International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 04 | April - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

Malware Detection Using Machine Learning

Siddharth
Chandigarh University
Chandigarh, India
i.sid2302@gmail.com

Dr. Bharti Sahu
Chandigarh University
Chandigarh, India
Bhartisahu8001@gmail.com

***Abstract*— In this work, we present a flexible system that makes use of various machine learning methods to efficiently distinguish between malware and clean files while purposefully reducing false positives. In the field of cybersecurity, our strong framework is both flexible and strong, working along with different machine learning algorithms. Our study unfolds with an exploration of basic principles using the Random Model, K Nearest Neighbouring Classifier (KNN), and Logistic Regression as foundational parts, emphasizing the differentiation between malware and benign files. Extensive experiments on mediumsized datasets that include malware and clean files verify the effectiveness of our methodology. The system then goes through a painstaking scaling-up process that guarantees smooth operation with big datasets containing both malware and clean files. Our methodology is validated by analysing three important algorithms: Random Model, KNN, and Logistic Regression, each of which adds unique advantages to the malware detection system. The evaluation, which is carried out on several datasets, aims to minimize false positives while striking a compromise between precision and recall. Finally, our flexible system, implemented and evaluated on many datasets, demonstrates its efficacy in distinguishing malware from clean files. The framework's flexibility and scalability make it an invaluable tool in the everevolving field of cybersecurity, providing a sophisticated method of malware detection. The proposed algorithms emphasize the framework's potential as a supplementary tool to current cybersecurity measures while also adding to its reliability.**

Keywords— ML, KNN, RM, LR

I. INTRODUCTION

Malware is defined as software designed to infiltrate or damage a computer system without the owner's informed consent. Malware is a broad category that includes a wide range of malicious programs and applications, from standalone viruses to file infectors. Among these, the rogues' gallery consists of characters with distinct digital weaponry, such as Ramnit, Lollipop, Vundo, Simda, Tracur, Kelihos_ver1, Obfuscator, Kelihos_ver3, Gatak, and ACY.
The malicious forces behind malware have also changed and mutated as our digital environment continuous its unstoppable progress. They have added many polymorphic layers to evade the conventional, signature-based techniques used by antivirus solutions.

Security measures face a great challenge from the modern malware landscape, which updates itself frequently to surpass antivirus software that uses static signatures in its detection [1]. The conflict between cyber attackers and defenders takes place in a dynamic environment with constantly changing rules of engagement. One can explore the field of dynamical file analysis to give a visual story of the ever-changing challenges presented by malware and the related advances in detection techniques. In this case, the use of virtual environment emulation acts as a stage for the elaborate dance that is performed between detection technology and malware [2]. Furthermore, a thorough comprehension of the terrain necessitates an investigation of conventional methods intended to detect metamorphic viruses. These methods provide as a basis of knowledge, illuminating the subtleties of identifying malicious code that modifies its form to avoid detection through traditional means [3]. In reaction to the growing complexity of malware, researchers are focusing on the potential of machine learning as a ray of hope in the fight against this constantly changing and dynamic threat environment. The literature study that follows provides a broad overview of various machine learning approaches that are ready to serve as sentinels in the search for effective malware detection tools. Of these, boosted decision trees' ability to use n-gram data makes them a strong competitor, outperforming more conventional classifiers like Support Vector Machines and the Naive Bayes classifier in terms of performance [5]. The extraction of association rules from Windows API execution sequences adds even more depth to the toolkit of malware detection techniques and demonstrates the adaptability of machine learning. By using Hidden Markov Models (HMMs), one can apply a probabilistic method to determine if a given program file is a variant of a known object. In a related effort, Profile Hidden Markov Models—which are well-known for their efficacy in the field of bioinformatics—are adapted to accomplish a comparable objective in the field of malware detection [8][9]. As a result, the literature presents a varied tapestry of machine learning approaches, each adding a special thread to the complex web of malware detection. As Section V develops, the story reaches a climax and reveals the outcome of this complex trip. Here, 52 key characteristics taken from the .asm files are prepared to serve as the cornerstone of a largescale system, capable of identifying malware in massive training datasets. The convergence of preprocessing and analysis signals that the framework is ready to be scaled up to take on the formidable challenge of very large training datasets.

Essentially, this study sets out on an extensive investigation of the complex field of malware detection—a voyage that starts with a threat taxonomy, traverses the shortcomings of traditional approaches, and reveals the exciting opportunities presented by a wide range of machine learning strategies. The prologue of the introduction sets the stage for the story that follows as it attempts

to create a flexible and robust framework to combat malware's constant growth in the digital sphere. With every step we take across this terrain, a new chapter in the history of cybersecurity is revealed.

II. LITERATURE REVIEW

The literature review provided in your research paper titled "Malware Detection Using Machine Learning" offers an indepth exploration of the existing research and techniques in the field of malware detection. Malware, a ubiquitous threat to computer systems, is a broad category of harmful software intended to compromise or corrupt systems without authorization from the user. In addressing the ever-evolving issues of malware detection, this study highlights the shortcomings of conventional signature-based approaches because of the malware's ever-increasing sophistication, which includes behaviours that are polymorphic and self-updating.

The literature study explores a range of machine learning methods used to identify malware, illustrating the trend toward data-driven strategies. An example of the potential of ensemble methods in classification applications is the higher performance of boosted decision trees using n-gram data over Naive Bayes and Support Vector Machines. Other methods investigate the use of Hidden Markov Models and Profile Hidden Markov Models, along with association criteria obtained from Windows API execution sequences, to identify malware variations. The paper also emphasizes how neural networks and Self-Organizing Maps can be used to detect polymorphic malware and detect patterns in the behaviour of Windows executable files that indicate the presence of viruses. When taken as a whole, these findings highlight the necessity for more advanced and complex methods to combat the complex nature of malware. The literature study also discusses the difficulties in handling big datasets of ".asm" and ".bytes" files, as well as data pretreatment and feature extraction. Although the sheer volume of these files presents major complications, they do offer a low-level insight on software activity. The effectiveness of machine learning models, such as K-Nearest Neighbours (KNN), is improved by reducing dimensionality and noise through the use of feature selection procedures, such as correlation-based filtering and wrapper techniques.

In addition, logistic regression is presented as a binary classification technique that models the probability of malware existence by utilizing the logistic (Sigmoid) function. Based on pertinent qualities and characteristics, this approach has the potential to simulate the likelihood of a system becoming infected with malware. The experimental results, which show that the Random Model, KNN, and Logistic Regression are the most dependable of the studied algorithms for malware detection, are presented at the end of the literature review. While the approaches do not completely reach the zero false positive target, they do

show a significant boost in the overall detection rate, suggesting that they could be a useful addition to existing antivirus programs.

In summary, the literature study offers insightful information on the current state of machine learning-based malware detection. It emphasizes how crucial data-driven approaches are becoming to combating the dynamic nature of malware threats and the difficulties posed by large-scale data analysis.

III. DATASETS

We used three datasets: a *training* dataset, a *test* dataset, and a *"scale-up"* dataset up to 200GB. The number of malware files and respectively clean files in these datasets is shown in the first two columns of Table I. As stated above, our main goal is to achieve malware detection with only a few (if possible 0) false positives, therefore the clean files in this dataset (and in the scale-up dataset) are much larger than the number of malware files.

From the whole feature set that we created for malware detection, 308 binary features were selected for the experiments to be presented in this paper. Files that generate similar values for the chosen feature set were counted only once. The last two columns in Table I show the total number of unique combinations of the 308 selected binary features in the training, test, and respectively scale-up datasets. Note that the number of clean combinations — i.e. combinations of feature values for the clean files — in the three datasets is much smaller than the number of malware unique combinations.

TABLE I
NUMBER OF FILES AND UNIQUE COMBINATIONS OF FEATURE VALUES IN THE TRAINING, TEST, AND SCALE-UP DATASETS UPTO 200GB.

	Files		Unique combinations	
Database	malware	clean	malware	clean
Training	6955	695535	6922	315
Test	21740	6521	609	220
Scale-up	approx. 2M	approx. 80M	8817	12230

TABLE II
MALWARE DISTRIBUTION IN THE TRAINING AND TEST DATASETS.

	Training Dataset		Test Dataset
Malware Type	Files	Unique combinations of feature values	Files
Ramnit	14.2%	5.19%	13.84%
Lollipop	22.8%	30.73% 40.15%	22.95%
Kelihos-ver3	27.1%	12.15%	27.50%
Vundo	4.4%	0.11%	4.50%
Simda	0.4%	2.66% 3.17%	0.09%
Tracur Kelihos-ver1	6.9% 3.7%	4.66% 6.10%	6.89% 4.41%
Obfuscator.ACY	11.3%		11.49%
Gatak	9.3%		9.24%

since the majority of features were developed to highlight a certain component of malware files (either a geometrical form or behaviour aspect).

The majority of the clean files in the training database are system files (from various operating system versions) and executable and library files from several well-known apps. In order to better train and test the system, we also employ clean files that are packed or that have the same form or geometrical characteristics with malware files (e.g., use the same packer). The training dataset includes malware files that were obtained from the Virus Heaven collection. The test dataset includes clean files from several operating systems and malware files from the Wild List collection (other files than those used in the first database). The training and test datasets malware collections include Ramnit, Lollipop, Kelihos_ver3, Vundo, Simda, Tracur, and Kelihos_ver1, Obfuscator.ACY, Gatak. The percentage of those malware kinds in the training and, respectively, test datasets is shown in the first and third columns of Table II. The second column in Table II shows the proportion of malware-specific unique combinations across all feature value combinations in the training dataset.

Divide the dataset into train, test, and cv parts, and then examine the distribution of class in each split to see if it is consistent across all splits. 6955 data points make up the train data. 2174 data points make up the test data. 1739 data points were used for cross validation.

IV. ALGORITHMS

The main goal of this section is to modify the Random Model so as to correctly detect malware files, while forcing detection

Algorithm 1: Random Model

test_data_len = X_test.get_row_count() cv_data_len = X_cv.get_row_count() cv_predicted_y = zeros(cv_data_len, 9) test_predicted_y = zeros(test_data_len, 9) for i in range(cv_data_len): rand_probs = random_values_between_0_and_1(9) cv_predicted_y[i] = normalize_to_sum_1(rand_probs) log_loss_cv = calculate_log_loss(y_cv, cv_predicted_y) for i in range(test_data_len):
rand_probs = random_values_between_0_and_1(9) test_predicted_y[i] = normalize_to_sum_1(rand_probs) log_loss_test = calculate_log_loss(y_test, test_predicted_y) predicted_y = find_argmax(test_predicted_y) plot_confusion_matrix(y_test, predicted_y + 1)

rate for one category [12]. In the sequel we will use the following data structures:

It is easy to create random probability for classification tasks using the "Random Model" method. For each class (in this case, there are nine classes) in this model, we generate random probability values so that the sum of these probabilities is 1. To establish a baseline model for comparison in machine learning tasks, this is done. The procedures to implement the Random Model for producing random probabilities and figuring out log loss for cross-validation and test datasets are described in the accompanying pseudocode.

We create a vector of probabilities in a random model, p = [p1, p2,..., pk], where k is the number of classes. Probabilities are added together, and the result is 1, hence pi = 1.

Without taking into account any attributes or patterns, a Random Model in the context of malware detection assigns random probabilities to various classes (malware or non-malware). Given that it relies on educated assumptions and is unable to accurately categorize malware, it is unsuitable for usage in realworld applications.

P(class = malware) = p, where 0 = p = 1 and P(class = nonmalware) = 1 - p are the definitions of the term. The Random

Model is not a useful method for detecting malware in computer systems. To evaluate the performance of more complex models, it serves as a baseline or reference model. Models should be educated on pertinent traits and behaviours of dangerous software in order to enable them to discriminate between malware and benign software in real-world malware detection.

Algorithm 2, henceforth called *KNN*. It performs the training for one chosen label (in our case either malware or clean), so

Algorithm 2: K Nearest Neighbour Classifier

import scikit-learn.neighbors knn_classifier = KNeighborsClassifier(n_neighbors=5, weights='uniform',

algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=1, **kwargs) knn_classifier.fit(X, y) knn_classifier.predict(X) knn_classifier.predict_proba(X)

for one chosen label (in our case either malware or clean), so that in the end the files situated on one side of the learned linear separator have exactly that label (assuming that the two classes are separable). The files on the other side of the linear separator can have mixed labels.

Machine learning algorithms for classification and regression include K-Nearest Neighbours (KNN). It categorizes data points in a feature space according to how close they are to other data points. KNN is typically not a good option for computer system malware detection. It is unable to detect intricate patterns and malware-specific traits. To properly detect malware, more sophisticated techniques that take a wider variety of traits and behaviours into account are needed.

Mathematical Representation of KNN: KNN determines the distance to its k nearest neighbours for a given data item X and classifies it into the category that is most prevalent among its neighbours. Distance metrics, such Euclidean distance, are used as the foundation for the mathematical representation. The K-Nearest Neighbours (KNN) method for malware detection entails classifying a particular data point as either malware or non-malware based on the dominant class among its k nearest neighbours. This method relies solely on closeness to neighbours in the feature space, which is not effective for malware identification because malware has complex and distinctive properties.

Algorithm 3: Logistic Regression

import scikit-learn.linear_model sgd_classifier = SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None, shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, class_weight=None, warm_start=False, average=False, n_iter=None) sgd_classifier.fit(X, y, coef_init=None, intercept_init=None, sample_weight=None) sgd_classifier.predict(X)

Binary categorization is handled by the machine learning algorithm logistic regression. A logistic (Sigmoid) function is used to model the likelihood that a data point will fall into a specific class. The Stochastic Gradient Descent (SGD) form of Logistic Regression in Scikit-Learn is described in the pseudocode. It offers techniques to fit the model and produce predictions, as well as initializing the classifier. A popular approach for binary classification problems, such as malware detection in computer systems, is logistic regression. It is useful for spotting dangerous software since it can simulate the likelihood that a system would become infected by malware.

The logistic (Sigmoid) function is used in logistic regression to determine the likelihood that a particular occurrence would fall into the positive class:

where,

L = the maximum value of the curve
e = the natural logarithm base (or Euler's number)
x_0 = the x-value of the sigmoid midpoint
k = steepness of the curve or the logistic growth rate

For detecting malware, logistic regression models the likelihood that a given system or file has malware. Based on the attributes

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

and qualities of the system or file, the algorithm determines this likelihood. The algorithm categorizes the instance as malware (1) or non-malware (0) by defining a threshold.

V. RESULTS

We performed cross-validation tests by running the three algorithm the Random Model, KNN and Logistic Regression presented in Section III on the training dataset described in Section II (6922 malware unique combinations, and 315 clean unique combinations).

For the Random Model, the following functions were used:

- The main function in the Random Model is in charge of producing random probabilities for each class. To generate random numbers, you can use Python functions like np.random.rand().

For the KNN classifier, the following function were used:

- Distance Metric Function: KNN relies on a distance metric function to measure the similarity between data points. Common distance functions include Euclidean distance, Manhattan distance, and Minkowski distance.
- Voting Function: KNN classifies data points according to the majority class among their k-nearest neighbours using a voting procedure.

For the Logistic Regression, the following function were used:

- The logistic function, often known as the sigmoid logistic function, is used in logistic regression to represent the likelihood that a given occurrence will fall into the positive class. The logistic function is defined as:

$$P(y = 1|X) = 1 + e - (X . w + b)$$

- Cost Function: The error between projected probabilities and actual labels is measured using the logistic loss, also known as the log loss, which is employed in logistic regression. During training, the logistic loss aids in adjusting the model's parameters.

For feature selection in conjunction with Random Model, KNN and Logistic Regression Model. The Random Model itself does not include feature selection because it assigns probability at random without taking features into account.

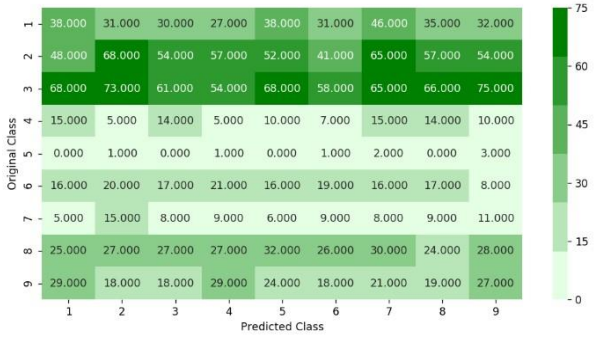


Fig 1.1 – Confusion Matrix of the Random Model.

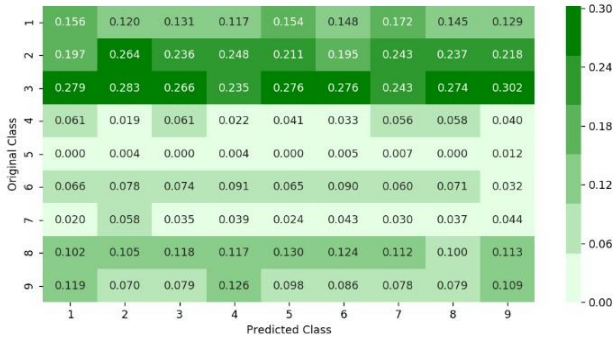


Fig 1.2 – Precision Matrix of the Random Model

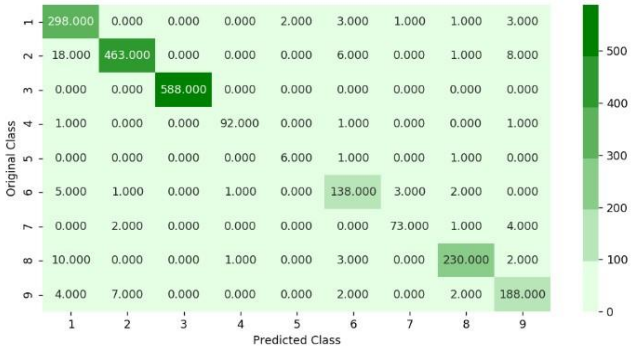
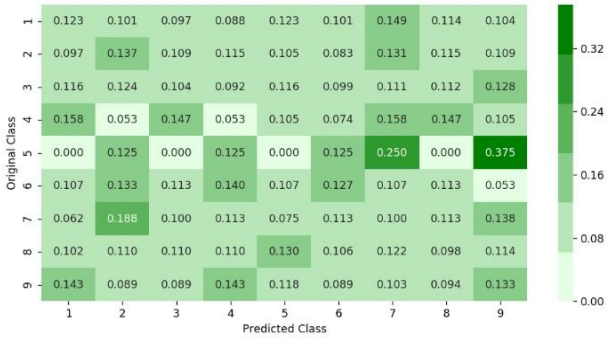


Fig 2.2 – Confusion Matrix



of the KNN classifier

Fig 1.3 – Recall Matrix of the Random Model

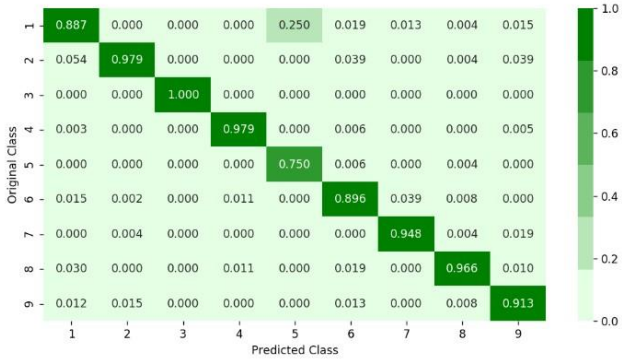


Fig 2.3 – Precision Matrix of the KNN classifier

Feature selection can be used to enhance the performance of K-Nearest Neighbo urs (KNN). Reducing dimensionality and noise in the dataset through the selection of pertinent features helps improve classification and distance calculations. Filter approaches (such as correlation -based feature selection), Wrapper methods (such as forward selection), and Embedded methods (such as L1 regularization with Lasso) are frequently used for KNN feature selection.

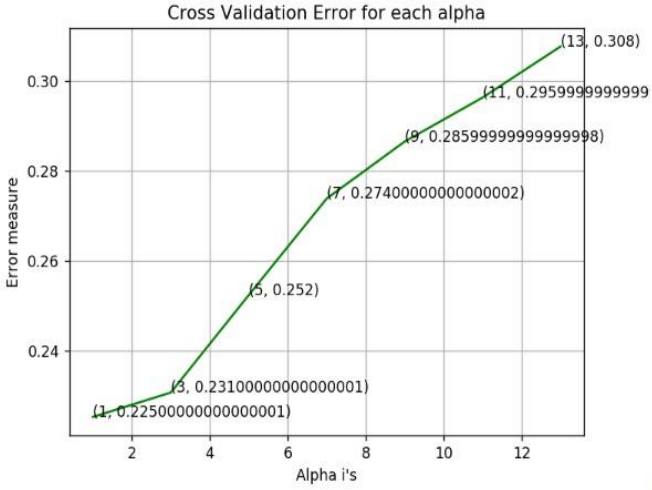


Fig 2.1 – Cross Validation, KNN Classifier

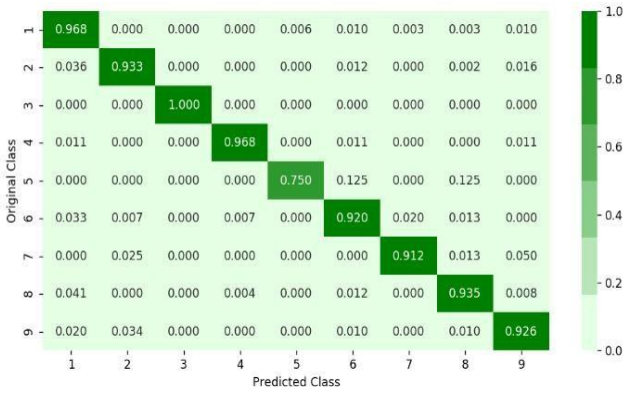


Fig 2.4 – Recall Matrix of the KNN classifier

Feature selection is frequently used in conjunction with logistic regression to improve model interpretability and lessen overfitting. The choice of features is important because logistic regression models the connection between features and the target variable. Recursive Feature Elimination (RFE), L1 regularization (Lasso), and mutual information-based algorithms are common approaches.

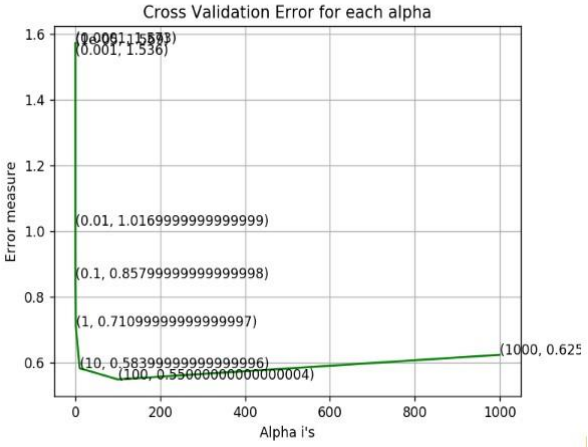


Fig 3.1 – Cross Validation, Logistic Regression

Algorithm: Multivariate Analysis on Final Features

```
xtsne = TSNE(perplexity=50)
results = xtsne.fit_transform(result_x, axis=1)
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=result_y,
cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(9))
plt.clim(0.5, 9)
plt.show()
```

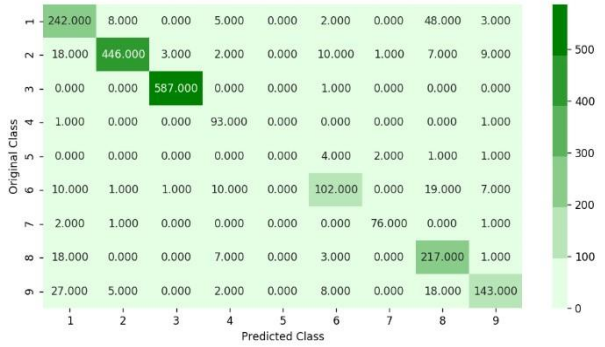


Fig 3.2 – Confusion Matrix of the Logistic Regression

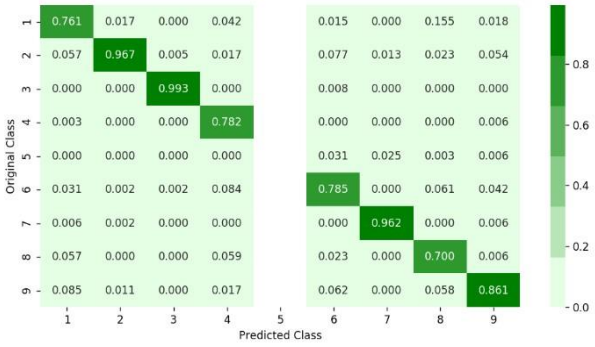


Fig 3.3 – Precision Matrix of the Logistic Regression

Fig 3.4 – Recall Matrix of the Logistic Regression

The sheer amount of data, though, is where the real intricacy lies. Data preprocessing, feature extraction, and computational resource requirements are a few of the difficulties large datasets of these files provide. Additionally, when examined together, the interactions between ".asm" and ".bytes" files can provide a more thorough knowledge of software behaviour.

Algorithm: Train and Test Split

X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y, stratify=result_y, test_size=0.20)

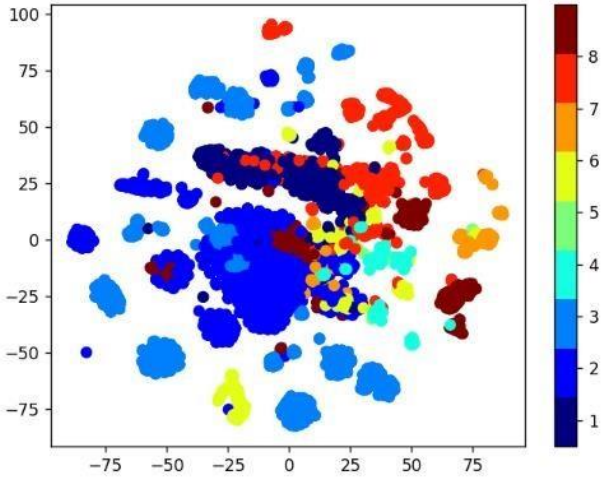


Fig 4 – Multivariate Analysis on Final Features

VI. WORKING WITH VERY LARGE DATASETS

All the results presented in this section are obtained on the large (*"scale-up"*) dataset that was described in Section II. Data analysis and machine learning need the use of big datasets of ".asm" and ".bytes" files, which is a difficult but essential task. For many applications, including malware detection, software classification, and cybersecurity research, these files frequently contain extensive data and patterns.

The ".asm" files, which are often created from disassembled software, offer a low-level perspective of a program's processes. They are composed of assembly language code, and examination of them reveals details on the operation, composition, and potential security risks of software. Alternatively, ".bytes" files offer a new perspective on the same software by encoding binary data. Often, this binary data contains important details about a program's features, structure, and even abnormalities.

X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train, stratify=y_train, test_size=0.20)

To make sense of the data in this situation, researchers and data scientists use a variety of approaches like feature engineering, deep learning, and natural language processing. The knowledge gained from these big datasets is invaluable for improving software security, spotting vulnerabilities, and comprehending the complexities of program development and execution.

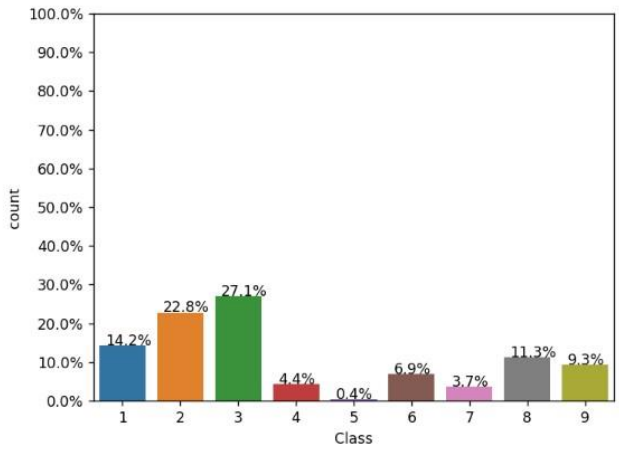


Fig – Distribution of malware in whole data set

Using the presented algorithm, it is intended to gather data on the file sizes of ".asm" files located in a directory and combine it with relevant class labels. *(Fig – Boxplot of .asm files)*

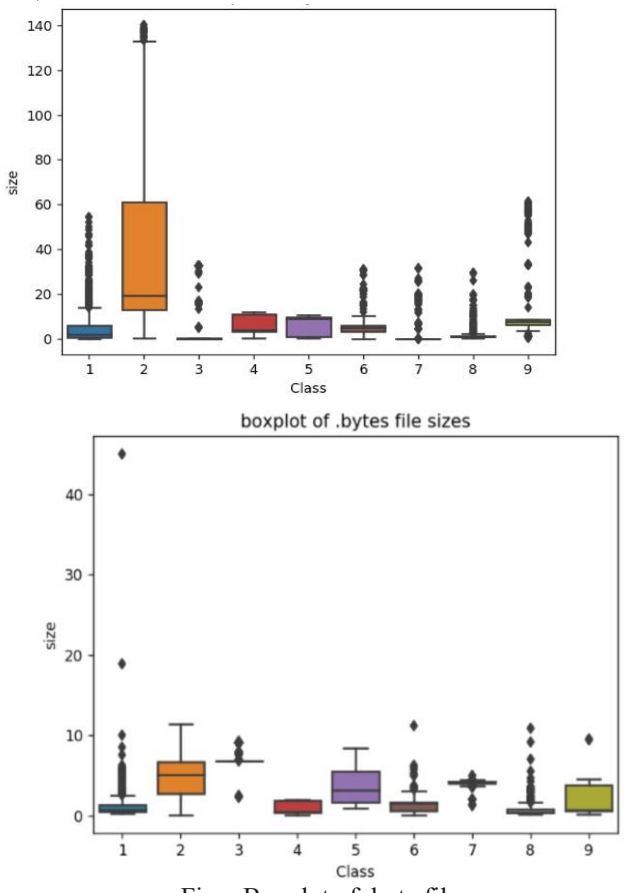


Fig – Boxplot of .byte files

Algorithm: Each .asm file size

```
files = list_files_in_directory('asmFiles')  filenames =
get_filenames_from_Y('ID')  class_y =
get_class_list_from_Y('Class')  class_bytes = []  sizebytes
= []  fnames = []

for file in files:
    statinfo = get_file_statistics('asmFiles/' + file)
    file = extract_file_name(file)

 if exists_in(filenames, file):
     i = find_index_of(filenames, file)
     class_bytes.append(class_y[i])
     sizebytes.append(convert_to_MB(statinfo.st_size))
     fnames.append(file)

asm_size_byte = create_dataframe({'ID': fnames, 'size':
sizebytes, 'Class': class_bytes})
print(asm_size_byte.head())
```

VII. Conclusion and Future Work

Our primary goal was to develop a machine learning framework with a zero false positive rate that can detect as many malware samples as possible in a generic manner. Even though we still have a non-zero false positive rate, we were really near to our target. Several deterministic exception mechanisms need to be developed for this framework to be included in a fiercely competitive commercial product. We believe that machine learning-based malware detection will complement existing anti-virus providers' standard detection techniques rather than replace them. Since every commercial anti-virus program has speed and memory constraints, the Random Model, KNN, and Logistic Regression algorithms are the most dependable ones among those shown here. Given that most antivirus programs are able to detect viruses at a rate of over 90%, an increase of 3%–4% in the overall detection rate, as achieved by our methods, is noteworthy. (Note that malware

samples that are not picked up by conventional detection techniques are used for training.)

ACKNOWLEDGMENT

We would like to express our heartfelt gratitude to Mr. Adil Husain Rathar for his invaluable guidance and mentorship throughout this project and research endeavour. We truly appreciate his persistent support and the clear guidance he gave us, which allowed us to effectively do our work within the allotted time. We also want to express our sincere gratitude to every individual in our group who worked so hard to ensure the effective completion of this research project by lending their knowledge and skills. Their commitment, collaboration, and invaluable insights were crucial in helping us reach our study objectives and identify workable answers. Finally, we would like to express our gratitude to our university for giving us a suitable platform and access to a large library, both of which tremendously helped with our research. These tools were essential to our capacity to carry out fruitful research and support the scholarly community.

REFERENCES

[1] I. Santos, Y. K. Penya, J. Devesa, and P. G. Garcia, “Ngrams-based file signatures for malware detection,” 2009.
[2] K. Rieck, T. Holz, C. Willems, P. Du¨ssel, and P. Laskov, “Learning and classification of malware behavior,” in DIMVA ’08: Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 108–125.E. Konstantinou, “Metamorphic virus: Analysis and detection,” 2008, Technical Report RHUL-MA-2008-2, Search Security Award M.Sc. thesis, 93 pagesJ. Z. Kolter and M. A. Maloof, “Learning to detect and classify malicious executables in the wild,” Journal of Machine Learning Research, vol. 7, pp. 2721–2744, December 2006, special Issue on Machine Learning in Computer Security.
[3] Y. Ye, D. Wang, T. Li, and D. Ye, “Imds: intelligent malware detection system,” in KDD, P. Berkhin, R. Caruana, and X. Wu, Eds. ACM, 2007, pp. 1043–1047.
[4] M. Chandrasekaran, V. Vidyaraman, and S. J. Upadhyaya, “Spycon: Emulating user activities to detect evasive spyware,” in IPCCC. IEEE Computer Society, 2007, pp. 502–509 [5] M. R. Chouchane, A. Walenstein, and A. Lakhotia, “Using Markov Chains to filter machine-morphed variants of malicious programs,” in Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on, 2008, pp. 77–84.
[6] M. Stamp, S. Attaluri, and S. McGhee, “Profile hidden markov models and metamorphic virus detection,” Journal in Computer Virology, 2008.
[7] R. Santamarta, “Generic detection and classification of polymorphic malware using neural pattern recognition,” 2006. [8] I. Yoo, “Visualizing Windows executable viruses using selforganizing maps,” in VizSEC/DMSEC ’04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security. New York, NY, USA: ACM, 2004, pp. 82– 89.
[9] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain,” pp. 89–114, 1988.
[10] T. Mitchell, Machine Learning. McGraw-Hill Education (ISE Editions), October 1997.