Exploring Machine Learning Techniques for Predicting Vulnerability from Source Code

Sahrima Jannat Oishwee
Dept of Computer Science
University of Saskatchewan
Saskatoon, Canada
sao107@usask.ca

***Abstract*—Vulnerable source code is a potential threat to software security. If it is possible to do the early prediction of vulnerable and non-vulnerable source code, it will be easier to take early precautions. Early precautions will lead to a more sure software system. To address this issue, we can use machine learning techniques to build the vulnerability prediction model. The goal of this study is to explore different machine learning techniques to predict vulnerable and non-vulnerable functions from source code. A publicly available dataset was used for this study where each row is consists of source code functions along with five different vulnerabilities. The functions of source code are the features and vulnerabilities are five different classes having "True" or "False" value for each source code function. One source code function may belong to multiple vulnerabilities or none of them. It was an imbalanced dataset and I perform under-sampling to make it a balanced one. The experiment was done with three different machine learning techniques for this study. The techniques are Gaussian Naive Bayes, Multinomial Naive Bayes, and Support Vector Machine (SVM) with Radial Basis Function (RBF) kernel. Next, I compare the performance of these three techniques. The performance was evaluated by two performance measurement metrics: accuracy and F1 score along with the precision and recall of these techniques. The results show that Multinomial Naive Bayes performs better than the Gaussian Naive Bayes. Finally, SVM performs the best compared to the other two.**

***Index Terms*—Vulnerable, Non-vulnerable**

I. Introduction

Every year, an increasing number of software vulnerabilities have been discovered. Code vulnerability is such a flaw that creates a potential risk to software security. It is particularly important to predict the software vulnerabilities at the early stage of the system as these vulnerabilities have widespread impact. The vulnerabilities also cause substantial economic and reputational damage to both companies and people, developing or using the software. Early prediction of the vulnerabilities from source code will help the software companies to take the early steps to resolve those vulnerabilities from their system. Taking necessary steps will be even easier if they have the information about the vulnerabilities like type, impact, severity, stakeholders, etc.

In this study, we are talking about Vulnerability and Common Weakness Enumeration(CWE). Vulnerability refers to weakness, flaw, or error found within a system that has the potential to be leveraged by a threat agent to compromise the security of that system. For example, Broken Authentication, SQL Injection, Cross-Site Scripting, Cross-Site Request Forgery, Security Misconfiguration, Denial of services, etc. These vulnerabilities can be created from the source code of a system. This project is focused on the vulnerabilities listed on CWE. The CWE[1] is a category system that categorized software weaknesses and vulnerabilities based on software and hardware weaknesses. It provides unique identifiers for each of the vulnerabilities. Additionally, it provides a description, reported date, venerability type, complexity, stakeholders, the impact of each vulnerability.

As the Common Weakness Enumeration (CWE) details provide the information about the vulnerabilities (type, impact, severity, stakeholders), in this paper, I provide a prediction model to find the vulnerabilities according to the CWE details, the software companies can take the necessary steps to resolve and prioritize. The goal of this paper is to build the prediction model for detecting vulnerability with different machine learning techniques and evaluate the performance of those techniques.

To address the goal of this study, I trained the prediction models with the dataset using three approaches: Gaussian Naive Bayes (GaussianNB), Multinomial Naive Bayes (MultinomialNB), and Support Vector Machine (SVM) to predict the possibility of having vulnerability in a code function. I used a sample dataset where different functions of the source codes are the features and five different vulnerabilities (CWE-119, CWE-120, CWE-469, CWE-476, and CWE-Others) are the five different classes. Each function may fall into multiple classes or none of them. I perform above mentioned three machine learnings separately for each of the classes. Performances of each of these classifiers are evaluated by using two metrics: accuracy, F1 score along with precision, and recall.

Multinomial Naive Bayes gives a higher accuracy rate than Gaussian Naive Bayes for all the classes except the class CWE-120. Additionally, the recall is higher than the other two classifiers for all the classes. The SVM performs better than the Gaussian Naive Bayes and Multinomial Naive Bayes with the higher accuracy and F1 score.

II. Literature Review

Currently, some tools provide metrics-based analysis in terms of the number of vulnerabilities found in a system.

[1]https://cwe.mitre.org/data/index.html

Beyond these tools, there are some Machine Learning (ML) and Deep Learning (DL) methods to detect the vulnerability from source code. A recent study, Bilgin et al. [1] investigated whether ML can distinguish vulnerable and non-vulnerable codes from the source code. They used Abstract Syntax Tree (AST) and Convolutional Neural Networks (CNN) on publicly available datasets. They have mentioned that using AST they get superior performance. Because AST provides categorical encoding of AST tokens and their consistent mapping to final vector correspondence. It reveals the hidden patterns indicating targeted vulnerabilities to the machine learning technique. Therefore, the machine learning technique performs better. Alves et al. has performed an experimental study on different machine learning techniques to predict vulnerabilities [2]. In this study, they have used a variety of datasets that have been used in different research papers. The authors have used the Bayesian network, Decision tree, Logistic regression, and Random forest for predicting vulnerabilities and compared the accuracy of those models, and Random forest performed best among all. In another study, [3]the authors have predicted the vulnerabilities from different systems: Apache Tomcat, CXF, Django, Keystone with software metrics. They have collected data labeled as vulnerable or non-vulnerable and then trained the machine. They have used Support Vector Machine (SVM) and Logistic Regression (LR) to predict vulnerable components based on the values of the features.

There is another study [4], adopting Deep representation learning approaches to detect vulnerability belongs to Russell et al. They have collected their dataset from C and C++ open-source code. Using this dataset, they have developed a vulnerability detection tool based on deep feature representation learning which directly interprets lexed source code. They have passed the source code into a lexer then performed embedding to the lexer. After embedding they created a convolutional filter to create learn source feature. Lastly, they have performed a Random forest classifier, CNN, and Recurrent neural network. They have compared the accuracy for those models and concluded that CNN performs best among the other techniques. In this study, I use the dataset from the work of Russell et al. [4] and I use the machine learning techniques rather than the deep learning techniques used in their work.

III. Algorithms Description

To build the prediction model, I am planning to use two types of Naive Bayes (NB) classification because it is one of the most used classifiers in Natural Language Processing (NLP) problems. It predicts the tag of a text and calculates the probability of each tag and finally gives output the tag with the highest one. So, there is a good chance of having high accuracy. Additionally, I am going to use Support Vector Machine (SVM) as it is an optimal solution for NLP [5]. These two classifiers have different attributes which make them efficient in different categories of the prediction model. Lastly, I will assess the performance of these two techniques. The study design is presented in Fig: 1. To understand the ML techniques, I followed Scikit-learn documentation[2] and MonkeyLearn[3].

A. Naive Bayes

Nave Bayes is a classification technique based on Bayes Theorem. It assumes that features are independent from each other. It is one of the simplest but strongest algorithms used for classification. The principal of Naive Bayes theorem is that each feature makes an independent and equal contribution. Bayes theorem provides a way of calculating posterior probability $P(Y|X)$ from $P(Y)$, $P(x)$, and $P(X|Y)$:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \tag{1}$$

In the equation above, $P(Y|X)$ is the posterior probability of class (Y, target) given features (X, features). $P(Y)$ is the prior probability of class. $P(X|Y)$ is the likelihood which is the probability of features given class and $P(x)$ is the prior probability of features. There is very little explicit training in Naive Bayes compared to other common classification methods. The only work that must be done before prediction is finding the parameters for the features individual probability distributions. If there is n number of features in $X = X_1, X_2, X_3, ..., X_n$ and they are conditionally independent, then we can write:

$$P(X|Y) = \prod_{i=1}^{n} P(Xi|Y) = P(X_1|Y) * P(X_2|Y) * .....* P(X_n|Y) \tag{2}$$

Finally we can re-write the equ. 1 for n number of features and get the Naive Bayes classifier equation:

$$P(Y|X) = \frac{P(Y)\prod_{i=1}^{n} P(Xi|Y)}{P(X_1, X_2, X_3, ..., X_n)} \tag{3}$$

Now, I have a way to estimate the probability of a given data point falling in a certain class, I need to be able to use this probability to produce classifications. Naive Bayes handles this in a very simple manner; simply pick the Y that has the largest probability given the data points features. For every value of Y, the denominator $P(X_1, X_2, X_3, ..., X_n)$ is same. So we can formulate the question:

$$P(Y|X) \propto P(Y)\prod_{i=1}^{n} P(Xi|Y) \tag{4}$$

[2]https://scikit-learn.org/

[3]https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/

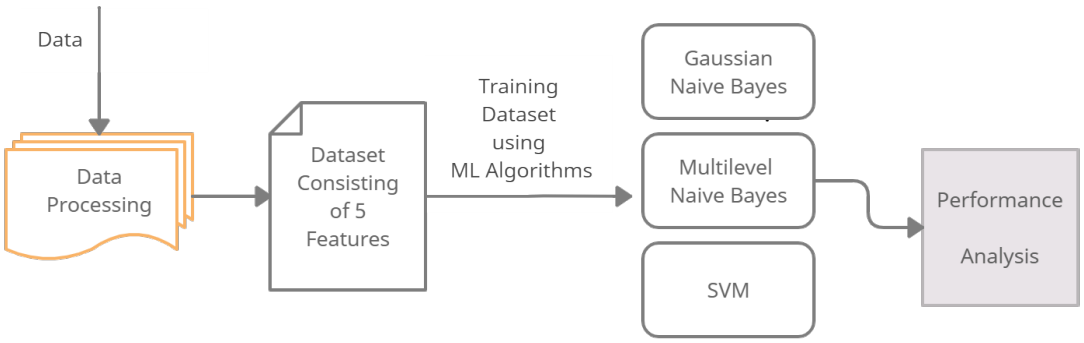

Fig. 1. Study Design

$$\hat{Y} = \arg\max_{Y} P(Y) \prod_{i=1}^{n} P(X_i \mid Y) \tag{5}$$

In Scikit learn library, there are three variations of Naive Bayes model: Gaussian, Multinomial and Bernoulli. For this dataset, I have used Gaussian NB model and Multinomial NB model and compare the performances.

In Gaussian Naive Bayes, it find the posterior probability with the mean μ and variance σ. The equation is below:

$$P(X_i|Y) = \frac{1}{\sqrt{2 * \pi * \mu}} * \exp(\frac{-(X_i - \mu)^2}{2 * \sigma^2}) \tag{6}$$

For the text data the Multinomail Naive Bayes use the probability of class $P(Y)$ and the likelihood of a word (X) fall into that class $P(X_i|Y_i)$. The equations of these two:

$$P(Y_i) = \frac{Total\ number\ of\ document\ in\ that\ class\ Y_i}{Total\ number\ of\ document} \tag{7}$$

$$P(X_i|Y_i) = \frac{\#\ of\ X_i\ in\ the\ class + 1}{\#\ of\ word\ X_i\ in\ all\ the\ documents + |V|} \tag{8}$$

Here $|V|$ is the total number of distinct words in the entire dataset.

B. Support Vector Machine

A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labeled training data for each category, they're able to categorize new text. The SVM use vector metrics to perform the classification. So, at first all the texts transform into vectors which is a list of numbers. To understand how SVM works, we provide an example with Fig: 2 [4], Fig: 3[5].

Linear Data: In the Fig: 2, We have two tags: white circle and black circle. We want to have a classifier that calculate that the input is either black or white. It takes the decision to draw a decision boundary among these data points which is the best hyperplane to separate them. We see three lines: H1, H2, H3. H1 can not separate the black and white tags but H2 and H3 both can separate those. Now, the question come, SVM will chose which of them. SVM will take H3 as it has large margin of maximum distance from the black and white data.

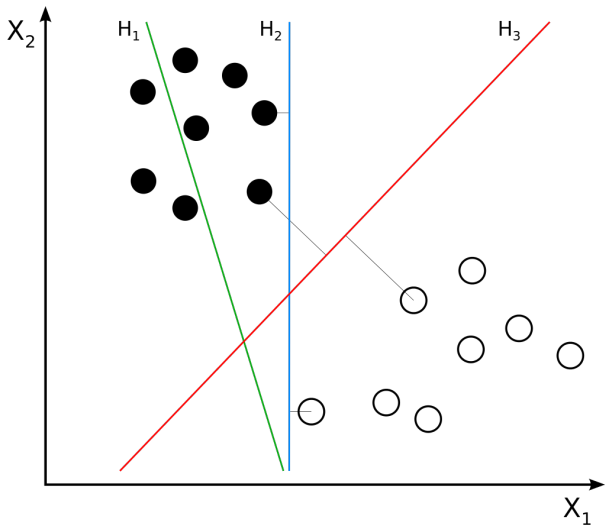

Fig. 2. Support Vector Machine (SVM) for linear data

[4]https://en.wikipedia.org/wiki/Support-vector$_machine/media/File$: $Svm_separating_hyperplanes_(SVG).svg$

[5]https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/

Non-linear Data: In the Fig: 3, We have two tags: blue and red. But, the data is non-linear and complex to draw a line to separate them. It takes the decision to draw a decision boundary among these data points which is the best hyperplane to separate them. So, SVM put these data to higher dimension and find the best separable boudary for those.

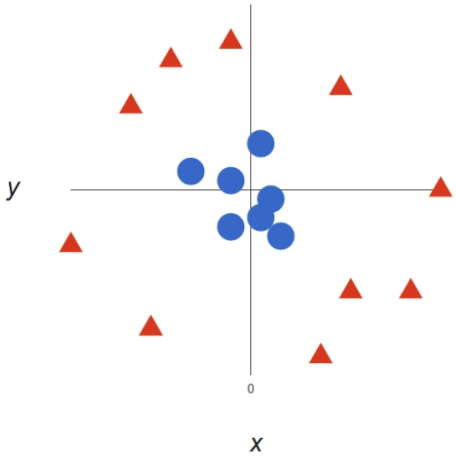

Fig. 3. Support Vector Machine (SVM) for non-linear data

To classify the nonlinear data SVM maps the space to a higher dimension. But, there can be a lot of new dimensions and each one of them possibly need a complicated calculation. So, we use different kernels as the solution.

Karnel: A set of mathematical functions are used in SVM algorithms which are called 'kernel'. The kernel takes data as input and processes it for the required formation. Various types of kernels are used by the SVM algorithms. Most of the real-world datasets are inseparable. To get more efficient results from the datasets we need to make the data separable and that is why more features are needed to be added in higher dimensions. So kernel in this regard helps to avoid the high dimensional problem but gets the decision without complex calculation and does this with a simple function. So, kernel functions aim to take the inseparable higher dimension data and convert them to linearly separable ones. In this study I am going to use Radial Basis Function (RBF) kernel.

Radial Basis Function kernel is mostly used when there is no prior knowledge about the data and it is the most popular in SVM classification. It measures the similarity between two vectors. The RBF equation is:

$$K(x, x') = exp(-\frac{||x - x'||}{\sigma^2})$$

Here $||x - x'||$ may be recognized as the squared Euclidean distance between two feature vectors. SVM with RBF is sensitive to two parameters: Gamma and C.

Gamma: The gamma parameter defines how far the influence of a single training example reaches. If the gamma value is low, it means the influence is far and if it is low, the influence is close. In other words, with low gamma, points far away from a reasonable separation line are considered in the calculation for the separation line. Whereas high gamma means the points close to the separation line are considered in the calculation.

C: C refers to the error margin acceptance of the model also called as a regularization parameter. If the C value is large, a smaller margin will be accepted if the decision function is better at classifying all training points correctly. If the C value is low, it encourages a larger margin, therefore a simpler decision function will be created at the cost of training accuracy.

In this study I am using two metrices : accuracy and F1 score to evaluate the performances.

IV. Data Description

I am going to use the dataset from the work that belongs to Russell et al. [4]. This dataset is available at https://osf.io/d45bw/. It is divided into three sets: train, test, and validate. It consists of the source code of 1.27 million functions written in C and C++. Those were mined from open-source software. The dataset is labeled by static analysis for potential vulnerabilities according to the CWE list. The dataset is considering five classifications of vulnerabilities such as: 'CWE-119', 'CWE-120', 'CWE-469', 'CWE-476', 'CWE-other'. CWE-119 addresses the vulnerability type called improper restriction of operations within the bounds of a memory buffer. The overflow, pointer subtraction to determine the size, and null pointer vulnerabilities are addressed by CWE-120, CWE-469, and CWE- 476, respectively. The CWE-other consists of CWE-20, CWE-457, and CWE-805 which refer to improper input validation, use of the uninitialized variable, buffer access with incorrect length value, etc. All these classifiers have a 'true' or 'false' value for each function of the source code. 'true' value refers to that the function is vulnerable and 'false' value refers to that the function is non-vulnerable. The shape of train data, test data and validate data is (1019471, 6), (127419, 6), and (127419, 6) respectively. It refers that the train data has 1019471 rows and both the test and validate data has 127419 and both have six columns where one column is the functions written in C and C++ and the other five columns are the type of different vulnerabilities.

To understand the data distribution, I calculated the number of vulnerable and non-vulnerable functions. Table: I presents the percentages of vulnerable and non-vulnerable functions in the five classes. We can see that in the three datasets the percentage of vulnerable functions is very low. For CWE-469 the percentage of vulnerable functions is less than one for the train (0.7), test (0.63) and validate (0.56). To visualize the data, I have randomly plotted 1000 data from the training data, so understand that, if I randomly pick a sample size how the data will look. The visualization is presented in Figure: 4. We can see that the non-vulnerable value is very high in comparison to the vulnerable functions. Therefore, I understand that the data is very imbalanced.

TABLE I
THE PERCENTAGE OF VULNERABLE AND NON-VULNERABLE SAMPLE IN IMBALANCE DATASET

Class	Training Data		Validation Data		Test Data	
	Vulnerable (%)	Non Vulnerable (%)	Vulnerable (%)	Non Vulnerable (%)	Vulnerable (%)	Non Vulnerable (%)
CWE-119	5.82	94.18	5.84	94.16	6.16	93.84
CWE-120	11.10	88.89	11.18	88.82	11.86	88.14
CEW-469	0.70	99.30	0.63	99.37	0.56	99.44
CWE-476	2.52	97.48	2.25	97.45	2.43	97.57
CWE-Other	7.15	92.85	7.30	91.70	6.92	93.08

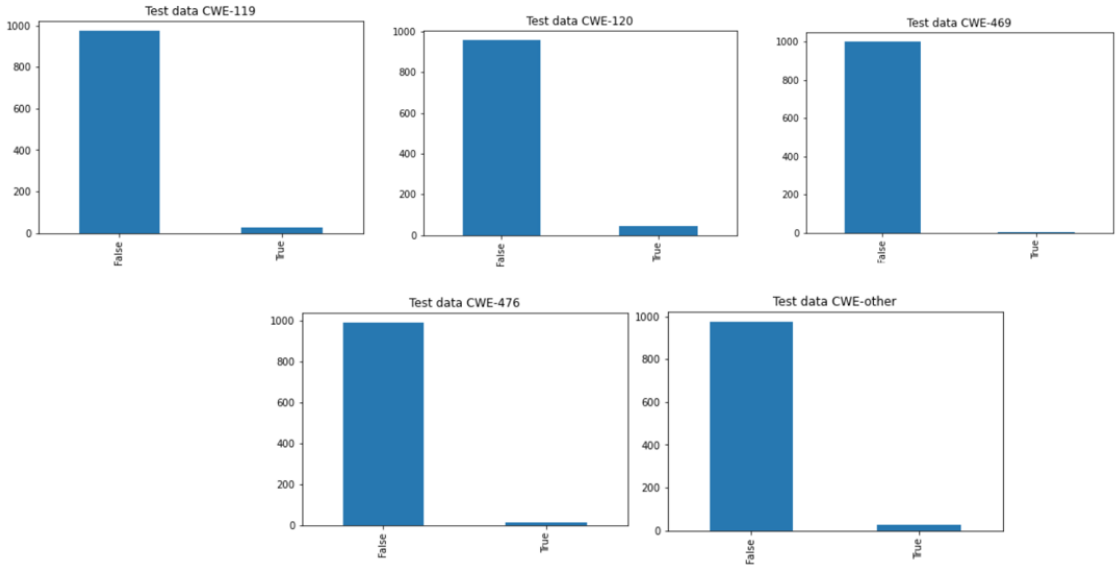

Fig. 4. Imbalanced Data

V. RESULTS

A. Data Pre-processing

The dataset is available in an hdf5 file and I have converted it to a pickle file for the ease of further analyzing. I performed the study on a sample dataset of 1000 data from the training dataset, test dataset, and validate dataset. As I have already discussed that the dataset is imbalanced, I have processed the three datasets to convert them into a balanced sample dataset. To make this sample dataset balanced, I performed under-sampling. I took randomly 500 vulnerable functions and 500 non-vulnerable functions for each of the five classes. To visualize my processed data set I plotted the train data and it is presented in Fig: 5. From Fig: 5, we can see every class has 500 functions for both vulnerable and non-vulnerable except, CWE-469. As I have already mentioned that for CWE-469 the vulnerable function is not even one percent. So, it was unable to have a total of 500 vulnerable functions in the sample dataset. Though CWE-469 has a very less vulnerable function, still the dataset is better than the imbalanced one.

I have tokenized the C and C++ functions into the numerical value. The token size is 11,663, 11,328, 11,328, 11108, and 11,237 for CWE-119, CWE-120, CWE-469, CWE-476, and CWE-Other. So, I have considered the average of these values which is 11,333 as the token size. For example: 'if' refers as 3126441,'0' refers as 2106459, and 'return' refers as 1745333.The total number of tokens is 1094129.

B. Using ML Techniques

After processing the data I have performed Gaussian Naive Bayes, Multinomial Naive Bayes, and RBF kernel SVM. The results are summarized in Table: II. Here it is visible that the Multinomial Naive Bayes has better accuracy and F1 score than Gaussian Naive Bayes. Only for the class CWE-120, Multinomial Naive Bayes has lower accuracy (58.40%) and F1 score (0.601) than the accuracy (74.10 %) and F1 score (0.623) of Gaussian Naive Bayes. Rather than this, we can say Multinomial Naive Bayes performed better than Gaussian Naive Bayes. This result is quite expected. In Multinomial Naive Bayes, the value is the count of each occurrence but Gaussian Naive Bayes is slightly more limited by its mean and variance. As a result, Multinomial Naive Bayes better fitted the data and performed better.

For SVM with RBF kernel, I wanted to use a higher C value to ensure lower training error. So, I used the C value as ten. For the value gamma, I used an intermediate value that is 0.5. With this value, the model is not too constrained and captures the complexity. Additionally, regularization with

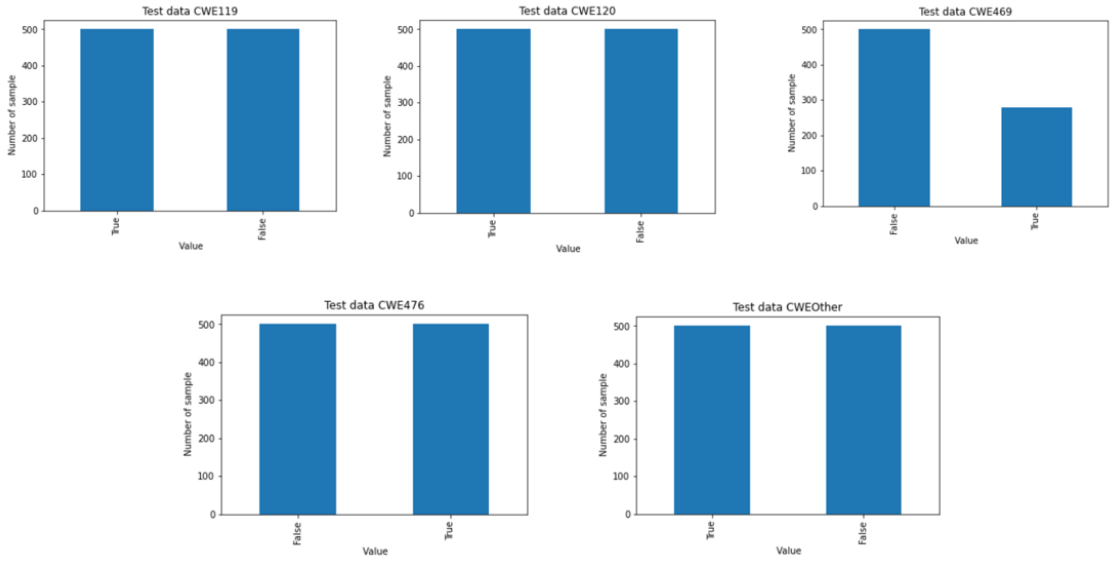

Fig. 5. Balanced data after processing

C will be able to prevent the over-fitting problem. With this configuration, I run the experiment with RBF kernel SVM.

Among SVM and the two Naive Bayes techniques, in terms of accuracy, SVM has a higher score in CWE-119, CWE-120, CWE-476, and CWE-Other. Only for CWE-476, SVM has 64.26 % of accuracy and Multinomial NB has 65.42 % of accuracy. The SVM has a slightly better F1 score than the Multinomial NB for all the vulnerabilities. So, SVM with RBF kernel is the best fit in this experiment. The Naïve Bayes techniques are probabilistic and SVM is geometric in nature. The biggest difference between the models is that Naive Bayes treats the features as independent, whereas SVM (RBF kernel) looks at the interactions between them to a certain degree. In the dataset, each variable and syntax of the functions can depend on other variables of the function. So that, SVM (RBF kernel) captures those better than Multinomial and Gaussian Naive Bayes. Hence, the SVM performs better at this classification task for this data.

VI. Discussion

I have worked with a small sample dataset to understand how this dataset and the model behave on the dataset. In the entire dataset, the number of vulnerable source codes is very low. To, handle this issue, I have under-sampled the dataset. Randomly selected 500 vulnerable and 500 non-vulnerable function sources. So, 50% data is vulnerable and 50% data is not vulnerable. In the dataset, for each vulnerability, there is only two output: vulnerable or non-vulnerable. Therefore, with 50% vulnerable and 50% not vulnerable, the model can easily achieve 50% of accuracy. In this experiment for each class with different ML techniques, I was able to achieve accuracy which is close to 50 to 60%. In my opinion, this performance is quite expected due to the 50% vulnerable and 50% non-vulnerable data.

TABLE II
Performance Metrices of GassianNB, MultinomialNB, SVM

Class	Performance Metrics	Gaussian Naive Bayes	Multinomial Naive Bayes	SVM
CWE-119	Recall	70.00	72.00	50.00
	Precision	13.40	35.00	100.00
	Accuracy(%)	54.20	60.00	62.00
	F1 score	0.633	0.630	0.667
CWE-120	Recall	43.00	53.38	52.50
	Precision	17.55	37.55	100.00
	Accuracy(%)	74.10	58.40	62.50
	F1 score	0.623	0.601	0.680
CWE-469	Recall	33.23	61.53	63.00
	Precision	84.89	8.632	94.00
	Accuracy(%)	33.80	65.42	64.26
	F1 score	0.444	0.753	0.754
CWE-476	Recall	49.64	65.62	51.00
	Precision	98.00	33.62	96.00
	Accuracy(%)	49.32	57.99	63.00
	F1 score	0.651	0.606	0.667
CWE-Other	Recall	60.93	63.97	50.00
	Precision	15.63	38.00	100.00
	Accuracy(%)	52.80	58.32	59.48
	F1 score	0.603	0.601	0.660

Next, in this experiment, my concern was to find both the vulnerable and non-vulnerable source code to separate them from each other. For that the entire accuracy and F1 score were important. SVM provides better accuracy and F1 score, so SVM is the best fit for me. But, if anyone wants to build a model to predict only the vulnerable source codes, then he/she needs to focus on the recall value to choose the best model. Higher Recall refers to a higher prediction of vulnerabilities. There may be some non-vulnerable source codes classified as vulnerable but there is a higher prediction of real vulnerable source codes (True Positive). In this case, predicting the more number of true vulnerable source code (True Positive) is

important. As a result, the prediction of some non-vulnerable source code (False-Positive) can be overlooked. Therefore the MultinomialNB will be the most suitable classifier for this vulnerability prediction model as it has the best recall value of all the classes.

Lastly, I compare the result of this study with the prior study. I am using the same data used in the work of Russel et al. [4]. In that work, they have used RNN and CNN to predict the vulnerabilities from the source code. They have concluded that CNN performs best for the dataset. They achieved the highest recall value of 94.4, precision value of 95.4, and F1 score of 0.840. SVM gave the best performance in my study. So, I compare the performance of CNN and SVM. In this experiment, the models archived relatively low recall and F1 values than those. As I worked with sample data and the prior work was done with the full dataset, it is hard to compare these results explicitly. The dataset is consists of a huge number of text data which I have tokenized. For SVM, this complex dataset requires a higher dimension to separate them perfectly, the number of dimensions can be n for this dataset. Additionally, we are fixing the parameters for the SVM model and it uses vector representations where every component usually makes some sense on its own. This complex dataset may have more information than a bunch of its properties that feed into SVM. On the other hand, CNN builds its own features from the raw information. So this can be the potential reason CNN performs better than the experimented Machine learning techniques for this dataset.

VII. Future Work

In this study, we have successfully performed three Machine Learning techniques to find the vulnerable and non-vulnerable functions from the source code analysis. Though the performance of the techniques was not that satisfactory, SVM performed better than Gaussian Naive Bayes and Multinomial Naive Bayes in terms of accuracy and F1 score. On the other hand, Multinomial Naive Bayes has the best recall values among these. If anyone only wants to detect the true positive (vulnerable functions) values at a higher rate, they can go for Multinomial Naive Bayes among this three. For SVM, I have taken a high C value to avoid training error rate and taken an intermediate gamma value to have a balance of the radius influence.

Bilgin et al. [1] has used AST to build a model for vulnerable source code analysis. They have achieved high performance from that. In our experiment, I did not perform this. Still, I do not know is it worthy to use AST for this dataset. So, In future this solution can be also explored for better performances. I have worked with a small sample dataset where 50% data is vulnerable and 50% data is not vulnerable. To make a balanced dataset, I performed under-sampling. With under-sampling and 50% data is vulnerable and 50% data the Machine Learning techniques achieve accuracy close to 50 to 60 %. In my opinion, this performance is not very great due to the 50% vulnerable and 50% not vulnerable data. I did not perform over-sampling and did not experiments with a large amount of data. There may be more information that may increase the performance if I feed them to the Machine learning model. To have a look into this assumption, this experiments and the analysis can be re-performed after over-sampling.

References

[1] Z. Bilgin, M. A. Ersoy, E. U. Soykan, E. Tomur, P. Çomak, and L. Karaçay, "Vulnerability prediction from source code using machine learning," *IEEE Access*, vol. 8, pp. 150 672–150 684, 2020.

[2] H. Alves, B. Fonseca, and N. Antunes, "Experimenting machine learning techniques to predict vulnerabilities," in *2016 Seventh Latin-American Symposium on Dependable Computing (LADC)*. IEEE, 2016, pp. 151–156.

[3] T.-Y. Chong, V. Anu, and K. Z. Sultana, "Using software metrics for predicting vulnerable code-components: a study on java and python open source projects," in *2019 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*. IEEE, 2019, pp. 98–103.

[4] R. Russell, L. Kim, L. Hamilton, T. Lazovich, J. Harer, O. Ozdemir, P. Ellingwood, and M. McConley, "Automated vulnerability detection in source code using deep representation learning," in *2018 17th IEEE international conference on machine learning and applications (ICMLA)*. IEEE, 2018, pp. 757–762.

[5] Y. Li, K. Bontcheva, and H. Cunningham, "Adapting svm for natural language learning: A case study involving information extraction," *Natural Language Engineering*, vol. 15, no. 2, pp. 241–271, 2009.