Journal of Ambient Intelligence and Humanized Computing (2024) 15:2641–2651
https://doi.org/10.1007/s12652-024-04778-y

ORIGINAL RESEARCH

Software security with natural language processing and vulnerability scoring using machine learning approach

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

Received: 14 October 2023 / Accepted: 21 February 2024 / Published online: 3 April 2024

Abstract

As software gets more complicated, diverse, and crucial to people's daily lives, exploitable software vulnerabilities constitute a major security risk to the computer system. These vulnerabilities allow unauthorized access, which can cause losses in banking, energy, the military, healthcare, and other key infrastructure systems. Most vulnerability scoring methods employ Natural Language Processing to generate models from descriptions. These models ignore Impact scores, Exploitability scores, Attack Complexity and other statistical features when scoring vulnerabilities. A feature vector for machine learning models is created from a description, impact score, exploitability score, attack complexity score, etc. We score vulnerabilities more precisely than we categorize them. The Decision Tree Regressor, Random Forest Regressor, AdaBoost Regressor, K-nearest Neighbors Regressor, and Support Vector Regressor have been evaluated using the metrics explained variance, r-squared, mean absolute error, mean squared error, and root mean squared error. The tenfold cross-validation method verifies regressor test results. The research uses 193,463 Common Vulnerabilities and Exposures from the National Vulnerability Database. The Random Forest regressor performed well on four of the five criteria, and the tenfold cross-validation test performed even better (0.9968 vs. 0.9958).

Keywords Common vulnerability and exposures · Software vulnerability scoring · National vulnerability database · Natural language processing

1 Introduction

These days, software is an integral component of our lives and has many different functions. Although the world's brightest minds work on software development, even they aren't immune to making errors that could lead to security vulnerabilities. These vulnerabilities are weaknesses in the software development process that can be exploited by attackers to gain system access. As a result, software vulnerabilities serve as a potential point of access for cybercriminals to gain control of a machine. Despite greater vulnerability research and awareness, a growing trend toward the identification and reporting of new vulnerabilities has been observed. Figure 1 shows that the overall number of detected vulnerabilities increased from 18,325 in 2020 to 20,171 in 2021, and then to 25,227 in 2022. Newly found vulnerabilities increased from 14,714 in 2017 to 16,557 in 2018 and 17,344 in 2019. When a new cyber vulnerability is discovered, it is assigned a Common Vulnerabilities and Exposures (CVE) number by a vulnerability number assigning authority like Cisco.

Figure 2 shows how the US National Institute of Standards and Technology (NIST) looks into the vulnerability and gives it a CVSS severity score based on eight common features of vulnerabilities (Shahid and Debar 2021). The severity of software vulnerabilities has been assessed by a variety of organizations, each of which uses its methodology. Over the course of the last few decades, numerous nonprofitable organizations and businesses that sell system security-related goods and services have developed and put into operation a variety of software vulnerability scoring and reporting systems. They are limited in scope, and incompatible.

The proposed methodology uses an NLP-based approach to score the severity of software vulnerabilities. With our natural language processing (NLP)-based approach, we pre-processed

✉ Ajay Kumar Yadav
ajay.iitdhn@gmail.com

Birendra Kumar Verma
birendraverma@jssaten.ac.in

[1] Banasthali Vidyapith, Jaipur, Rajasthan, India

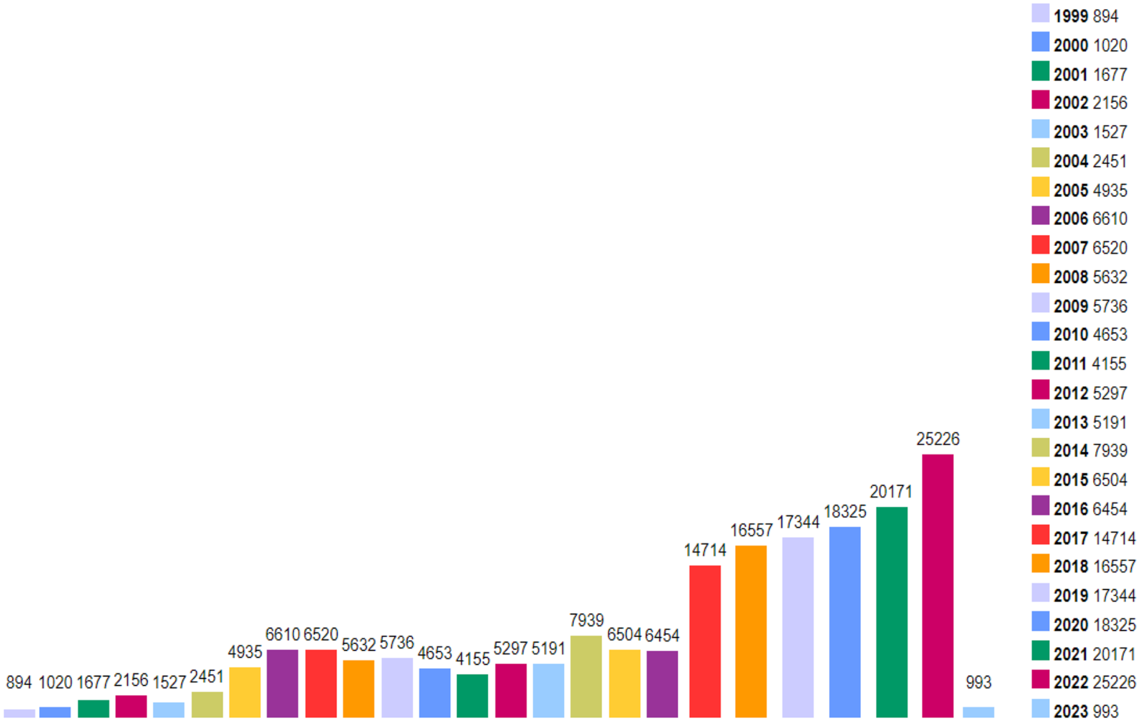
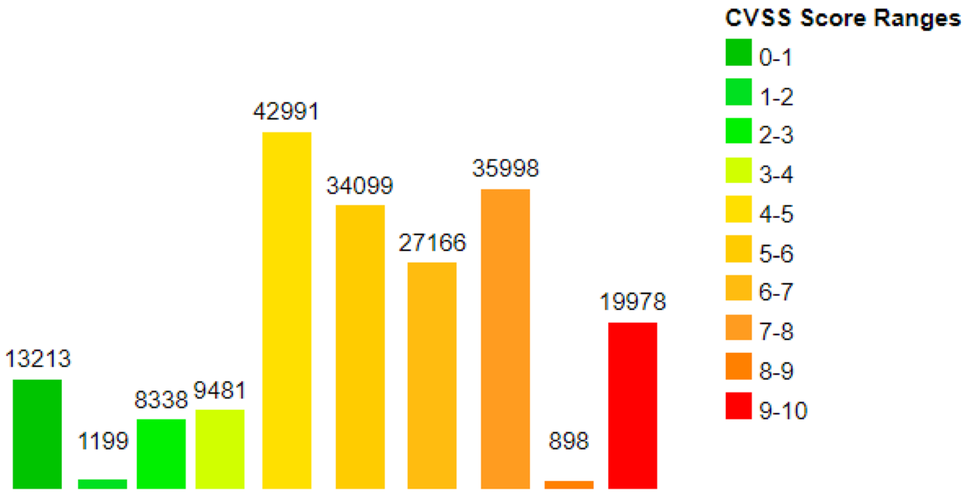


Fig. 1 Vulnerabilities by year

Fig. 2 Vulnerability distribution by CVSS scores



the textual data in the description attribute that characterizes the specific vulnerability. All relevant pre-processing processes, such as removing numerals and special characters, were performed prior to tokenizing the description for use in the model's construction.

2 Literature review

In this section, we discussed the efforts of various researchers to quantify and categorize software vulnerabilities.

Very few researchers have focused on regression, which provides more precise value than classification, despite the fact that classification is the more common method used in research. Classification and regression were both considered in the review process.

Abedin et al. (2006) presented a formula for estimating the metric that considers a variety of considerations, including the system's vulnerability, traffic dynamics, as well as the history of infrastructure vulnerability and network visibility. They also used a mining algorithm, so that they could be able to obtain the vulnerability background of the service history on the system using NIST's National Vulnerability Database, NVD. Frühwirth and Männistö (2009) presented a tool that is particularly useful to users who do not have the ability to collect vast quantities of empirical data because it enables them to simulate change potential using only publicly accessible data from the NVD. They tested the approach on a set of 720 vulnerability announcements from the NVD and discovered that providing context information improved vulnerability prioritization and selection significantly. The authors also did manual classification and data collection from NVD over a fixed period of time. Huang et al. in (2010) proposed vulnerability classification based on text clustering on NVD to address the issue of classification vulnerability bunch in vulnerability. The authors used overlap index of cluster for the evaluation of Batch Sum, Bisection means, and Simple K Mean. Out of 40,000 vulnerabilities Authors used only 45 main clusters by the reference of DDI (Descriptor Dominance Index). Zhang et al. in (2015) applied a Data Mining approach to NVD data to predict the time to the next vulnerabilities for a given app. They also used a machine learning algorithm for the prediction purpose. The authors propose possible explanations for why the NVD data did not provide an accurate prediction model for TTN vulnerability with our current approach, as well as alternate uses for the NVD data. Spanos and Angelis (2018) proposed an improved manual procedure for the characteristic assignment of vulnerabilities. The authors also applied text analysis and multi-target classification techniques (label power set transformation algorithms). The authors applied Boosting, Decision Tree, and Random Forest algorithms for vulnerability classification. Ruohonen (2019) explained the time lag between the publication of CVE in NVD and CVSS information attached with published CVEs. They used regression analysis on 80,000 stored vulnerabilities and found that time delays do not affect the content of CVSS. Anjum et al. in (2020) built a tool for calculating the relative attack susceptibility of critical network security resources. They prioritized the bugs discovered in the program using a hybrid Fuzzy Best Worst Method (FBWM). According to the authors, SQL Injection, Gain of Privileges, and Information Gain (IG) were listed as the most serious vulnerabilities that needed to be addressed as soon as possible. Chen et al. in (2020) proposed a classification model based on frequency inverse gravity moment (TF-IGM). They also applied a machine learning algorithm on a vulnerable application having 27,248 vulnerabilities. Authors showed by experiment that in the classification process feature selection varies according to the database selected and improves the efficiency. Shuang et al. in (2020) Presented convolution—a deconvolution word embedding multi-prototype model that is capable of fusion. They also applied CDWE to NLP tasks, translation, and classification and depicted the efficiency. Wijayasekara in (2023) presented a model that is capable of identifying vulnerabilities in programs using text information for Hidden Impact Bugs (HIBs) from publicly available datasets. The authors obtained syntactical information by applying a mining algorithm from publicly available bug reports and also generated vectors for classification purposes. Vishnu et al. (2022) came up with an idea for a mechanism to categorize vulnerabilities by employing the technique of deep learning. A self-attention deep neural network (SA-DNN) model and a text mining technique are used in the proposed approach to categorize vulnerabilities based on how they are written about. A SA-DNN-based system for classifying vulnerabilities is put through its paces by being tested with 134,091 vulnerability reports taken from the CVE information website. The results of the experiments demonstrated that the suggested method is efficient 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.

2.1 Dataset information

The data utilized in this research was sourced from the publicly accessible National Vulnerability Database, widely regarded as the most extensive and comprehensive repository of vulnerabilities. The NVD has been compiling vulnerability records since 2002, and it now contains more than 1,93,463 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 pertaining to 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. Regarding 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. 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 to 10. The Base Score is consistent across time and between different user contexts. Primarily there are 1,93,463 rows and 32 columns as shown in Table 1. Each row contains information on a different common vulnerability exposure, such as CVE-2023-32501, as well as different categories of information about that vulnerability. (Exploitability Score.1, Impact Score, Severity, etc.). The Description column in Table 1 provides descriptive information in the form of text for each of the common vulnerability exposures. The phrase “cross-site scripting x vulnerability easy upload An example of description data is “php city post simple php upload allows remote attackers to inject arbitrary web script HTML via message argument”. Additionally, the published date and the last modified date for each CVE are formatted as follows: 2021-01-15T18:15Z and 2021-01-21T19:50Z, respectively. One can use this information for vulnerability scoring. We relied a lot on the data in the description column and other statistical data for our study. We used natural language processing to get the data ready.

3 Methodology

In Fig. 3, we provide an overview of the text-mining-based software vulnerability scoring method we created. The proposed framework’s overall structure is shown in Fig. 3. The architecture comprises of multiple levels, the first of which is the pre-processing of vulnerability description data, followed by indexing, embedding, vectorization, and the construction of a regression model. Tokenizing, upper/lower

Table 1 Dataset description

Columns/rows	CVE	Vector string	Exploitability Score	Impact Score	Base Score	Exploitability Score.1	Impact Score	……	Description
Row 0	CVE-2021-0109	CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H	1.8	5.9	7.8	3.9	6.4	……	Insecure inherited permissions for the Intel(R…
Row 1	CVE-2021-0202	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H	3.9	3.6	7.5	10	2.9	……	On Juniper Networks MX Series and EX9200 Serie…
Row 2	CVE-2021-0203	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:N/I:N/A:H	3.9	4	8.6	8.6	2.9	……	On Juniper Networks EX and QFX5K Series platfo…
Row 3	CVE-2021-0204	CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H	1.8	5.9	7.8	3.9	10	……	A sensitive information disclosure vulnerabili…
Row 4	CVE-2021-0205	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:N/I:N/A:L	3.9	1.4	5.8	8.6	2.9	……	When the "Intrusion Detection Service" (IDS) f…
……	………	………	…	…	…	…	……	……	…………………
Row 193,463	CVE-2002-2446	NaN	NaN	NaN	NaN	10.0	10.0		GE Healthcare Millennium MG, NC, and MyoSIGHT …

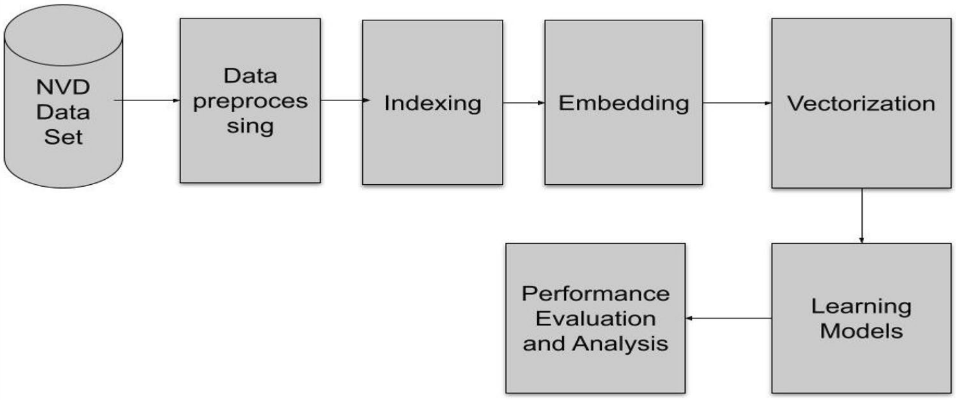


Fig. 3 Model architecture

case switching, stop word filtering, deleting punctuation, digits, special characters, and word stemming are all part of the pre-processing stage for vulnerability descriptions. Here's an example of a vulnerability Description: 'Multiple PHP remote File Inclusion Vulnerability'.

3.1 Data pre-processing

Vulnerability descriptions, which provide textual information regarding vulnerabilities, are processed in the data preprocessing module. In the following phase of pre-processing called tokenization, we separate the statement into its parts, such as ['Multiple', 'Php', 'remote', 'File,' 'Inclusion', and 'Vulnerability']. After employing a case-changing technique that allowed us to alter the text's upper- and lowercase forms, we settled on ['multiple,' 'php', 'remote,' 'file,' 'inclusion,' 'vulnerability']. Due to the increased focus on semantics, terms with equivalent meanings must be presented in the same case. We filter out unnecessary elements from the description data, such as URLs, numbers, punctuation, hyperlinks, special characters, and stop words, to obtain refined information. Stemming, the elimination of a word's affixes to reveal its original form, was the final step. In this context, 'inclusion' stands in for 'inclusions, 'including, and other words of a similar meaning. Improves the regressor's ability to represent words.

3.2 Indexing

We applied one hot encoding to the entire corpus after obtaining the pre-processed data, yielding a complete word index. As an illustration, the format might be [4276, 4441, 2336, 1870, 3123, 3069, 2679, etc.].

3.3 Embedding representation

Since the length of the sentences in my corpus varies, we used a pre-paid sequence in our embedding module to provide a constant output. The output of embedding representation looks like an array ([[0,0, 0, ….,2068,4743,4581], [0,0, 0, ….,1800,2901,2499]]).

3.4 Vectorization

To make use of the index contained in the embedding representation, we must first convert it to a feature vector. After generating a corpus-wide feature vector, we fed it to a variety of regressors.

3.5 Proposed regression models

3.5.1 Random forest regression

The predictions made by Random Forest are identical to those made by the average of all algorithmic regression trees. The algorithm's name alludes to the fact that it is random. Like bagging, random selection is used to select the sample data for each iteration. A method based on random subspace for regression variables of interest According to the findings of (Khoshgoftaar et al. 2007) investigation, just two of the parameters need to be set and tuned.

3.5.2 Decision tree regression

The decision tree regressor is a regressor that is in the shape of a tree (Kohavi and John 1995). It is the building block upon which Random Forest and boosting are constructed. The decision tree does a good job of processing categorical data and is also good at handling outliers and gaps**.**

3.5.3 K- nearest neighbors

It is a method of learning that does not rely on parameters and can be used for both classification and regression (Lessmann et al. 2008). The dataset is stored by lazy learner algorithms, and they use it to train regressors. K-Nearest Neighbors determines, based on a set of K training examples, which main class is most likely to apply to test cases.

3.5.4 AdaBoost regression

Random training subsets are used in AdaBoost, which results in learners with less skill (Shanmugasundar et al. 2021). Weights are used in the learning process for each hypothesis. When determining how incorrect the hypothesis is across the entire dataset, weights indicate how much emphasis should be placed on each case.

3.5.5 Support vector regression

It is a highly effective method of machine learning that can be applied in a variety of contexts. It can also be used for regression and classification. According to the findings of Lessmann et al. (2008), SVM performed better than Naive Bayes.

3.6 Metrics to measure performance

The following measures were used to evaluate the efficacy of the above regressors (Hyndman and Koehler 2006).

3.6.1 Explained variance

The amount of variation in a model's predictions that can be explained by its explained variance is an important metric. To put it another way, it is the difference between the expected value and the predicted value. When we reconcile the dataset, we risk losing a lot of information, so we must understand this idea well.

3.6.2 Negative mean squared error (NMSE)

If score = "negative mean squared error" is used in the validation function, it will give negative results. If an MSE is 3, assume the answer is − 3. The answer is − 6 if an MSE is 6. This is because the cross-validation function is based on maximization. Scorer objects are usually made so that they always return a number greater than zero.

3.6.3 r-Squared (r2)

The r2 score is a key part of figuring out how well a regression-based machine-learning model works (Gupta et al. 2021). The usefulness of the dataset is shown by how much of the difference between the predictions can be linked to it. It is the difference between the samples in the dataset and what the model says will happen. Values of 0 and 1 for r2 show that the model doesn't do a good job of predicting future data, while values of 1 show that the model will work well with a new set of data.

3.6.4 Mean absolute error (MAE)

The absolute error is the difference between the value that was predicted and the value that occurred, and it is stated in absolute terms. The mean absolute error, which is also called the MAE, gives an idea of how often a forecast is wrong.

3.6.5 Mean squared error (MSE)

A statistic called the mean squared error (MSE) is used to measure the error in statistical models. It figures out the mean squared difference between what happened and what was expected. When the model is perfect, the MSE is equal to zero. Its value goes up directly to how wrong the model it is based on. Mean squared error and mean squared deviation are both names for the same statistic (MSD).

3.6.6 Root mean squared error (RMSE)

Root mean squared error (RMSE) is the most common way to measure how well regression models work. The goal is to figure out how far off the predictions of the model are from the truth when compared to other data. So, it's best to have a small RMSE and avoid having a big one. The r^2, an MAE, an MSE, and an RMSE are all calculated by Eqs. (1–4).

R-squared (r^2)

$$1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y}_i)^2} \tag{1}$$

Mean absolute error (MAE)

$$\frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{2}$$

where y_i is the actual value and $\hat{y}_i$ is predicted?

Mean squared error (MSE)

$$\frac{\sum_{\boldsymbol{i}=1}^{\boldsymbol{n}} \left(\boldsymbol{y_i} - \boldsymbol{\hat{y}_i}\right)^2}{\boldsymbol{n}} \tag{3}$$

Root mean squared error (RMSE)

$$\sqrt{\frac{\sum_{\boldsymbol{i}=1}^{\boldsymbol{n}} \left(\boldsymbol{y} - \boldsymbol{\hat{y}_i}\right)^2}{\boldsymbol{n}}} \tag{4}$$

4 Results and analysis

Figure 8 compares the train, test, and tenfold r^2 values for DT, RF, KNN, AdaBoost, and SVR. Figure 7 shows how DT, RF, KNN, AdaBoost, and SVRs did in the tenfold cross-validation based on an NMSE. Figure 6 shows how well the DT, RF, KNN, AdaBoost, and SVR did in a tenfold cross-validation based on the r^2 value. Figures 4 and 5 show how the DT, RF, KNN, AdaBoost, and SVR models did when they were trained and tested using explained variance, r^2, MAE, an MSE, and RMSE as metrics. In Table 2, we compare the DT, RF, KNN, Ada Boost, and SVR models using explained variance, r^2, MAE, MSE, and RMSE. The set for testing is 20%, and the set for training is 80%. The RMSE, the MAE, and the MSE, along with the explained variance, are used to look at the results of each regression model. Here, we talk about the new findings and give our thoughts on them.

On four out of the five test set measures, the random forest regressor did a good job. This model had an explained variance of 0.9958, an r2 of 0.9958, an MAE of 0.0076, an MSE of 0.0055, and an RMSE of 0.0744. On five of the five measures, AdaBoost did badly, making it the worst regressor. This model has an explained variance of 0.9217, an r^2 of 0.9140, an MAE of 0.2175, an MSE of 0.1138, and an RMSE of 0.3373. Support vector regressors perform average. The model's statistics are: 0.9898 explained

Fig. 4 Training results of different regressors based on metrics: explained variance, r^2, MAE, MSE, and RMSE

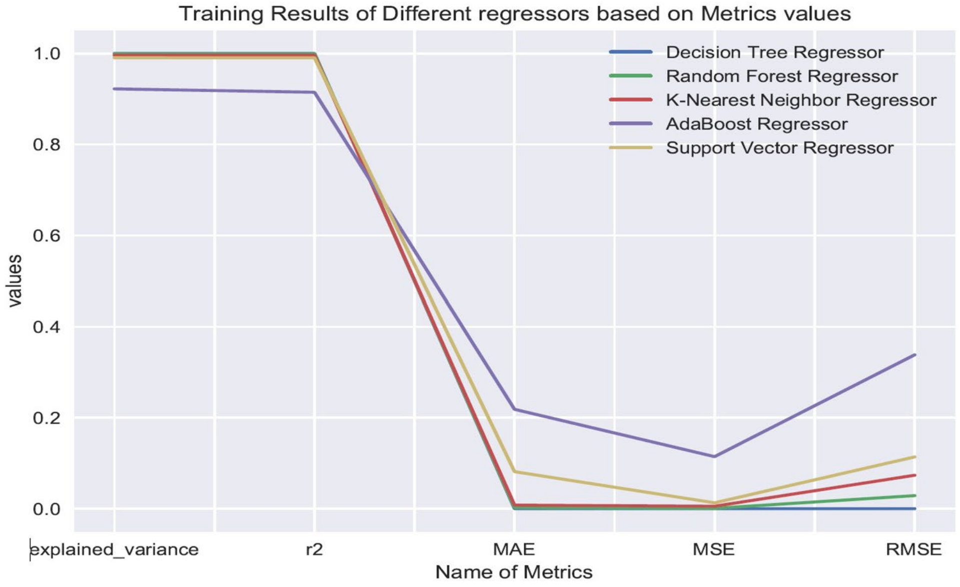


Fig. 5 Testing results of different regressors based on metrics: explained variance, r^2, MAE, MSE, and RMSE

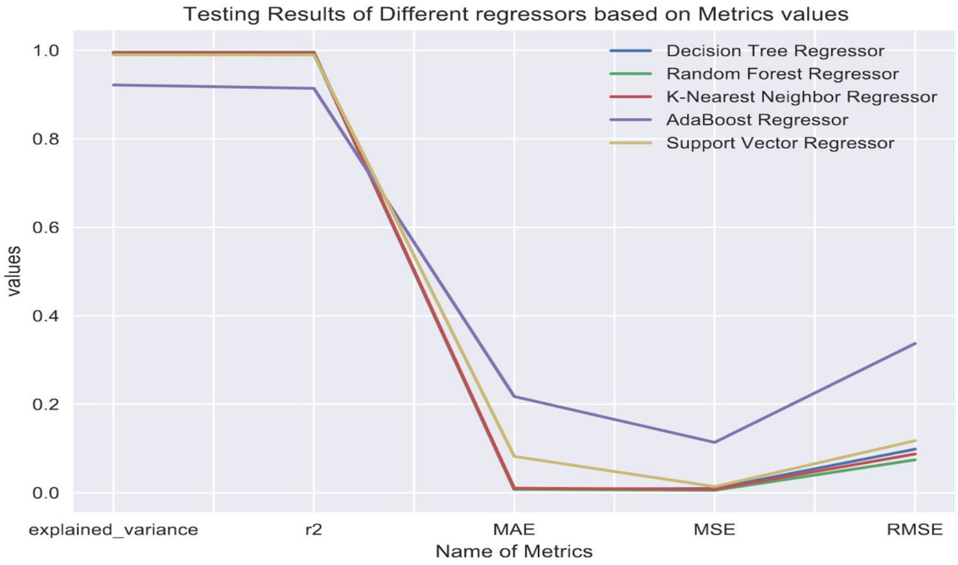


Table 2 Based on 20% of testing and 80% of training, these are the results

	Training results of different regressors					Testing results of different regressors				
Evaluation metrics	Decision tree regression	Random forest regression	K-Nearest neighbor regression	AdaBoost regression	Support vector regression	Decision tree regression	Random forest regression	K-nearest neighbor regression	AdaBoost regression	Support vector regression
Explained variance	**1.0000**	0.9994	0.9960	0.9221	0.9905	0.9927	**0.9958**	0.9942	0.9217	0.9898
r2	**1.0000**	0.9994	0.9960	0.9147	0.9904	0.9927	**0.9958**	0.9942	0.9140	0.9896
MAE	**0.0000**	0.0022	0.0079	0.2184	0.0814	**0.0074**	0.0076	0.0103	0.2175	0.0821
MSE	**0.0000**	0.0008	0.0054	0.1144	0.0129	0.0097	**0.0055**	0.0077	0.1138	0.0138
RMSE	**0.0000**	0.0287	0.0735	0.3382	0.1138	0.0985	**0.0744**	0.0876	0.3373	0.1176

variance, 0.9896 R-squared value, 0.0821 mean absolute error, 0.0138 MSE, and 0.1176 RMSE.

The explained variance and r^2 for the DT, RF, KNN, Ada Boost, and SVR models are 0.9927, 0.9958, 0.9942, 0.9217, and 0.9898, respectively, as shown in Table 2 and Fig. 5. Experiments demonstrate that DT, RF, and KNN regressors are unbiased because their scores for explained variance and r^2 are similar. The AdaBoost and SVR models are biased because the explained variance and r^2 scores are different.

The r^2 values for DT, RF, KNN, AdaBoost, and SVR after 10 rounds of cross-validation are displayed in Fig. 6. In tenfold cross-validation, each of these regressors performs admirably, but the Random Forest regressor consistently outperforms the rest. The results of validating the DT, RF, KNN, Ada boost, and SVR regressors are 0.9948, 0.9968, 0.9959, and 0.9894. A negative mean squared error measure was also utilized. As can be seen in Fig. 7, we applied this metric for scoring in a tenfold cross-validation setting with DT, RF, KNN, Ada boost, and SVR regressors.

Figure 8 shows that Random Forest does better than DT, KNN, AdaBoost, and SVR in terms of r^2 values for training, testing, and tenfold results. AdaBoost performed poorly across the board, with an explained variance of 0.9217, an r^2 of 0.9140, an MAE of 0.2175, an MSE of 0.1138, and an RMSE of 0.3373, making it the worst regressor. In addition, Fig. 8 shows that support vector regressors have an average level of performance.

4.1 Comparison to recent field developments

Many software vulnerability classification methods rely solely on technical parameters, while others rely solely on vulnerability descriptions utilizing NLP. The majority of the papers in the reviewed literature classified vulnerabilities solely based on vulnerability descriptions using NLP and conventional metrics like precision, recall, etc. There are two publications we found that use similar metrics to ours when assessing vulnerabilities and performance. Using data mining on NVD datasets, Zhang et al. (2015) looked into determining if and when software systems would become vulnerable once more. The performance of various ML methods was evaluated using the WEKA tool, with the following results: CC = 0.1974, RRSE = 694.7868, and CC = 0.1974. In their study of three different methods for scoring CVSS, Atefeh et al. (2016)found that the fuzzy system performed the best, with an accuracy of 88%.

We used to explain variance, r-squared, MAE, MSE, and RMSE to compare DT, RF, KNN, Ada Boost, and Support Vector Regressor. We showed ten-fold cross-validation results that, as far as we know, are better than the state of the art right now.

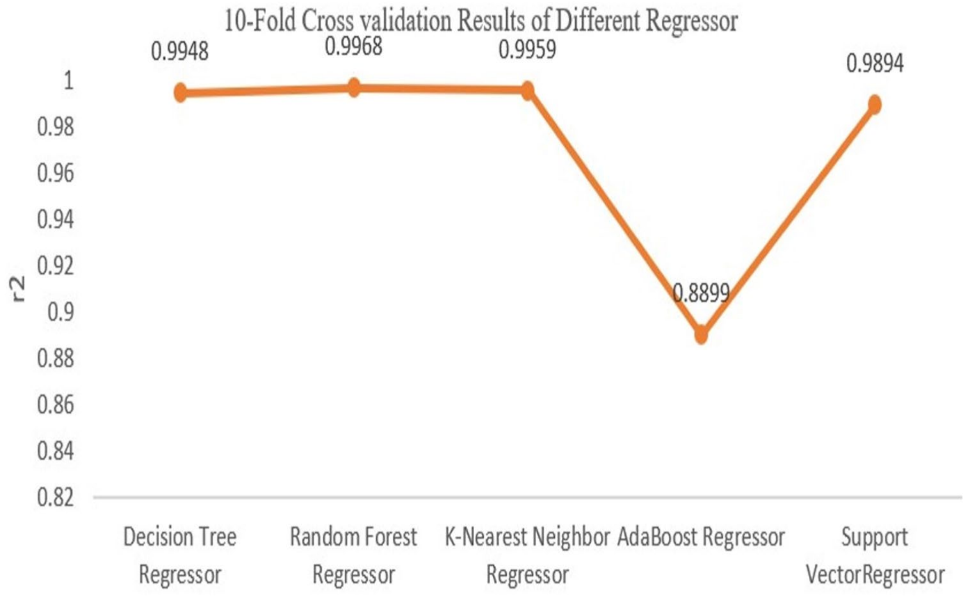


Fig. 6: 10-fold cross-validation results of different regressors based on r^2 value

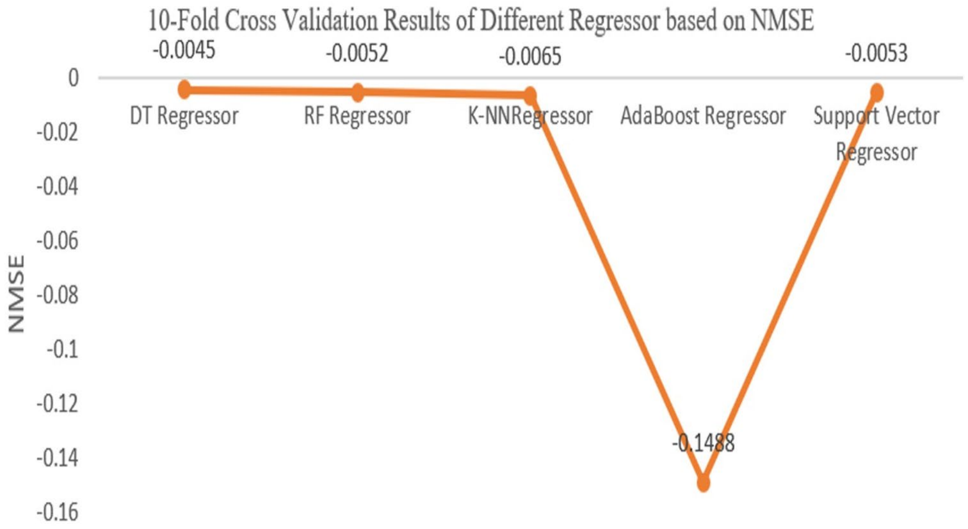


Fig. 7 10-fold cross-validation results of different regressors based on NMSE

5 Conclusion

As a result of this research, we proposed a methodology for scoring software vulnerabilities that is based on natural language processing. We pre-processed the textual data that was provided in the description category that characterized the vulnerability as part of our NLP-based strategy. This method was implemented so that we could better understand the issue. The scoring model was built with the help of a number of different machine-learning techniques, such as the Decision Tree Regressor, the Random Forest Regressor, the AdaBoost Regressor, the K-Nearest Neighbors Regressor, and the Support Vector Regressor. Additionally, the model makes use of 193,463 Common Vulnerabilities and Exposures. The Random Forest regressor did well on four out of the five criteria, and the tenfold cross-validation improved it to a score of 0.9968 from 0.9958.

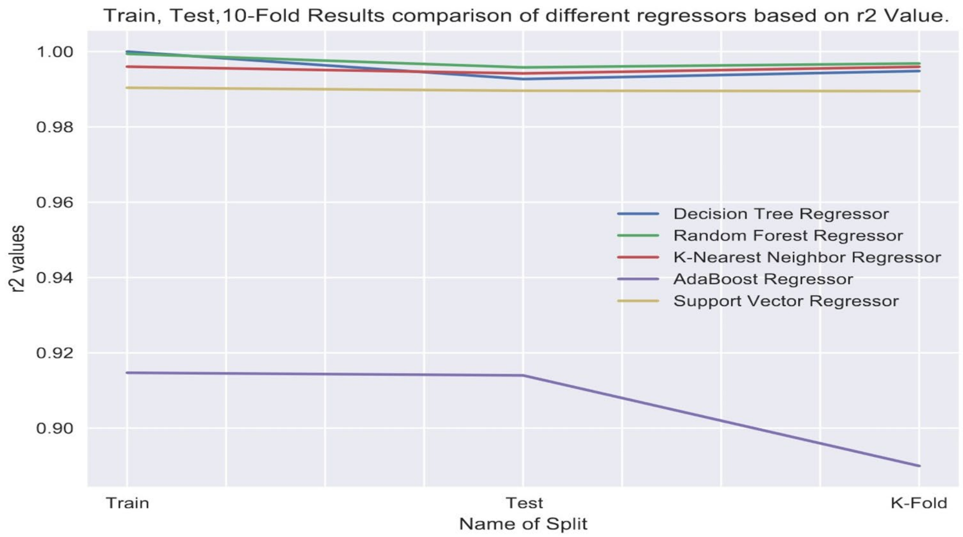


Fig. 8 Train, test, 10-fold results comparison of different regressors based on r^2 value

Funding Not applicable.

Data availability The datasets have been used from the "National Vulnerability Database" (NVD) https://nvd.nist.gov/vuln.

Declarations

Conflict of interest Authors declare that they have no conflict of interest.

Ethical approval This article does not consider humans and any kinds of animals.

Consent to publish The author is agreeing to submit this version of the paper for publication.

References

Abedin M, Nessa S, Al-Shaer E, Khan L (2006) Vulnerability analysis for evaluating quality of protection of security policies. In: Proc 2nd ACM Work Qual Prot QoP'06 Co-located with 13th ACM Conf Comput Commun Secur CCS'06, pp 49–52. https://doi.org/10.1145/1179494.1179505

Anjum M, Kapur PK, Agarwal V, Khatri SK (2020) 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), pp 311–316. https://doi.org/10.1109/ICRITO48877.2020.9197854

Chen J, Kudjo PK, Mensah S et al (2020) An automatic software vulnerability classification framework using term frequency-inverse gravity moment and feature selection. J Syst Softw 167:110616. https://doi.org/10.1016/j.jss.2020.110616

Frühwirth C, Männistö T (2009) Improving CVSS-based vulnerability prioritization and response with context information. In: 2009 3rd Int Symp Empir Softw Eng Meas ESEM 2009, pp 535–544. https://doi.org/10.1109/ESEM.2009.5314230

Gupta KK, Kalita K, Ghadai RK et al (2021) Machine learning-based predictive modelling of biodiesel production—a comparative perspective. Energies. https://doi.org/10.3390/en14041122

Huang S, Tang H, Zhang M, Tian J (2010) Text clustering on national vulnerability database. In: 2010 2nd Int Conf Comput Eng Appl ICCEA 2010, 2:295–299. https://doi.org/10.1109/ICCEA.2010.209

Hyndman RJ, Koehler AB (2006) Another look at measures of forecast accuracy. Int J Forecast 22:679–688. https://doi.org/10.1016/j.ijforecast.2006.03.001

Khazaei A, Ghasemzadeh M, Derhami V (2016) An automatic method for CVSS score prediction using vulnerabilities description. J Intell Fuzzy Syst 30:89–96. https://doi.org/10.3233/IFS-151733

Khoshgoftaar TM, Golawala M, Van Hulse J (2007) An empirical study of learning from imbalanced data using random forest. Proc Int Conf Tools Artif Intell ICTAI 2:310–317. https://doi.org/10.1109/ICTAI.2007.46

Kohavi R, John GH (1995) Automatic parameter selection by minimizing estimated error. Mach Learn Proc 1995:304–312. https://doi.org/10.1016/b978-1-55860-377-6.50045-1

Lessmann S, Baesens B, Mues C, Pietsch S (2008) Benchmarking classification models for software defect prediction: a proposed framework and novel findings. IEEE Trans Softw Eng 34:485–496. https://doi.org/10.1109/TSE.2008.35

Ruohonen J (2019) A look at the time delays in CVSS vulnerability scoring. Appl Comput Inform 15:129–135. https://doi.org/10.1016/j.aci.2017.12.002

Shahid MR, Debar H (2021) Cvss-bert: explainable natural language processing to determine the severity of a computer security vulnerability from its description. In: 2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA), pp 1600–1607

Shanmugasundar G, Vanitha M, Čep R et al (2021) A comparative study of linear, random forest and adaboost regressions for modeling non-traditional machining. Processes. https://doi.org/10.3390/pr9112015

Shuang K, Zhang Z, Loo J, Su S (2020) Convolution–deconvolution word embedding: an end-to-end multi-prototype fusion embedding method for natural language processing. Inf Fusion 53:112–122. https://doi.org/10.1016/j.inffus.2019.06.009

Spanos G, Angelis L (2018) A multi-target approach to estimate software vulnerability characteristics and severity scores. J Syst Softw 146:152–166. https://doi.org/10.1016/j.jss.2018.09.039

Vishnu PR, Vinod P, Yerima SY (2022) A deep learning approach for classifying vulnerability descriptions using self attention based neural network. J Netw Syst Manag 30:1–27. https://doi.org/10.1007/s10922-021-09624-6

Wijayasekara D, Manic M, Mcqueen M (2023) Vulnerability identification and classification via text mining bug databases

Zhang S, Ou X, Caragea D (2015) Predicting cyber risks through national vulnerability database. Inf Secur J 24:194–206. https://doi.org/10.1080/19393555.2015.1111961

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