CVSS Base Score Prediction Using an Optimized Machine Learning Scheme

Dinesh T. Vasireddy
Bentonville High School
dineshteja@gmail.com

Dakota S. Dale
University Of Arkansas
dsdale@uark.edu

Qinghua Li
University of Arkansas
qinghual@uark.edu

***Abstract*—The Common Vulnerability Scoring System (CVSS) is commonly used to measure the severity of software vulnerabilities. It consists of a CVSS Score Vector (i.e., a vector of metrics) and a CVSS Base Score calculated based on the vector. The Base Score is widely used by electric utilities to measure the risk levels of vulnerabilities and prioritize remediation actions. However, the process of determining the CVSS metric values is currently very time-consuming since it is manually done by human experts based on text descriptions of vulnerabilities, which increases the delays of remediating vulnerabilities and hence increases security risks at electric utilities. In this paper, we develop an efficient and effective solution to automatically predict the CVSS Base Score of vulnerabilities primarily based on their text descriptions, leveraging Natural Language Processing and machine learning techniques. Text descriptions for tens of thousands of vulnerabilities are comprehensively interpreted and vectorized using Doc2Vec, fed to a neural network with a condensed regression structure, which is then fine-tuned using Bayesian Optimization. By exploring and selecting the most efficient option at each stage of development, we create an optimized scheme that predicts CVSS Base Scores with very low error. Our work shows that it is possible to effectively predict CVSS Base Scores using simple but optimized neural networks. It makes crucial progress toward addressing the inefficiencies of the current CVSS severity assessment process through automation.**

***Index Terms*—Cybersecurity, Vulnerability, Machine Learning, Neural Networks, Automation**

I. Introduction

Public software vulnerability databases such as the National Vulnerability Database [1] and ExploitDB [2] are widely used by electric utilities and third-party services to discover the vulnerabilities in an organization's assets [3, 4, 5]. They have also become a major source of information for organizations to assess the risk of vulnerabilities. The vulnerability entries stored within these databases are called Common Vulnerabilities and Exposures (CVEs) [6]. The CVE system gives each of the vulnerabilities a unique and easily accessible identifier code, a CVE-ID, allowing for the CVEs to be much more organized. Each CVE has a text description. The initial documentation process of CVEs revealed that it was essential to develop a system to classify vulnerabilities based on their severity. This problem motivated the birth of the Common Vulnerability Scoring System (CVSS) [7], which usually utilizes a number of exploitability and impact metrics (see Table I) to calculate a base score valued between 0 and 10 that is representative of a vulnerability's severity.

TABLE I
CVSS Base Score Metrics

Metric	Possible Values
Attack Vector	*Network, Adjacent, Local, Physical*
Attack Complexity	*Low, High*
Privileges Required	*None, Low, High*
User Interaction	*None, Required*
Scope	*Unchanged, Changed*
Confidentiality Impact	*None, Low, High*
Integrity Impact	*None, Low, High*
Availability Impact	*None, Low, High*
Base Score	[0,10]

Nevertheless, the modern CVSS has been criticized by numerous data scientists for its more hard-arithmetic approach and the delays in manually providing severity assessments for vulnerabilities. A study [8] shows that, on average, it takes over 134 days after publication for a vulnerability to receive a full CVSS severity assessment. Additionally, it emphasizes that the CVSS' detailed metrics for severity assessments could be a factor for larger delays between vulnerability publication and severity assessment due to the heavily manual nature of CVSS classification.

With the advent of more advanced Natural Language Processing (NLP) models and neural network techniques, it is critical to explore their usability in predicting CVSS scores solely based on CVE descriptions as well as impact and exploitability criteria provided when a CVE is published to credible databases. In an effort to reduce the time delay between vulnerability submission and severity classification, in this paper, we develop a multi-faceted machine learning model for predicting CVSS base scores that combines NLP, neural networks, and Bayesian optimization. Our model primarily prioritizes efficiency and accuracy. Evaluations show that it outperforms two state-of-the-art solutions. Our solution does not aim to fully replace manual assessment of CVSS metrics, but can provide a quick, accurate estimate of CVSS base scores so that organizations can use it first before the manual assessment is available.

This paper is organized as follows. Section II reviews related work. Section III provides background to the techniques used. In Section IV, we break down the proposed model architecture and justify the development of this structure for our use case. Section V describes the parameter optimization and tuning methods. Section VI presents the evaluation results. Finally,

Section VII concludes the paper.

II. Related Work

Although much work has been done in vulnerability management [9, 10, 11, 12, 13, 14], there is relatively little research on the automated prediction of CVSS Scores.

Shahid and Debar [15] proposed a CVSS prediction approach based on BERT [16]. It focuses on predicting the value of individual metrics in the CVSS vector, and then calculating the base score. A similar approach was adopted by two other work [17, 18]. Different from them, our solution focuses on predicting the CVSS base score, but not the vector of metrics, considering that most electric utilities simply use the CVSS base score to assess risk levels and prioritize vulnerabilities. Two other solutions [19, 20] focused on predicting the categorical severity levels but our work predicts the numerical base score value.

Elbaz et al. [21] proposed to predict CVSS Base Scores by leveraging a Bag-of-Words model for text vectorization and a Linear Regression model for prediction. However, this approach has two limitations. First, text descriptions could easily be misinterpreted since a Bag-of-Words model only considers the number of occurrences of each word and completely disregards their order. Second, a linear relationship is assumed without basis between the vulnerability descriptions and the CVSS Score Vector. Our solution addresses these limitations by interpreting vulnerability descriptions as whole documents rather than sets of individual words with Doc2Vec and utilizing a neural network to capture the accurate relationship between descriptions and CVSS Base Scores.

III. Preliminaries

A. Background of Text Vectorization Tools

In order for a probabilistic model to interpret a CVE's text description, we need to find a reliable method to enumerate the description into computable values that the model can understand and train on.

1) Word2Vec: Developed and published in 2013 by Mikolov et. al., Word2Vec [22] has been the standard for interpreting text inputs in the machine learning world for many years. The Word2Vec model learns the context of a corpus of text by using a two-layer neural network structure that transforms individual words into numerical vectors representing their meaning. Word2Vec contains two main model architectures: *Continuous Bag-of-Words (CBOW)* and *Continuous skip-gram*. The Bag-of-Words Word2Vec approach predicts the middle word of a text input based on the surrounding words, disregarding the order of those words, and utilizes this process to interpret the entire input. The skip-gram Word2Vec approach, however, differs slightly by predicting words within a certain range around a target word and eventually learning specific representations of many words in a text input. Despite being considered reliable in past research applications, Word2Vec still lacks accuracy when attempting to understand the context of larger text inputs with multiple sentences because it treats the text as a collection of individual words rather than a singular entity.

2) Doc2Vec: Shortly after the publication of the Word2Vec algorithm, Le and Mikolov recognized that simply averaging Word2Vec vectors to vectorize and later interpret a multi-sentence or multi-paragraph text input was an ineffective and fairly inaccurate strategy. In order to address this issue, they developed the Doc2Vec Algorithm [23], which creates one single paragraph vector of each document rather than generating separate vectors for each individual word. The paragraph vectorization capabilities of Doc2Vec allow it to be more accurate and efficient when interpreting larger text inputs, like vulnerability descriptions. Similar to the Word2Vec Model, Doc2Vec also has two primary model types: *Distributed Bag-of-Words (DBOW)* and *Distributed Memory (DM)*. DBOW is similar to the previously mentioned skip-gram model from Word2Vec, except that DBOW uses the document ID as the input rather than the target word in order to predict randomly sampled words from the document. DM is similar to the Word2Vec Continuous Bag-of-Words model, but it uses a document ID when predicting the meaning of a target word based on its neighboring words.

B. Fast.ai Deep Learning Library

Our method is implemented based on the Fast.ai library. The library was created [24] to improve the efficiency of machine learning development by redesigning them to be rapidly productive while also being highly configurable. Fast.ai's focus on generating cheaper and more efficient model structures has allowed for more efficient model performance across many applications [25]. For example, using tools from the library, researchers were able to leverage a more efficient variant of an LSTM structure to develop an NLP model (MultiFit [26]) competitive with BERT, but with a cheaper pre-training approach and faster training times. These reasons are mainly why we selected Fast.ai for our implementation.

IV. Model Architecture and Components

This section describes our CVSS Base Score Prediction model that determines accurate Base Scores primarily using the human-generated text descriptions along with two simple binary identifiers (hasExploit, hasFix) that are all provided for each vulnerability upon entry to the National Vulnerability Database. For any vulnerability, a 1 for hasExploit signifies that the vulnerability has a known exploit while 0 signifies the opposite. Similarly, for hasFix, a 1 signifies that the vulnerability has a known solution or "fix" while 0 signifies the opposite.

A. Model for Text Vectorization

The complexity and specificity of CVE descriptions in vulnerability databases necessitate the use of a large-scale text vectorization tool that has the capacity to interpret text inputs as whole entities rather than sets of individual words. The Doc2Vec model meets these performance requirements by comprehensively analyzing text inputs as whole multi-sentence

or multi-paragraph inputs when attempting to interpret their meaning [27]. For the text vectorization process of the data preprocessing stage, we use the Doc2Vec implementation from Gensim Library [28], particularly the Distributed Memory approach within the Doc2Vec Algorithm, which allows for target word prediction based on surrounding context information.

B. Neural Network Model

The specific neural network model structure that we propose is as follows. In training, we have three LinBnDrop Blocks within a Sequential Container, each of which contains a BatchNorm, Linear, and Dropout Layer. Beginning with the Linear Layer, the block will input the vectorized description data and begin making connections between document vectors and representations within the realm of severity classification. The output of the Linear Layer is then passed through a Rectified Linear Activation Function (ReLU), which will activate the appropriate neurons in the neural network and directly output the previous layer's input if it is positive and zero if not. Following the ReLU function, the output is passed through a BatchNorm Layer, which normalizes the outputs from the previous layers before allowing them to continue through the remaining neurons and stabilizes the network as it runs. Finally, in order to prevent overfitting, the output is processed through a Dropout Layer, which randomly removes certain units of the input in order to simulate training on larger data sets and allow for the model to yield more accurate results. This process occurs in three different blocks throughout the model's training stage and prepares the model for the prediction stage.

V. Optimization and Hyperparameter Tuning Methodology

This section describes our methods to tune and optimize the model parameters for more effective training.

A. Data Preparation

1) Data Collection: In order to assemble a reliable set of CVEs, including text descriptions and calculated CVSS scores, the data was collected from two nationally recognized CVE databases: the National Vulnerability Database and ExploitDB. This data was collected and organized by VulnIQ [29], a vulnerability and intelligence solution dedicated to tracking and analyzing system vulnerabilities. We extracted this data from VulnIQ using web-scraping techniques to traverse and record VulnIQ's vulnerability records, which resulted in a dataset with CVSS data from 100,000 CVEs. The resulting dataset covers CVEs from June 2017 to July 2022. After merging CVE data from the previously mentioned data sources using CVE Identifiers, any CVE records with missing or invalid CVSS data were removed in a cleaning operation, resulting in a final dataset of 95,046 CVEs.

2) Doc2Vec Model Training and Text Vector Creation: In order to convert the text descriptions of the selected vulnerabilities into numerical data to enhance data processing speed and accuracy, we have selected the Doc2Vec technique to vectorize the text descriptions into a more understandable set of values representing the information in each description. Before training the Doc2Vec model on the available data to generate text vectors, we need to treat each individual text description as a separate "document" and then separate the text documents into arrays of individual words. Following the initial transformation of the text data, the corpus of text "documents" is then converted to a list and then analyzed to identify specific words/phrases that represent the negative or positive sentiment of the descriptions. After building a dictionary of the important vocabulary from the collected data, the Doc2Vec model is trained for 40 epochs using 70% of the converted corpus to become more efficient at understanding the meaning of vulnerability descriptions. The custom-trained Doc2Vec model is then used to generate a vector with 50 embeddings for every CVE's text description that most accurately explain their meaning (as exemplified in Table II).

TABLE II
CVE-2022- 20862 After Doc2Vec Description Vectorization

CVE_ID	Description	Generated Vector
CVE-2022-20862	A vulnerability in the web-based management interface of Cisco Unified Communications Manager (Unified CM) and Cisco Unified Communications Manager Session Management Edition (Unified CM SME)...	[0.34770614, 0.24144247, -0.3812419, -0.105862066, 0.27085847, -0.3862508, -0.19297537, -0.5055367, 1.3502474, -0.71859527, -0.13281806, 0.0069170436, -0.036240224, 0.82979566...

Then the vectors were attached to their respective CVE records in the existing data frame and then separated into 50 different columns, each with an embedding from the vector. Additionally, two binary identifiers (hasExploit, hasFix) were also One-Hot Encoded to use for CVSS Base Score Prediction and provide more support for each prediction rather than relying solely on the Doc2Vec generated vectors. The dataset with the vectorized descriptions was then split 70% and 30% for training and testing data, respectively, to be used by the neural network model.

B. Initial Optimization and Hyperparameter Identification

Though there are numerous parameters that can be manipulated to increase performance, we begin the optimization process by choosing six major hyperparameters that have the most impact on prediction error: Number of Epochs, Batch Size, Learning Rate (lr), Weight Decay (wd), Dropout Value (dp), Number of Layers Generated by LinBnDrop blocks. Additionally, depending on the number of layers generated, we will also focus on identifying the most beneficial layer sizes for each layer generated by each LinBnDrop block in the neural network when predicting CVSS Scores. The exact ranges of testable values for each hyperparameter in the optimization process are provided in Table III.

TABLE III
IDENTIFIED HYPERPARAMETERS WITH TESTABLE RANGES

Hyperparameter	Testable Range
Dropout Value	(0.01,0.50)
Weight Decay	$(4 \times 10^{-4}, 0.40)$
Learning Rate	$(1 \times 10^{-5}, 1 \times 10^{-1})$
Generated Layers	(1,3)
Layer 1 Size	(50,200)
Layer 2 Size	(100,1000)
Layer 3 Size	(200,2000)

TABLE IV
IDENTIFIED OPTIMAL HYPERPARAMETER COMBINATIONS

Hyperparameter	Comb 1	Comb. 2	Comb. 3	**Comb. 4**	Comb. 5
Dropout Value	0.415	0.238	0.416	**0.415**	0.179
Weight Decay	0.031	0.239	0.138	**0.283**	0.351
Learning Rate	0.098	0.093	0.060	**0.086**	0.069
Generated Layers (≈)	3	2	2	**2**	1
Layer 1 Size (≈)	89	85	139	**167**	110
Layer 2 Size (≈)	725	273	652	**746**	585
Layer 3 Size (≈)	1787	374	1	**2**	955

C. Justification for Bayesian Optimization

For the purpose of maximizing the model's performance, the Bayesian optimization algorithm is selected to identify the most lucrative set of values for the chosen hyperparameters. The Bayesian optimization algorithm, a probabilistic optimization technique based on the Bayes Theorem, generates a probabilistic model of the original function (CVSS score prediction) to add efficiency to the hyperparameter tuning process. Following the creation of this probabilistic surrogate model, the algorithm then directly searches the surrogate model to identify the optimal combinations for evaluation with the original prediction model.

$$P(B_j \mid A) = \frac{P(A \mid B_j)P(B_j)}{\Sigma_{i=0}^{n} P(A \mid B_i)P(B_i)} \tag{1}$$

As stated by the Bayes Theorem (Eq. 1), the conditional probability of an event can be calculated using the occurrence of a prior event by multiplying the probability of the first event by the probability of the second event given the first event. Essentially, it states that the probability of a hypothesis can be calculated based on prior probabilities of related events. In [30], Agnihotri and Batra state that, in the case of Bayesian Optimization, it allows the algorithm to find the most promising combination of hyperparameters at each stage of the process and then use it to determine the best combination to evaluate in the next stage until the most optimal combination is found. By using this optimization algorithm, we are able to accelerate the tuning process with intelligent optimization rather than tedious manual optimization.

D. Algorithm Execution

After defining the Bayesian optimization algorithm for our CVSS score prediction model, we execute the algorithm to optimize the chosen hyperparameters on the previously identified testable ranges in Table III, running for 40 epochs on each identified combination. We choose the Mean Absolute Error metric to evaluate the effectiveness of each combination of hyperparameter values due to its ability to accurately gauge the magnitude of error in the generated forecasts rather than be more inaccurately swayed by negative errors and outliers like Mean Absolute Percentage Error and occasionally Mean Squared Error.

The Bayesian optimization algorithm identifies the combinations with the minimum error values to find the optimal combination of hyperparameter values. The optimization algorithm is then repeated 4 more times to confirm the identified combinations as optimal for enhancing the model's performance.

Through multiple iterations of the Bayesian optimization algorithm, we found 5 primary combinations (shown in Table IV) that were consistently identified as most likely to decrease mean absolute error in the original model's CVSS score predictions. Upon analyzing the combinations, we found that the Dropout value (dp) and Weight Decay (wd) generally hovered around 0.33 and 0.21 respectively, but the layer sizes, Number of Generated Layers, and learning rate (lr) value showed more variation among the combinations, indicating that they are generally more combination specific.

E. Final Optimization Steps and Statistical Benefits

Based on the statistical results from testing the model's performance at each of the identified hyperparameter combinations (See Table IV), we conclude that combination 4 (Dropout Value: 0.415, Weight Decay: 0.283, Number of Generated Layers: 2, Layer 1 Size: 167, Layer 2 Size: 746, Layer 3 Size: Near 0) is the most optimal. At 40 and 50 epochs, combination 4 yielded the lowest Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) when compared to the results from the other combinations. Therefore, we focus solely on optimizing the results for combination 4 while manipulating the only remaining hyperparameters we have not optimized via Bayesian Optimization: Number of Epochs and Batch Size. We incrementally increase the number of epochs of the model with hyperparameter combination 4 to identify the number of epochs with the lowest values on the selected error metrics: MAE, MSE, RMSE, and MAPE (Mean Absolute Percentage Error).

For the sake of consistency and simplicity, we keep the Batch Size constant across all trials. In the initial testing of all the hyperparameter combinations, we use a batch size of 256 since a relatively smaller batch size is generally beneficial for a multi-faceted model in its initial stages of development. Therefore, we again set the batch size as 256 for the new trials in which we gradually increase the epochs of the model to minimize the error margin in the optimization stage prior to conducting a final evaluation of the model's usability.

Beginning by testing the model at 100 epochs, we then incrementally increase the training time by 50 epochs and test the model at each value, the results of which are evaluated using the same original error metrics of MAE, MSE, RMSE, and MAPE (Results in Table V). Upon analyzing the results

TABLE V
EPOCHS MANIPULATION TRIAL RESULTS WITH **HYP. COMB. 4** AT BATCH SIZE OF 256

Epochs	MAE	MSE	RMSE	MAPE	R^2
100	0.5685	0.7446	0.8629	0.1084	0.7624
150	0.5331	0.6846	0.8274	0.1054	0.7830
200	0.5181	0.6844	0.8273	0.1106	0.7857
250	0.5037	0.6563	0.8101	0.0982	0.7982
300	0.5007	0.6692	0.8180	0.1892	0.7966
350	0.4927	0.6606	0.8128	0.1246	0.8010
400	0.4906	0.6535	0.8084	0.1221	0.8042
450	**0.4748**	**0.6251**	**0.7906**	**0.1073**	**0.8144**

TABLE VI
BATCH SIZE MANIPULATION TRIAL RESULTS WITH **HYP. COMB. 4** W/ 450 EPOCHS

Batch Size	MAE	MSE	RMSE	MAPE	R^2
64	0.7743	1.1026	1.0501	0.1907	0.6078
128	0.6224	0.8286	0.9103	0.1413	0.7341
256	0.4748	0.6251	0.7906	0.1073	0.8144
512	**0.4246**	**0.5990**	**0.7739**	**0.0924**	**0.8192**
1024	0.4395	0.6453	0.8033	0.6528	0.8020
2048	0.5025	0.7338	0.8566	0.0954	0.7645

of all our trials, we found that 450 epochs resulted in the lowest values on all error metrics. By training the model for 450 epochs, the error metrics are significantly decreased: MAE from 0.568 to 0.475, MSE from 0.745 to 0.625, and RMSE from 0.862 to 0.790. Further increasing the number of epochs for training beyond 450 epochs showed signs of overfitting. Thus, 450 epochs is considered as the best choice.

We continue manipulating the training requirements of the model to improve its performance. With the number of epochs also optimized along with the other original hyperparameters, the only parameter that needs to be optimized is batch size.

Similar to the trials to optimize the number of epochs, we conduct trials to optimize the batch size parameter by testing the model at a variety of batch sizes (64, 128, 256, 512, 1024), with a fixed number of epochs of 450 to remain consistent. The results are shown in Table VI. It can be seen that a batch size of 512 yields the lowest Mean Absolute Error (0.42), Mean Squared Error (0.59), and Root Mean Squared Error (0.77) as well as the highest R-Squared of 82.24% (see Table VII). Thus, we identify 512 as the optimal batch size of training examples used per iteration when training the model.

Now, we conclude the optimization phase of our model's development and confirm the final set of hyperparameters for the model: [Number of Epochs: 450, Batch Size: 512, Dropout Value: 0.415, Weight Decay: 0.283, Number of Generated Layers: 2.014, Layer 1 Size: 166.6, Layer 2 Size: 745.6, Layer 3 Size: 1.552].

VI. EVALUATION

A. Prediction Performance

We first evaluate the performance of the CVSS base score prediction model with the optimized hyperparameters. The results are shown in Table VII. Out of five separate tests, the model yielded an average MAE of 0.4209, MSE of 0.5899, RMSE of 0.7680, MAPE of 0.0919, R-squared of 0.8224.

TABLE VII
FINAL RESULTS WITH **HYP. COMB. 4**, TRAINING FOR 450 EPOCHS, AND A BATCH SIZE OF 512

Test	MAE	MSE	RMSE	MAPE	R^2
1	0.4246	0.5990	0.7739	0.0924	0.8192
2	0.4166	0.5757	0.7587	0.0941	0.8267
3	0.4226	0.5861	0.7656	0.0819	0.8232
4	0.4220	0.5959	0.7720	0.0854	0.8216
5	0.4186	0.5926	0.7698	0.1054	0.8215
AVG	**0.4209**	**0.5899**	**0.7680**	**0.0919**	**0.8224**

TABLE VIII
CVSS QUALITATIVE SEVERITY RATING SCALE

Rating	CVSS Score
None	0.0
Low	0.1 - 3.9
Medium	4.0 - 6.9
High	7.0 - 8.9
Critical	9.0 - 10.0

With an average R^2 value of 82.24%, it can be seen that the model explains approximately 82.24% of the variance of the possible CVSS scores on the CVSS v3 scale. This metric paired with the large size of the dataset tells us that our model is able to identify a strong correlation between vulnerabilities' text descriptions and their CVSS Base Scores.

Since the average Mean Absolute Error, i.e., the average absolute difference between predicted and actual values, is only 0.4209 and the CVSS v3 score ranges from 0.0 to 10.0, we can see that this model is quite accurate in determining the CVSS base score. Similarly, the Mean Squared Error and Root Mean Squared Error are also very low and are improved significantly from the beginning of the optimization stage, thus showing the effectiveness of our optimization strategy.

Additionally, we refer to the Qualitative Severity Rating Scale in [7] and Table VIII which classifies numerical CVSS scores into categories of None, Low, Medium, High, or Critical, and then sort our predicted and actual scores into these qualitative ratings. Upon comparing the predicted and actual CVSS severity ratings, we found that approximately 83.21% of the vulnerabilities in our testing dataset were classified correctly by our model.

B. Comparison to Existing Models

Then we compare our solution with two state-of-the-art approaches [15] and [17]. The implementation of those two approaches adopted the code from GitHub repositories [31] and [32] provided in [15] and [17], respectively, and the parameter configurations in their original publications [15] and [17] were also adopted. We run them and our proposed model on the dataset described in Section V-A1, with 70% of the data used for training and 30% used for testing. For the model of [15], according to their documentation [33], it is first trained for two warm-up epochs, and then it is trained for two more epochs because their method determines that two additional full epochs will be optimal.

Since the two baseline approaches focus on predicting different categorical variables, specifically ratings for Con-

TABLE IX
CVSS BASE SCORE PREDICTION COMPARISON RESULTS

Model	Accuracy	Precision	Recall	f1-Score
CVSS-BERT [6]	82.53%	0.8229	0.8253	0.8234
DistilBERT [9]	75.87%	0.5756	0.7587	0.6546
Our Model	84.55%	0.8588	0.8454	0.8471

fidentiality, Integrity, Availability, and Overall Severity, as opposed to our model which directly predicts CVSS base scores, for fair comparisons, we convert our model's CVSS base score predictions into CVSS Severity Levels using the ranges listed in Table VIII. As shown in Table IX, our solution performs better than the other two approaches in accuracy, precision, recall, and f1-score.

VII. CONCLUSION

We focused on efficiently automating the severity classification process for CVSS base scores to shorten the time needed and reduce security risks for electric utilities. The prediction model that we developed performed very well in predicting the CVSS base scores from human-generated text descriptions and two supporting binary identifiers (hasExploit, hasFix). It generated more accurate predictions than two state-of-the-art solutions. Our scheme is also efficient, through implementing a Fast.ai-based neural network with a low-maintenance text vectorization tool.

ACKNOWLEDGMENT

This work is supported in part by NSF under award 1751255, and also by the Arkansas High Performance Computing Center which is funded through multiple NSF grants and the Arkansas Economic Development Commission.

REFERENCES

[1] *National Vulnerability Database*. National Institute of Standards and Technology. URL: https://nvd.nist.gov/.

[2] *Offensive Security's Exploit Database Archive (Exploit-DB)*. Offensive Security. URL: https://www.exploit-db.com/.

[3] Philip Huff, Kylie McClanahan, Thao Le, and Qinghua Li. "A Recommender System for Tracking Vulnerabilities". In: *International Conference on Availability, Reliability and Security (ARES)*. 2021.

[4] Yatish Dubasi, Ammar Khan, Qinghua Li, and Alan Mantooth. "Security Vulnerability and Mitigation in Photovoltaic Systems". In: *IEEE International Symposium on Power Electronics for Distributed Generation Systems (PEDG)*. 2021, pp. 1–7.

[5] Fengli Zhang and Qinghua Li. "Security Vulnerability and Patch Management in Electric Utilities: A Data-Driven Analysis". In: *The 1st Radical and Experiential Security Workshop (RESEC)*. 2018.

[6] *Common Vulnerabilities and Exposures(CVE)*. The MITRE Corporation. URL: https://cve.mitre.org/.

[7] *Common Vulnerability Scoring System version 3.1: Specification Document*. Forum of Incident Response and Security Teams. URL: https://www.first.org/cvss/specification-document.

[8] Jukka Ruohonen. "A look at the time delays in CVSS vulnerability scoring". In: *Applied Computing and Informatics* 15.2 (2019), pp. 129–135.

[9] Philip Huff and Qinghua Li. "Towards Automated Assessment of Vulnerability Exposures in Security Operations". In: *EAI International Conference on Security and Privacy in Communication Networks (SecureComm)*. 2021, pp. 62–81.

[10] Fengli Zhang, Philip Huff, Kylie McClanahan, and Qinghua Li. "A Machine Learning-based Approach for Automated Vulnerability Remediation Analysis". In: *IEEE Conference on Communications and Network Security (CNS)*. 2020, pp. 1–9.

[11] Fengli Zhang and Qinghua Li. "Dynamic Risk-Aware Patch Scheduling". In: *IEEE Conference on Communications and Network Security (CNS)*. 2020, pp. 1–9.

[12] Kylie McClanahan and Qinghua Li. "Automatically Locating Mitigation Information for Security Vulnerabilities". In: *IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. 2020, pp. 1–7.

[13] Marie Louise Uwibambe, Yanjun Pan, and Qinghua Li. "Fuzzing for Power Grid Systems: A Comparative Study and New Approaches". In: *IEEE Design Methodologies Conference (DMC)*. 2023.

[14] Dakota Dale, Kylie McClanahan, and Qinghua Li. "AI-based Cyber Event OSINT via Twitter Data". In: *International Conference on Computing, Networking and Communications (ICNC)*. 2023, pp. 436–442.

[15] Mustafizur R Shahid and Hervé Debar. "CVSS-BERT: Explainable Natural Language Processing to Determine the Severity of a Computer Security Vulnerability from its Description". In: *IEEE Int'l Conf. on Machine Learning and Applications (ICMLA)*. 2021, pp. 1600–1607.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[17] Joana Cabral Costa, Tiago Roxo, João B. F. Sequeiros, Hugo Proenca, and Pedro R. M. INÁCIO. "Predicting CVSS Metric via Description Interpretation". In: vol. 10. 2022, pp. 59125–59134. DOI: 10.1109/ACCESS.2022.3179692.

[18] Atefeh Khazaei, Mohammad Ghasemzadeh, and Vali Derhami. "An automatic method for CVSS score prediction using vulnerabilities description". In: *Journal of Intelligent & Fuzzy Systems* 30.1 (2016), pp. 89–96.

[19] Zhuobing Han, Xiaohong Li, Zhenchang Xing, Hongtao Liu, and Zhiyong Feng. "Learning to Predict Severity of Software Vulnerability Using Only Vulnerability Description". In: *IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME)*. 2017, pp. 125–136.

[20] Kerem Gencer and Fatih Başçiftçi. "The fuzzy common vulnerability scoring system (F-CVSS) based on a least squares approach with fuzzy logistic regression". In: *Egyptian Informatics Journal* 22.2 (2021), pp. 145–153.

[21] Clément Elbaz, Louis Rilling, and Christine Morin. "Fighting N-day vulnerabilities with automated CVSS vector prediction at disclosure". In: *The International Conference on Availability, Reliability and Security*. 2020, pp. 1–10.

[22] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[23] Quoc Le and Tomas Mikolov. "Distributed representations of sentences and documents". In: *International conference on machine learning*. PMLR. 2014, pp. 1188–1196.

[24] Jeremy Howard and Sylvain Gugger. "Fastai: a layered API for deep learning". In: *Information* 11.2 (2020), p. 108.

[25] Rita Yi Man Li, Herru Ching Yu Li, Beiqi Tang, and WaiCheung Au. "Fast AI Classification for Analyzing Construction Accidents Claims". In: *Proceedings of the Artificial Intelligence and Complex Systems Conference*. 2020, pp. 1–4.

[26] Julian Martin Eisenschlos, Sebastian Ruder, Piotr Czapla, Marcin Kardas, Sylvain Gugger, and Jeremy Howard. "MultiFiT: Efficient multi-lingual language model fine-tuning". In: *arXiv preprint arXiv:1909.04761* (2019).

[27] Jey Han Lau and Timothy Baldwin. "An empirical evaluation of doc2vec with practical insights into document embedding generation". In: *arXiv preprint arXiv:1607.05368* (2016).

[28] *Gensim: Topic Modelling For Humans*. URL: https://radimrehurek.com/gensim/auto_examples/tutorials/run_doc2vec_lee.html.

[29] *VulnIQ: Vulnerability Intelligence and Management Solution*. VulnIQ. URL: https://www.vulniq.com/.

[30] Apoorv Agnihotri and Nipun Batra. "Exploring bayesian optimization". In: *Distill* 5.5 (2020), e26.

[31] Mustafizur Rahman Shahid and Hervé Debar. Jan. 2022. URL: https://github.com/mus-shd/CVSS-BERT.git.

[32] Joana Cabral Costa, Tiago Roxo, JOÃO SEQUEIROS, HUGO PROENÇA, and PEDRO INÁCIO. *Predicting CVSS Metric via Description Interpretation (GitHub)*. Nov. 2022. URL: https://github.com/Joana-Cabral/CVSS_Prediction.

[33] URL: https://github.com/mus-shd/CVSS-BERT/blob/main/demo_notebook.ipynb.