2024 IEEE 15th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON) | 979-8-3315-4090-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/UEMCON62879.2024.10754709

Enhanced Categorization of Cybersecurity Vulnerabilities

Ocheme Anthony Ekle
Department of Computer Science
Tennessee Technological University
Cookeville, USA
oaekle42@tntech.edu

Denis Ulybyshev
Department of Computer Science
Florida Polytechnic University
Lakeland, USA
dulybyshev@floridapoly.edu

***Abstract*—According to the public National Vulnerability Database, maintained by the National Institute of Standards and Technology, the number of registered cybersecurity vulnerabilities keeps growing despite all known techniques for developing secure software and detecting vulnerabilities. Categorizing cybersecurity vulnerabilities is essential for choosing effective protection mechanisms to prevent or mitigate vulnerabilities in software and reduce cyber risks. In this paper, we present a machine learning-based methodology to categorize cybersecurity vulnerabilities using their descriptions registered in the public National Vulnerability Database. We will show that handling imbalanced data with Synthetic Minority Over-sampling Technique improves the categorization model accuracy by 20%.**

***Index Terms*—software security, machine learning, cybersecurity vulnerability categorization**

I. Introduction

New cybersecurity vulnerabilities are being constantly discovered in software. Many of these vulnerabilities are registered in the public database of Common Vulnerabilities and Exposures (CVE) maintained by the MITRE Corporation since 1999 [1], as well as in the National Vulnerability Database (NVD) [2], maintained by the National Institute of Standards and Technology (NIST). According to these public databases, there are 240,830 vulnerabilities, as of August 27, 2024. Defending against cyber attacks that aim to exploit certain cybersecurity vulnerabilities is essential to protect individual and corporate assets. In this paper, we propose a machine learning-based categorization technique with improved accuracy for categorizing cybersecurity vulnerabilities that are registered in the public CVE [1] and NVD [2] databases. Accurate results of cybersecurity vulnerability categorization can help to choose the most effective mitigation strategies to make computing systems less vulnerable against cyber attacks. Categorization also helps to determine most common cybersecurity vulnerabilities. The rest of the paper is organized as follows: in Chapter II we review the related work; Chapter III presents the core design of our solution, which is evaluated in Chapter IV. Chapter V discusses our findings and conclusions.

II. Related Work

Methodologies to assess and categorize software vulnerabilities have been proposed by researchers in past decades. Neuhaus and Zimmermann [3] proposed a solution in finding security trends using CVE records. Their approach is able to find types of security weaknesses in the CVE records. We aim to categorize software vulnerabilities with improved accuracy.

Blinowski and Piotrowski [4] performed a CVE-based classification of vulnerable IoT systems. They used SVM algorithm to classify vulnerability records on selected subset of CVEs "that describe vulnerabilities of potential IoT devices of different applications, such as: home, industry, mobile controllers, networking, etc. " [4] "VulDB" [5] solution categorizes multiple types of software products and offers good statistics on vulnerabilities. However, it is closed-source and may not have categories that a user needs. Our work focuses on categorizing software vulnerabilities, not software products.

Bozorgi et al. [6] proposed a comprehensive approach for vulnerability assessment using data mining and machine learning tools. The authors studied the possibility of exploiting a vulnerability and how quickly it would be exploited. They utilized the dataset features from the Open Source Vulnerability Database (OSVDB) and the CVE database. Their study also employed the reference feature from the vulnerability information and converted it into binary form using the bag-of-words technique. In our case, we utilized records from the NVD website [2] to categorize software vulnerabilities by applying Synthetic Minority Over-Sampling Technique (SMOTE) techniques on imbalanced data.

Edkrantz and Said [7] used five different machine learning algorithms: Naive Bayes, Liblinear, LibSVM linear, LibSVM RBF, and Random Forest to examine the correlations in the vulnerability data from the NVD and the Exploit Database (EDB) to see if some vulnerability types are more likely to be exploited. In our solution, we employed the Naive Bayes, Random Forest, Decision Tree, and LightGBM classifiers in categorizing software vulnerability types. Bhatt et al. [8] conducted research on "modeling and characterizing software vulnerabilities," exploring different aspects of their characteristics and behavior. Our study aims to utilize machine learning-based approaches to improve the accuracy and precision of categorizing software vulnerabilities. Wang and Guo [9] also proposed a vulnerability categorization method using Bayesian networks. Their study focused on utilizing Bayesian networks to categorize vulnerabilities, providing a systematic approach

to identify and classify different types of vulnerabilities based on their characteristics and relationships.

Chen et al. [10] focused on predicting invalid vulnerabilities in the CVE database. The study aims to understand the root causes of rejected CVEs and proposes a text mining approach for predicting invalid vulnerability reports. Kiran et al. [11] worked on categorizing 2019 - 2020 CVEs based on vulnerability software. The authors worked on data analysis, considering factors such as Common Vulnerability Scoring System (CVSS) scores, Common Weakness Enumeration (CWE) type, and Common Platform Enumeration (CPE) applicability reports. Whereas they only capture records for two years, our research incorporates a dataset from a 20-year scope (1999–2019). We applied advanced feature engineering techniques and predicted the software vulnerability types with high precision and accuracy.

Yosifova et al. [12] conducted research similar to ours, focusing on the application of machine learning classifiers to predict vulnerability types in CVE database. Their study aimed to enhance vulnerability management by categorizing vulnerabilities through the analysis of CVE records, feature extraction, and the use of machine learning techniques. Compared to [4, 11, 12], we advanced the solution by incorporating advanced feature engineering, text preprocessing techniques, and we applied SMOTE algorithm to address the dataset imbalance problem, thereby improving the accuracy and precision of predictions by 20% over the baseline performance.

III. Core Design

In this section, we present the design of our framework for CVE categorization. We explain the details of the entire process, starting from data collection to data preprocessing and feature engineering. We employed a NLP (Natural Language Processing) toolkit and the TF-IDF Transformer library to convert the textual data into vector representations. Additionally, we addressed the issue of imbalanced data distribution by utilizing the *Synthetic Minority Over-Sampling Technique* (SMOTE) method. To build our classification model, we employed the Multinomial Naive Bayes (MNB), Random Forest (RF), and Linear SVC classifiers using Python programming language and Scikit-learn libraries. These classifiers were applied to the training set, as well as the LightGBM classifier. We evaluated the performance of each classifier by testing the model on a separate testing dataset and measuring accuracy, precision, recall, and F1-score. Remarkably, our final model achieved a 20% increase in accuracy compared to the baseline results reported in [12]. Subsections below elaborate on the tools and techniques that were instrumental in our cybersecurity vulnerability classification task. A comprehensive overview of our framework is illustrated in Figure 1

A. Data Collection

The dataset used in this study was extracted from the public GitHub repository of Yosifova et al. [12], who conducted a study on vulnerability type classification. This data was scraped from the CVE Security Vulnerability Database [13],

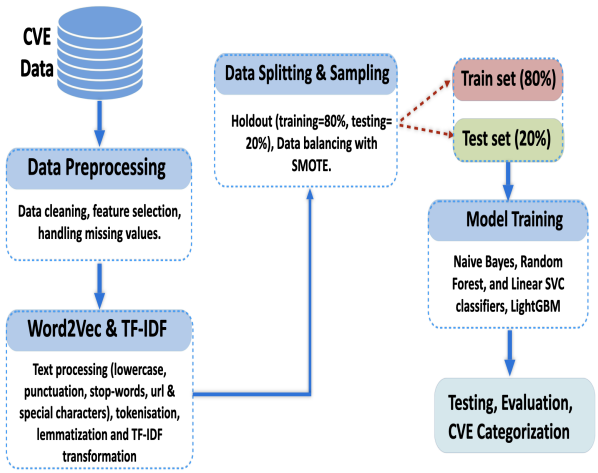

Fig. 1: CVE Categorization System Architecture

which provides comprehensive information about CVEs. The raw dataset obtained from these sources contains CVEs spanning over a period of 20 years, from 1999 to 2019, comprising a total of **123,000** records. It consists of 15 columns representing various attributes associated with each CVE, such as CVE-ID, CVSS score, mode of Access, complexity of attack, detailed description of attack containing vulnerability types and others. The dataset contains 13 unique vulnerability types, shown in Figure 2. To enhance clarity and facilitate further analysis, we renamed and encoded each vulnerability type using numerical values. The initial categorization of vulnerabilities in the raw dataset included 14 categories: *'NaN', 'DoS', 'Exec Code', 'Overflow', 'XSS', 'Dir. Trav.', 'Bypass', '+Info', '+Priv', 'Sql', 'File Inclusion', 'Mem. Corr.', 'CSRF', 'Http R.Spl.'*. We transform the categories into the following numerical representations: *0: '+Gain-Information'; 1: '+Gain-Privilege'; 2: 'Bypass-Something'; 3: 'CSRF'; 4: 'Denial-of-Service'; 5: 'Directory-Traversal'; 6: 'Execute-Code'; 7: 'File-Inclusion'; 8: 'Http-Response-Splitting'; 9: 'Memory-Corruption'; 10: 'Overflow'; 11: 'Sql-Injection'; 12: 'XSS'*. We noticed that some CVE records contain multiple vulnerability types; hence, we decided to separate each vulnerability type into unique records. This process resulted in an increase in the total number of CVE records or entries, generating a new dataset with **162,789** records. The distribution of CVEs across different vulnerability types is shown in Figure 2.

B. Data Preprocessing

The preprocessing step is a crucial component of our CVE classification pipeline, as it involves transforming our raw text data into a format suitable for machine learning training. Various common techniques are employed for data preprocessing, including data cleaning, feature selection, handling missing values, addressing imbalanced datasets, and dimensionality reduction. The specific techniques utilized vary depending on the characteristics of the data.

Considering the nature of our dataset, as discussed in Section III-A, we first removed all the records with missing

979-8-3315-4090-6/24/$31.00 ©2024 IEEE

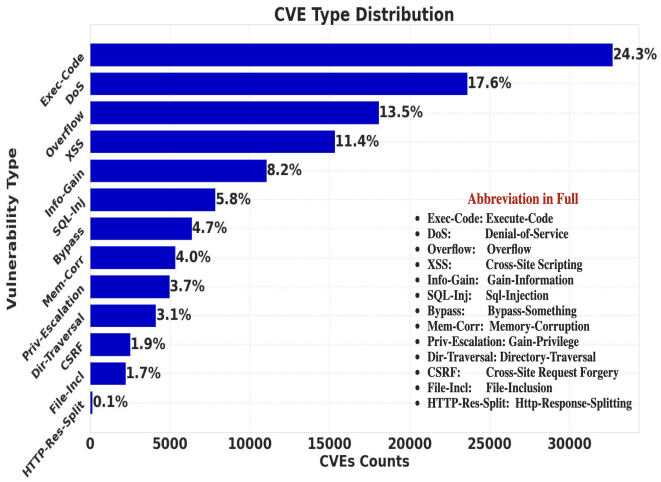

Fig. 2: CVE Type Distribution

vulnerability types. Among the **162,827** records generated in the final stage of data collection, we identified **28,430** instances with missing values, accounting for approximately 17% of the dataset. After eliminating the missing data, we proceeded with *feature selection*, specifically extracting the *"CVEs Description"* feature as our attribute X and the *"Vulnerability Type"* as the label Y. The label encoding for all 13 vulnerability types was performed using the *LabelEncoder* package from the *sklearn.preprocessing* Python library. As a result, we had **134,397** records after dropping the missing values and all other columns. The resulting dataset comprises two columns: the encoded vulnerability type and the vulnerability description, which will be further processed using NLP toolkits.

C. Vector Representation and TF-IDF Transformer

Since we are working with the text data for our CVE classification task, which falls under the NLP domain, we need to perform three essential phases to transform our text data into vectors. These phases include text preprocessing, tokenization, and vectorization. *Text preprocessing* involves cleaning and normalizing the text data by removing noise and irrelevant information [14]. In our approach, we employed several techniques to preprocess text data and performed the following steps: (1) Convert all text to lowercase; (2) Remove all punctuation marks from the text, as they often carry little or no semantic meaning; (3) Remove stop words, which are common English words that frequently occur in sentences but do not contribute significantly to the overall meaning, such as "a," "the," "and," etc; (4) Eliminate special characters, symbols, and numbers; (5) Remove all URLs from our text data, as they do not provide meaningful information for our task. These preprocessing techniques help to clean the text data for further analysis and vectorization. Next, we performed *word tokenization*, which implies breaking the text documents into smaller units called tokens [15]. *Tokenization* can be performed using different techniques, namely, word tokenization, sentence tokenization, and character tokenization. In our case, we did word tokenization. For instance, the sentence "heap-based buffer overflow" would be tokenized as ["heapbased", "buffer", "overflow"]. We performed lemmatization to break the words down to their root meaning to identify similarities and classify them.

TF−IDF is a statistical measure used in NLP tasks to convert text to vector representation and to capture the importance weights of terms (i.e., words or k-grams) within a text document [16]. It entails two major metrics: Term Frequency (TF), which captures the frequency of terms in text, and Inverted Document Frequency (IDF), which measures how rare a term is in a given entire list of documents. We utilized the *TfidfVectorizer* provided by the *Scikit-Learn* library and set specific parameters for instantiation. Subsequently, the resulting TF−IDF transformed matrix is employed as input for our machine learning classifiers.

D. Data Splitting and Sampling Technique

Data splitting is an essential and pivotal process in the building of machine learning models. It aims to divide the dataset into distinct subsets, serving the objectives of training and testing. This technique ensures that we adequately test and measure the performance of our model using previously unseen data. To accomplish this, we employed the *Holdout technique* [17], which segregates the data into two sets that do not overlap: the training set and the testing set. The ratio we adopted for this division was 80 : 20, with 80% allocated for training and 20% for testing. Additionally, we enabled the "*random_state*" parameter to ensure the shuffling of our dataset during the split.

E. Handling Imbalance data with Synthetic Minority Oversampling Technique (SMOTE)

Most machine learning datasets commonly encounter the issue of class imbalance, wherein the distribution of data across different classes is uneven. This imbalance can lead to biased machine learning algorithms and poor predictions for the minority class(es). Various approaches are used to address this problem, including data resampling, class weighting, ensemble methods, and cost-sensitive learning. In our approach, we used *SMOTE* [18], a popular resampling technique utilized to tackle imbalanced datasets in classification tasks. The primary objective of SMOTE is to generate synthetic samples for the minority class by interpolating existing samples from that class. In our case, we applied SMOTE separately to both the training set and the test set to avoid data bias and skewness in our data distribution. By using SMOTE, we increased our final processed data from **137,349** samples to **425,165** samples when considering both the training and testing sets combined. For a *trade-off* between accuracy and time, it is important to note that although SMOTE provides greater accuracy compared to random under-sampling, it requires more time for training since no rows are eliminated, as previously mentioned.

F. Model Training and Testing Phase

In our study, we constructed two sets of CVE categorization models for the CVE classification task. Firstly, we developed

classifiers without using SMOTE-implemented data. Then we built a second set of models using SMOTE-implemented data.

For the classifiers without SMOTE data, we had a total of $137,349$ samples available for model training and testing. These samples were divided into an 80% training set, which accounted for 107,517 samples, and a 20% testing set, as detailed in Section III-D. To ensure unbiased results, we applied SMOTE separately to the training and testing datasets that were already split, according to Section III-D. This process resulted in the generation of $341,224$ training samples and $83,941$ testing samples. In total, we had $425,165$ samples after incorporating SMOTE. The classifiers we utilized in our experiment include Naive Bayes, Random Forest, and Decision Tree, which were also employed as baseline models in [12]. Additionally, we implemented the LightGBM classifier using Python and the Scikit-Learn libraries.

Naive Bayes algorithm is a probabilistic method commonly utilized for text classification tasks, particularly in NLP tasks such as spam filtering and sentiment analysis [19]. It is based on Bayes' theorem and assumes that the features are conditionally independent given the class label [20]. We employed the Multinomial Naive Bayes (MNB) classifier [21], which is specifically designed for classifying data with discrete features [12]. For our experiments, we utilized TF-IDF-transformed values discussed in Section III-C as input features. The hyper−parameters used include Laplace smoothing with an *alpha*$= 1.0$*, fit_prior= "True"* and *class_prior= "None."*

Random Forest (RF) classifier [22] is an ensemble-based classification algorithm that constructs multiple decision trees by randomly selecting subsets of training samples and variables [23]. It is known for its ability to mitigate overfitting and achieve high accuracy across different domains [24]. In our experiments, we set the hyperparameters as follows: *'n_estimators=100',* indicating that the RF will consist of 100 decision trees. We specified *'max_depth=15'* and *'max_features="sqrt"'*, aiming to reduce overfitting and enhance the diversity of trees within the RF ensemble.

Decision Tree algorithm is a supervised machine learning method utilized for classification and regression tasks [25]. It divides the data recursively into tree-like structures based on feature values [22]. The algorithm selects the most suitable feature at each node using measures like information gain, Gini impurity, or entropy. In our scenario, we specified a *max_depth = 3*, *min_samples_leaf = 5* (representing the minimum number of samples required at a node), and *random_state = 42*.

LightGBM is a gradient-boosting algorithm that uses tree-based structures and is applied for both classification and regression tasks [26]. It is efficient in terms of training speed and memory usage. The algorithm involves randomly discarding data points with lower gradients while retaining others to ensure accuracy. This technique generally outperforms random sampling when the same sampling rate is applied [27]. We used the default parameters for our *LGBMClassifier*, namely, *boosting_type = 'gbdt'*, *num_leaves = 31*, *learning_rate = 0.1*, *n_estimators = 100*, *objective = 'binary'* (i.e., loss function to be optimized during training), and *random_state = 'None'*.

IV. Evaluation

We conducted experiments on our preprocessed dataset, which consisted of **137,349** samples without SMOTE and **425,165** samples with SMOTE. We applied different machine learning classification algorithms during the experiments.

A. Experimental Setup

All experiments were conducted on the following hardware configuration: Apple® M1 Chip, equipped with an 8-Core™ , 64-bit processor running at 3.2 GHz, 7-Core™ GPU, 16-Core™ Neural Engine, 8 GB of RAM, 256GB SSD storage.

Operating system is MacOS® Ventura 13.3.1.

The implementation of all machine learning algorithms was done using the Python® programming language, detailed documentation was created on Jupyter Notebooks™ [28]. For training the models in all experiments, Google ™ Colab [29] was utilized, which had Nvidia® Tesla™ T4 graphics enabled.

B. Evaluation Metrics

In our solution for CVE categorization, we employed the commonly used classification metrics, including accuracy, precision, recall, and F1-Score, which were also utilized as baseline metrics in [12].

Accuracy measures the overall correctness of predicted classes by calculating the ratio of correctly classified CVE instances to the total number of instances in the dataset.

$$\text{Accuracy} = \frac{\text{Number of Correctly Classified Instances}}{\text{Total Number of Instances}} \quad (1)$$

Precision measures the correctly predicted positive instances.

$$P = \frac{TP}{TP + FP} \quad (2)$$

while ***Recall*** (or Sensitivity) captures all positive instances out of the total number of actual positive instances.

$$R = \frac{TP}{TP + FN} \quad (3)$$

where **TP** is True Positives, **FP** - False Positives, **FN** - False Negatives, **TN** - True Negatives.

F1-score, a combined measure of precision and recall using the harmonic mean, provides an overall evaluation of the model's performance.

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4)$$

For our classification task, we employed the *precision_recall_fscore_support* utility from the *scikit−learn* library to calculate the accuracy, precision, recall, and F1−score. This utility allowed us to obtain these metrics in a comprehensive manner for multiple classes [30]. To account for the imbalanced nature of our data, we used the harmonic weighted average when computing the overall metrics for each model. The weighted precision, recall, and F1-score formulas are given in equations (5), (6), and (7), respectively.

$$\bar{P} = \frac{(w_1 \cdot P_1 + w_2 \cdot P_2 + \ldots + w_n \cdot P_n)}{(w_1 + w_2 + \ldots + w_n)} \quad (5)$$

$\bar{P}$ represents the weighted precision, $w_1, w_2, \ldots, w_n$ are the weights assigned to each class, and $R_1, R_2, \ldots, R_n$ represent the precision of each class.

$$\bar{R} = \frac{(w_1 \cdot R_1 + w_2 \cdot R_2 + \ldots + w_n \cdot R_n)}{(w_1 + w_2 + \ldots + w_n)} \tag{6}$$

$\bar{R}$ represents the weighted recall, $w_1, w_2, \ldots, w_n$ are the weights assigned to each class, and $R_1, R_2, \ldots, R_n$ are the individual recalls of each class.

$$\bar{F1} = \frac{(w_1 \cdot F1_1 + w_2 \cdot F1_2 + \ldots + w_n \cdot F1_n)}{(w_1 + w_2 + \ldots + w_n)} \tag{7}$$

$\bar{F1}$ represents the weighted F1-score, $w_1, w_2, \ldots, w_n$ are the weights assigned to each class, and $F1_1, F1_2, \ldots, F1_n$ are the individual F1-scores. The weighted average took into account the individual support (i.e., the number of occurrences) of each class, thereby obtaining a more accurate representation of the overall performance of the models.

C. Experimental Results and Discussion

In this section, we provide a detailed analysis of the results, including the following: (1) the outcome of our initial (first) models that did not utilize SMOTE; (2) the models implemented with SMOTE; (3) the visual representation of the confusion matrix, showing how accurately our models predicted the cybersecurity vulnerability types.

Result 1: Baseline Model vs. First Model

Table I presents the experimental results and performance comparison between our *First Model* without SMOTE on imbalanced dataset and the baseline model from [12]. Our models were trained on an imbalanced dataset comprising $134,397$ CVE records. Our initial models exhibited 5% and 2% improvements over the baseline model [12] in terms of accuracy. Specifically, our **Naive Bayes** model obtained an accuracy of **0.64**, surpassing the baseline accuracy of **0.57** by **5**$\%$. Our **RF** model achieved an accuracy of **0.65**, which represents a **2**$\%$ improvement over the baseline score in [12].

TABLE I. Model Performance with Imbalanced Dataset

	Methods	Accuracy	Precision	Recall	F1-score
1^{st} Model Baseline [12]	Naive Bayes	**0.57**	0.65	0.59	0.62
	Random Forest	**0.63**	0.76	0.72	0.74
	Linear SVC	0.68	0.67	0.66	0.73
	Performance without SMOTE				
	Decision Tree	0.51	0.46	0.51	0.43
	Naive Bayes	**0.64**	0.67	0.64	0.63
	Random Forest	**0.65**	0.64	0.65	0.61
	LightGBM	0.61	0.60	0.61	0.60

The improvement is due to the application of advanced feature engineering techniques described in Sections III-B and III-C. However, our Decision Tree and LightGBM classifiers performed below the baseline model, with accuracy scores of **0.51** and **0.61,** respectively, as shown in Table I. This discrepancy can be attributed to the *models' sensitivity* to imbalanced datasets; we address this by employing SMOTE.

Result 2: Baseline Model vs. Second Model with SMOTE

The experimental results for our Second Model that uses balanced dataset with SMOTE are shown in Table II. Initially, we had **137,349** preprocessed samples, but after applying SMOTE to balance the dataset, the number increased to **425,165**. This technique was applied separately to the training and testing sets to address any bias errors. We implemented the Naive Bayes, RF, and LightGBM classifiers. The Naive Bayes classifier yielded an accuracy score of **0.80**, precision score of 0.79, recall score of 0.80, and F1 score of 0.79. The RF classifier showed an accuracy of **0.81**, precision of 0.84, recall of 0.81, and F1-score of 0.78. The LightGBM classifier performed even better, with an accuracy of **0.84**, precision of 0.84, recall of 0.84, and F1-score of 0.83.

TABLE II. Model Performance with SMOTE Balanced Dataset

	Methods	Accuracy	Precision $\bar{P}$	Recall $\bar{R}$	F1-score
2^{nd} Model	Naive Bayes	**0.80**	0.79	0.80	0.79
	Random Forest	**0.81**	0.84	0.81	0.78
	LightGBM	**0.84**	0.84	0.84	0.83

Parameters $\bar{P}$ and $\bar{R}$ represent the weighted precision, and weighted recall respectively.

Notably, the LightGBM classifier obtained optimal performance in predicting types of cybersecurity vulnerabilities, outperforming the baseline results [12] by 20%. The novel performance and wide margin between our initial result in Table I and final model in Table II validate the importance of feature engineering techniques discussed in Section III-B and a methodology discussed in Section III-E.

Result 3: Confusion Matrix Visualization and Analysis for Predicted CVE Types

The heatmap matrix was used to evaluate the performance of our multi-class categorization task, specifically to visualize the predicted cybersecurity vulnerability types against the actual values. Figures 3, 4, 5 and 6 provide a detailed visualization of how our **Naive Bayes** and **LightGBM** models performed with and without the implementation of SMOTE in capturing vulnerability classes. The matrix is plotted with true values on the *X*-axis and model-predicted values on the *Y*-axis. The diagonal values from the top left to the bottom right measure the accuracy of the model's predictions. The darker colors indicate a higher frequency of correct predictions, while lighter colors represent lower frequency of correct predictions.

Result 3.1. Naive Bayes: Figure 3 shows the prediction accuracy of the Naive Bayes without SMOTE, and Figure 4 shows the Naive Bayes with SMOTE. In Figure 3, "*DoS*", "*Exec-code*", and "*XSS*" are the best-classified vulnerability types, with a weighted average precision of 67% and accuracy of 64% (see Figure 3 and Table I). This is because these

vulnerability types appear more frequently in the CVE data distribution, as shown in Figure 2. However, in Figure 4, the model with SMOTE correctly predicted **11** out of **13** vulnerability types with a weighted average precision of 79% and an accuracy of 80%.

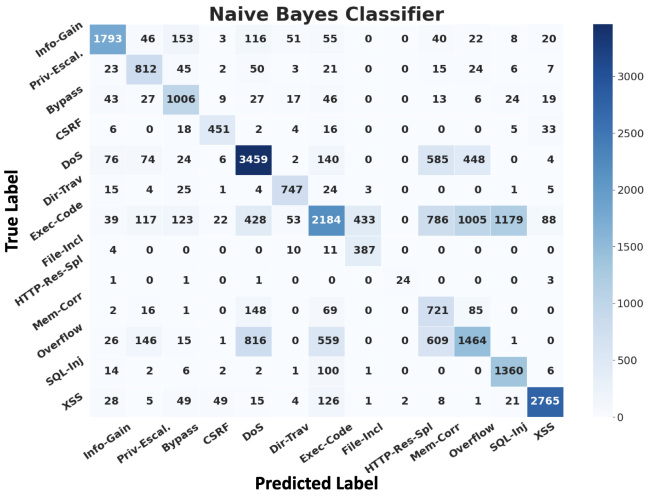

Fig. 3: Predicted Vulnerability Type Distribution using a Naive Bayes Classifier without SMOTE

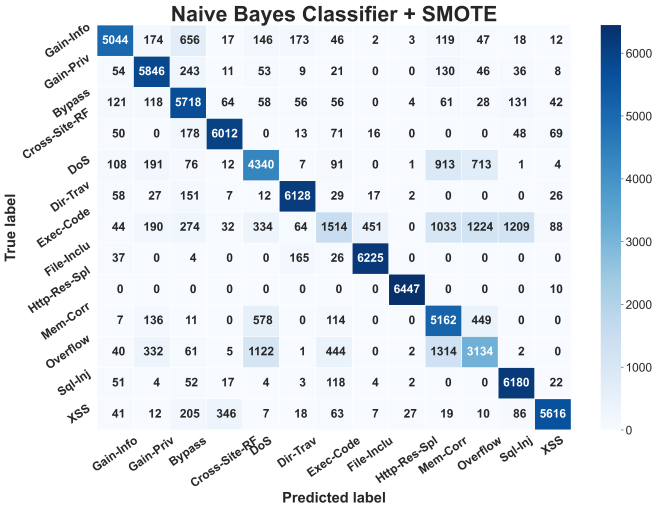

Fig. 4: Predicted Vulnerability Type Distribution using a Naive Bayes Classifier with SMOTE

Result 3.2. LightGBM: Figures 5 and 6 show the model performance of the LightGBM classifier with and without SMOTE. Similar to the performance of Naive Bayes, Figure 5 indicates that "DoS," "Exec-code," and "XSS" emerged as the best-classified vulnerability types, with a precision of 60% and accuracy of 61%. As shown in Figure 6, the **LightGBM+SMOTE** model correctly predicted **12** out of **13** vulnerability types, achieving a weighted average precision of 84% and an accuracy of 84%. This is represented in the darker diagonal cell in Figure 6. The most accurately predicted vulnerability type is *"File-Inclusion"*. It was best classified by our optimal model (**LightGBM+SMOTE**), with a precision of 96% and recall of 96%. Figure 7 highlights predictions for all vulnerability types.

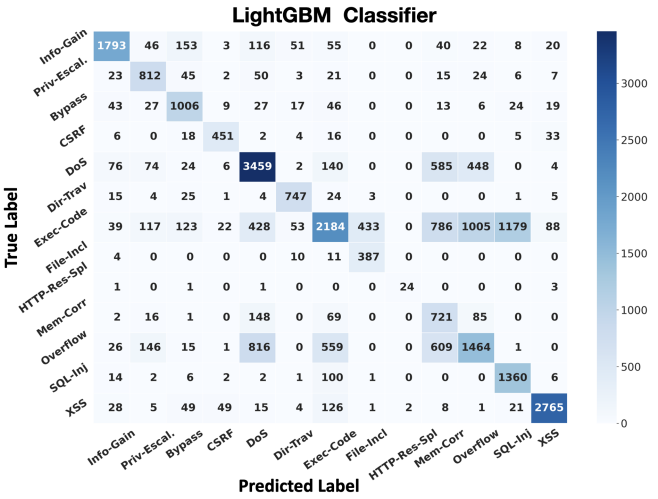

Fig. 5: Predicted Vulnerability Type Distribution using a LightGBM Classifier without SMOTE

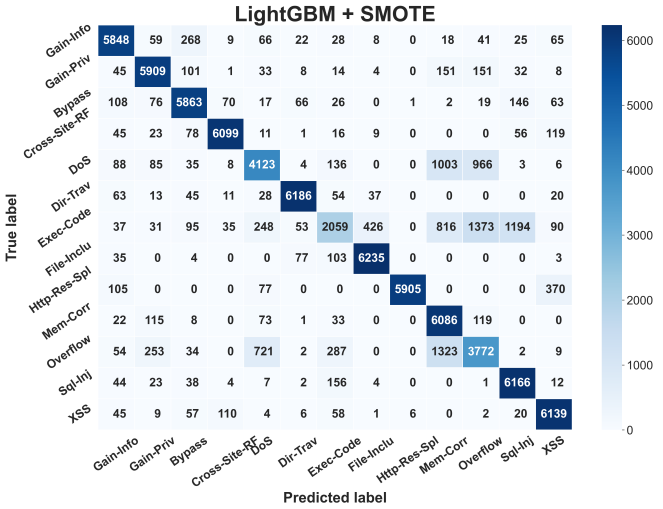

Fig. 6: Predicted Vulnerability Type Distribution with LightGBM Classifier + SMOTE

V. Conclusions

This paper presents a solution for categorization of cybersecurity vulnerabilities in software by utilizing machine learning models trained on a dataset of 134,397 imbalanced CVE records and 425,165 balanced records with the Synthetic Minority Over-sampling Technique (SMOTE). We improved upon the baseline models [12] of Naive Bayes and Random Forest by 5% and 2%, respectively, despite working with imbalanced datasets. The experimental result with Multinomial Naive Bayes, Random Forest, and LightGBM classifiers achieved accuracy of 80%, 81%, and 84%, respectively, as

LightGBM+SMOTE Classification Report on Test Set

	precision	recall	f1-score	support
0	0.89	0.91	0.90	6457
1	0.90	0.92	0.91	6457
2	0.88	0.91	0.90	6457
3	0.96	0.94	0.95	6457
4	0.76	0.64	0.69	6457
5	0.96	0.96	0.96	6457
6	0.69	0.32	0.44	6457
7	0.93	0.97	0.95	6457
8	1.00	0.91	0.95	6457
9	0.65	0.94	0.77	6457
10	0.59	0.58	0.58	6457
11	0.81	0.95	0.87	6457
12	0.89	0.95	0.92	6457
accuracy			0.84	83941
macro avg	0.84	0.84	0.83	83941
weighted avg	0.84	0.84	0.83	83941

```
Precision: 0.8391347519501998
Recall: 0.8385651826878403
F1-score: 0.8302370380043498
```

Fig. 7: Comprehensive Analysis of the LightGBM+SMOTE Model Classification

shown in Tables I and II. These results highlight the efficacy of our approach in detecting and categorizing various types of cybersecurity vulnerabilities, as shown in Figures 4, 6, and 7. Our approach significantly improves the accuracy of cybersecurity vulnerability type categorization by 20% over existing baseline. This advancement can assist cybersecurity experts in automating the categorization of cybersecurity vulnerabilities and therefore choosing effective cyber protection strategies.

ACKNOWLEDGMENTS

We thank Cybersecurity Education Research and Outreach Center (CEROC), the Departments of Computer Science at Florida Polytechnic University and Tennessee Technological University for providing resources to work on this project.

REFERENCES

[1] MITRE, "Common vulnerability and exposure," https://cve.mitre.org/, accessed: August 27, 2024.
[2] "National vulnerability database," https://nvd.nist.gov/, accessed: August 27, 2024.
[3] S. Neuhaus and T. Zimmermann, "Security trend analysis with cve topic models," in *2010 IEEE 21st International Symposium on Software Reliability Engineering*. IEEE, 2010, pp. 111–120.
[4] G. J. Blinowski and P. Piotrowski, "Cve based classification of vulnerable iot systems," in *International Conference on Dependability and Complex Systems*. Springer, 2020, pp. 82–93.
[5] "Vulnerability Database," https://vuldb.com/, 2022, accessed: August 27, 2024.
[6] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond heuristics: learning to classify vulnerabilities and predict exploits," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 105–114.
[7] M. Edkrantz, "Predicting exploit likelihood for cyber vulnerabilities with machine learning," 2015.
[8] N. Bhatt, A. Anand, V. S. S. Yadavalli, and V. Kumar, "Modeling and characterizing software vulnerabilities," 2017.
[9] J. A. Wang and M. Guo, "Vulnerability categorization using bayesian networks," in *Proceedings of the sixth annual workshop on cyber security and information intelligence research*, 2010, pp. 1–4.
[10] Q. Chen, L. Bao, L. Li, X. Xia, and L. Cai, "Categorizing and predicting invalid vulnerabilities on common vulnerabilities and exposures," in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2018, pp. 345–354.
[11] S. Kiran, S. Rajper, R. A. Shaikh, I. A. Shah, and S. H. Danwar, "Categorization of cve based on vulnerability software by using machine learning techniques," *International Journal*, vol. 10, no. 3, 2021.
[12] V. Yosifova, A. Tasheva, and R. Trifonov, "Predicting vulnerability type in common vulnerabilities and exposures (cve) database with machine learning classifiers," in *2021 12th National Conference with International Participation (ELECTRONICA)*. IEEE, 2021, pp. 1–6.
[13] "CVE Security Vulnerability Database," https://www.cvedetails.com/, 2024, accessed: August 27, 2024.
[14] M. Anandarajan, C. Hill, T. Nolan, M. Anandarajan, C. Hill, and T. Nolan, "Text preprocessing," *Practical text analytics: Maximizing the value of text data*, pp. 45–59, 2019.
[15] J. J. Webster and C. Kit, "Tokenization as the initial phase in nlp," in *COLING 1992 volume 4: The 14th international conference on computational linguistics*, 1992.
[16] S. Qaiser and R. Ali, "Text mining: use of tf-idf to examine the relevance of words to documents," *International Journal of Computer Applications*, vol. 181, no. 1, pp. 25–29, 2018.
[17] C. H. Larcher and H. J. Barbosa, "Evaluating models with dynamic sampling holdout," in *Applications of Evolutionary Computation: 24th International Conference, EvoApplications 2021, Held as Part of EvoStar 2021, Virtual Event, April 7–9, 2021, Proceedings 24*. Springer, 2021, pp. 729–744.
[18] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.
[19] S. Raschka, "Naive bayes and text classification i-introduction and theory," *arXiv preprint arXiv:1410.5329*, 2014.
[20] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, pp. 131–163, 1997.
[21] A. M. Kibriya, E. Frank, B. Pfahringer, and G. Holmes, "Multinomial naive bayes for text categorization revisited," in *AI 2004: Advances in Artificial Intelligence: 17th Australian Joint Conference on Artificial Intelligence, Cairns, Australia, December 4-6, 2004. Proceedings 17*. Springer, 2005, pp. 488–499.
[22] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.
[23] M. Belgiu and L. Drăguţ, "Random forest in remote sensing: A review of applications and future directions," *ISPRS journal of photogrammetry and remote sensing*, vol. 114, pp. 24–31, 2016.
[24] B. Ghojogh and M. Crowley, "The theory behind overfitting, cross validation, regularization, bagging, and boosting: tutorial," *arXiv preprint arXiv:1905.12787*, 2019.
[25] M. Bansal, A. Goyal, and A. Choudhary, "A comparative analysis of k-nearest neighbour, genetic, support vector machine, decision tree, and long short term memory algorithms in machine learning," *Decision Analytics Journal*, p. 100071, 2022.
[26] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in Neural Information Processing Systems*, vol. 30, p. 11.
[27] ESRI, "How lightgbm algorithm works," https://pro.arcgis.com/en/pro-app/latest/tool-reference/geoai/how-lightgbm-works.htm, last Accessed: August 27, 2024.
[28] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay *et al.*, *Jupyter Notebooks-a publishing format for reproducible computational workflows.*, 2016, vol. 2016.
[29] E. Bisong and E. Bisong, "Google colaboratory," *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners*, pp. 59–64, 2019.
[30] C. Goutte and E. Gaussier, "A probabilistic interpretation of precision, recall and f-score, with implication for evaluation," in *Advances in Information Retrieval: 27th European Conference on IR Research, ECIR 2005, Santiago de Compostela, Spain, March 21-23, 2005. Proceedings 27*. Springer, 2005, pp. 345–359.