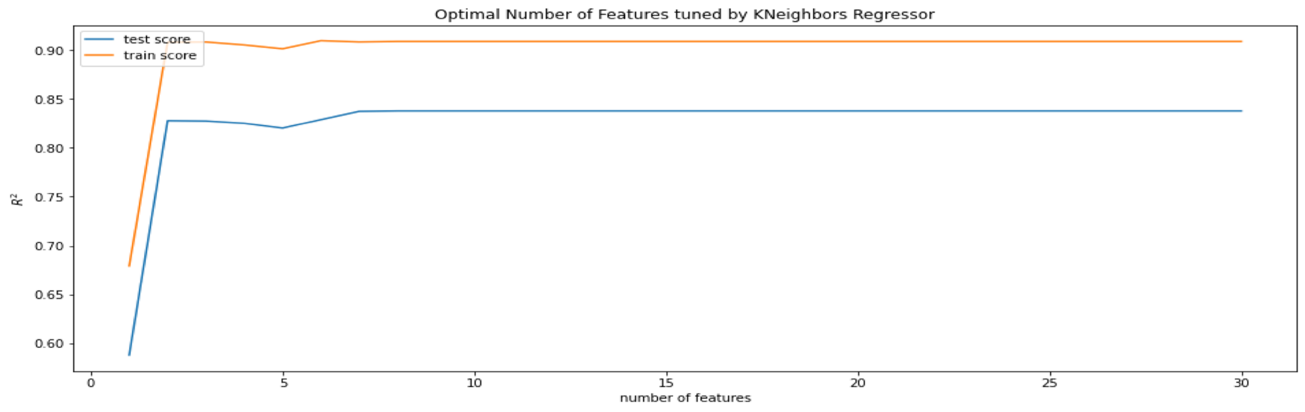


Fig. 14 Optimal value of R^2 for K-nearest neighbor regressor (Train and Test)

K-Nearest Neighbor Regression

Cross-validation using GridSearchCV is used to judge the accuracy of the KNN model, as mentioned in the section on experimental design. Twenty different features can be used as predictors. With 30 candidates, 150 fits were obtained when the model was fitted with five folds for each candidate. The following data was returned by the function GridSearchCV: best_ params ('knn_ n_ neighbors': 21), mean_ score _time (0.1833) s and best_ score obtained (0.8377). Even though GridSearchCV gives back 21 predictors, we checked the results by making a graph between the test score and the training score as depicted in Fig. 14 and Table 5.

AdaBoost Regression

We assess the accuracy of the Ada Boost regression model using gridsearchCV cross-validation, following the steps outlined in the experimental design section. This cross-validation strategy was calculated with the following parameters: estimator = RFE (Ada Boost Regressor (random state = 0, n-estimators = 100), parameter grid = hyper_ params{return-train-score = True}, scoring = R^2, verbose = 1, cv = folds. Thirty different features can be used as predictors. With 30 candidates, 150 fits were obtained when the model was fitted with five folds for each candidate. The following data was returned by the function GridSearchCV, mean score time (0.1827) s and best_ score obtained (0.6877). Even though GridSearchCV gives back 21 predictors, we checked the results by making a graph between the test score and the training score as depicted in Fig. 15 and Table 5.

Support Vector Regression

As mentioned in the section on experimental design, cross-validation using GridSearchCV is used to assess the efficiency of the support vector model. Through the use of preset hyperparameters, this function assists in fitting the model to the training set. Hyperparameters can help us choose the best parameters. As stated, we predefined GridSearchCV hyperparameters. We did this by defining a dictionary with each hyperparameter's possible values. Here's an example C {0.001, 0.001, 0.1, 1, 1.5, 2}, gamma {0.001, 0.01, 0.01, 0.1, 1}, kernel {RBF, C, gamma}, and kernels are hyperparameters, the rest have default values in an SVM model. GridSearchCV checks each possible dictionary value

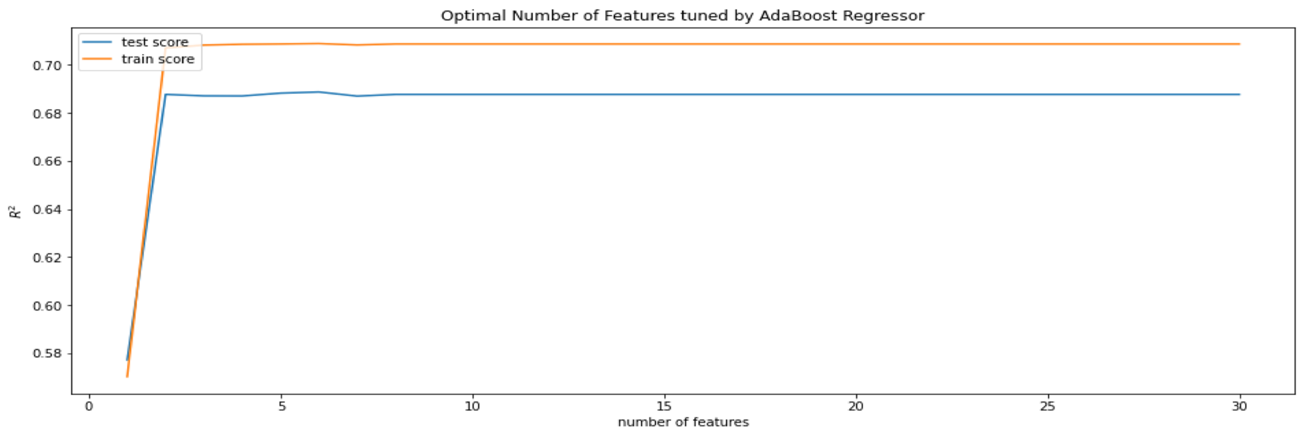


Fig. 15 Optimal value of R^2 for AdaBoost regressor (Train and Test)

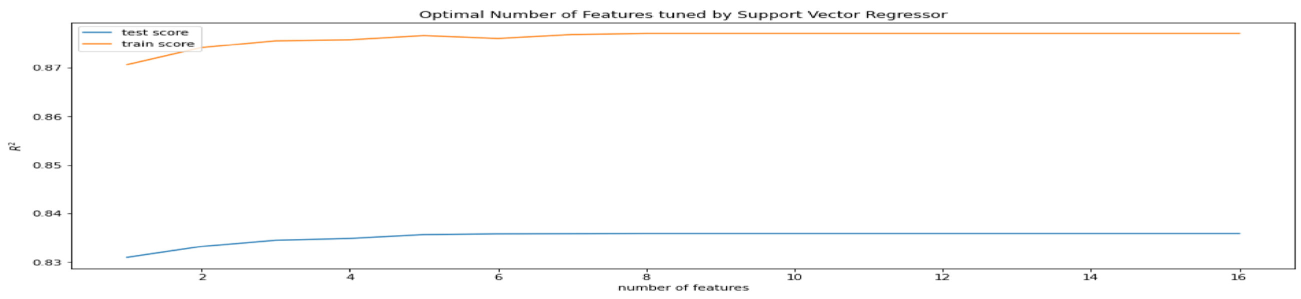


Fig. 16 Optimal value of R^2 for support vector regressor (Train and Test)

combination using cross-validation. This function returns the GridSearchCV: best_ params {'C': 2, 'gamma': 0.001}, mean_ score _time (0.0246) s and best_ score obtained (0.8358). Even though GridSearchCV gives back 21 predictors, we checked the results by making a graph between the test score and the training score as depicted in Fig. 16 and Table 5.

Comparison with the Most Recent Advances in the Field

Most of the techniques used for score vulnerabilities are based on vulnerability description only using natural language processing. Researchers have not yet investigated several important criteria that provide valuable indications. These include attack difficulty, vector, availability, secrecy, and integrity, privileges required, scope, user interaction, and others, as shown in Table 3. In the literature survey, we found two papers where vulnerabilities are scored, and performance is evaluated using metrics such as ours. Georgios et al. [14] evaluated the values from the calculated features that are rather near the real mean values, 5.22 CVSS value and 4.98 WIVSS, value whereas true mean values are 5.5 and 5.95, respectively. Zhang et al. [11] investigated how to predict when a software system might be vulnerable again by applying data mining techniques to NVD datasets. They employed the WEKA tool to deploy a range of machine learning approaches, assess their performance, and yield the following outcomes: The correlation coefficient is 0.19, the root mean squared error is 493.68, and the root relative squared error is 694.78.

We compared linear regression, decision trees, random forests, K-nearest neighbors, AdaBoost, and support vector regression models using explained variance, mean squared logarithmic error, R-squared, mean absolute error, mean squared error, and root mean squared error metrics. The results are shown in this work. As far as we are aware, the outcomes of tenfold cross-validation and a GridSearchCV are superior to the state-of-the-art studies.

Conclusion

The significance of efficient scoring in risk mitigation for critical systems has grown in tandem with the proliferation of software vulnerabilities. This research examines additional variables that could potentially enhance results, notwithstanding the predominant use of natural language processing (NLP) in current methodologies for vulnerability descriptions. Various standard metrics were utilized to assess the regression outcomes, including Explained Variance, Mean Squared Log Error, R^2, Mean Absolute Error, Mean Squared Error, and Root Mean Squared Error with an R2 value of 0.9459, explained variance of 0.9459, mean squared log error of 0.001, mean absolute error of 0.0076, mean squared error of 0.0055, and root mean squared error of 0.0744. RF regressor stood out among the six models evaluated. It is worth mentioning that the decision tree, random forest, and K-nearest neighbor regressors exhibited impartiality, as supported by comparable R^2 values. On the other hand, R^2 values of the linear, AdaBoost, and SVM models (0.7538, 0.7537), (0.7092, 0.7083), and (0.8298, 0.8297), respectively, demonstrated that these models were biased. Throughout cross-validation, the decision tree outperformed alternative regressors in terms of MAE& NMSE.

RF demonstrated exceptional performance across all six metrics, as confirmed by GridSearchCV and tenfold cross-validation, yielding a final score of 0.9486. The findings of this research offer significant contributions to the knowledge base of security professionals and researchers, facilitating the detection of critical software flaws and vulnerabilities.

Data Availability The data that support the findings of this research are openly available in National Vulnerability Database [22].

Declarations

Conflict of interest There is no conflict of interest among biographers.

Informed Consent Not Applicable.

Research Involving Human Participants and/or Animals Not Applicable.

References

1. Skybox Security (2022); 2022. https://www.vulnerabilitycenter.com/#home.
2. Holm H, Afridi KK. An expert-based investigation of the common vulnerability scoring system. Comput Secur. 2015;53:18–30. https://doi.org/10.1016/j.cose.2015.04.012.
3. Khazaei A, Ghasemzadeh M, Derhami V. An automatic method for CVSS score prediction using vulnerabilities description. J Intell Fuzzy Syst. 2016;30(1):89–96. https://doi.org/10.3233/IFS-151733.
4. Spanos G, Angelis L. A multi-target approach to estimate software vulnerability characteristics and severity scores. J Syst Softw. 2018;146:152–66. https://doi.org/10.1016/j.jss.2018.09.039.
5. Ruohonen J. A look at the time delays in CVSS vulnerability scoring. Appl Comput Inform. 2019;15(2):129–35. https://doi.org/10.1016/j.aci.2017.12.002.
6. Chatterjee S, Thekdi S. An iterative learning and inference approach to managing dynamic cyber vulnerabilities of complex systems. Reliab Eng Syst Saf. 2020. https://doi.org/10.1016/j.ress.2019.106664.
7. Chen H, Liu J, Liu R, Park N, Subrahmanian VS. VEST: a system for vulnerability exploit scoring and timing. IJCAI Int Jt Conf Artif Intell. 2019;2019:6503–5. https://doi.org/10.24963/ijcai.2019/937.

8. Syed R. Cybersecurity vulnerability management: a conceptual ontology and cyber intelligence alert system. Inf Manag. 2020. https://doi.org/10.1016/j.im.2020.103334.
9. Mell P, Scarfone K. Improving the common vulnerability scoring system. IET Inf Secur. 2007;1(3):119–27. https://doi.org/10.1049/iet-ifs:20060055.
10. Wang JA, Guo MM, Camargo J. An ontological approach to computer system security. Inf Secur J. 2010;19(2):61–73. https://doi.org/10.1080/19393550903404902.
11. Zhang S, Ou X, Caragea D. Predicting cyber risks through national vulnerability database. Inf Secur J. 2015;24(4–6):194–206. https://doi.org/10.1080/19393555.2015.1111961.
12. Xiao Q, Li K, Zhang D, Xu W. Security risks in deep learning implementations. In: Proc. 2018 IEEE Symp. Secur. Priv. Work SPW 2018;2018. p. 123–8. https://doi.org/10.1109/SPW.2018.00027.
13. Zheng W, et al. The impact factors on the performance of machine learning-based vulnerability detection: a comparative study. J Syst Softw. 2020. https://doi.org/10.1016/j.jss.2020.110659.
14. Spanos G, Angelis L. Impact metrics of security vulnerabilities: analysis and weighing. Inf Secur J. 2015;24(1–3):57–71. https://doi.org/10.1080/19393555.2015.1051675.
15. HuangS, Tang H, Zhang M, Tian J. Text clustering on national vulnerability database. In: 2010 2nd Int. Conf. Comput. Eng. Appl. ICCEA 2010, vol. 2; 2010. p. 295–9. https://doi.org/10.1109/ICCEA.2010.209.
16. WijayasekaraD, Manic M, McQueen M. Vulnerability identification and classification via text mining bug databases. In: IECON Proc. (Industrial Electron Conf.) 2014. p. 3612–8. https://doi.org/10.1109/IECON.2014.7049035.
17. Chen J, Kudjo PK, Mensah S, Brown SA, Akorfu G. An automatic software vulnerability classification framework using term frequency-inverse gravity moment and feature selection. J Syst Softw. 2020;167:110616. https://doi.org/10.1016/j.jss.2020.110616.
18. AnjumM, Kapur PK, Agarwal V, Khatri SK. A framework for prioritizing software vulnerabilities using fuzzy best-worst method. In: ICRITO 2020—IEEE 8th Int. Conf. Reliab. Infocom. Technol. Optim. (Trends Futur Dir); 2020. p. 311–16. https://doi.org/10.1109/ICRITO48877.2020.9197854.
19. Vishnu PR, Vinod P, Yerima SY. A deep learning approach for classifying vulnerability descriptions using self attention based neural network. J Netw Syst Manag. 2022;30(1):1–27. https://doi.org/10.1007/s10922-021-09624-6.
20. Allodi L, Massacci F. Comparing vulnerability severity and exploits using case-control studies. ACM Trans Inf Syst Secur. 2014. https://doi.org/10.1145/2630069.
21. Johnson P, Lagerstrom R, Ekstedt M, Franke U. Can the common vulnerability scoring system be trusted? A bayesian analysis. IEEE Trans Dependable Secur Comput. 2018;15(6):1002–15. https://doi.org/10.1109/TDSC.2016.2644614.
22. NVD2021; 2021. https://nvd.nist.gov/vuln/data-feeds.
23. Kohavi R, John GH. Automatic parameter selection by minimizing estimated error. Mach Learn Proc. 1995;1995:304–12. https://doi.org/10.1016/b978-1-55860-377-6.50045-1.
24. Lessmann S, Baesens B, Mues C, Pietsch S. Benchmarking classification models for software defect prediction: a proposed framework and novel findings. IEEE Trans Softw Eng. 2008;34(4):485–96. https://doi.org/10.1109/TSE.2008.35.
25. Gupta KK, Kalita K, Ghadai RK, Ramachandran M, Gao XZ. Machine learning-based predictive modelling of biodiesel production—a comparative perspective. Energies. 2021. https://doi.org/10.3390/en14041122.
26. Shanmugasundar G, Vanitha M, Čep R, Kumar V, Kalita K, Ramachandran M. A comparative study of linear, random forest and adaboost regressions for modeling non-traditional machining. Processes. 2021. https://doi.org/10.3390/pr9112015.
27. Hyndman RJ, Koehler AB. Another look at measures of forecast accuracy. Int J Forecast. 2006;22(4):679–88. https://doi.org/10.1016/j.ijforecast.2006.03.001.
28. Jollife IT, Cadima J. Principal component analysis: a review and recent developments. Philos Trans R Soc A Math Phys Eng Sci. 2016. https://doi.org/10.1098/rsta.2015.0202.

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

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.