RansomwareGuard: Ransomware Detection Using Machine Learning

A PROJECT REPORT

Submitted by

RISHI BHATT [Reg No: RA2011030010185]

ASHUTOSH PAIKARAY [Reg No: RA2011030010196]

Under the Guidance of

DR. VARUN KUMAR

Assistant Professor, Department of Networking And Communications*In partial fulfillment of the requirements for the degree of* **BACHELOR OF TECHNOLOGY**

in

COMPUTER SCIENCE AND ENGINEERING

with a specialization in CYBER SECURITY

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

COLLEGE OF ENGINEERING AND TECHNOLOGYSRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603203

NOV 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this B.Tech project report titled "**RansomareGuard: Ransomware Detection Using Machine Learning**" is the bonafide work of Mr. Rishi Bhatt [Reg. No.: RA2011030010185] and Mr. Ashutosh Paikaray [Reg. No.: RA2011030010196] who carried out the project work under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

DR .Varun Kumar
SUPERVISOR
Assistant Professor.
Department of Networking and Communications

DR. ANNAPURANI PANAIYAPPAN
HEAD OF THE DEPARTMENT
Department of Networking and Communications

Examiner I

Examiner II

Department of Networking And Communications

SRM Institute of Science and Technology

Own Work Declaration Form

Degree/ Course : B.Tech in Computer Science and Engineering with specialization in Cyber Security

Student Names : Rishi Bhatt, Ashutosh Paikaray

Registration Number: RA2011030010185, RA2011030010196

Title of Work : RansomwareGuard: Ransomware Detection Using Machine Learning

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web,etc.)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s)either past or present
- Acknowledged in appropriate places any help that I have received

fromothers (e.g. fellow students, technicians, statisticians, external sources)

- Compiled with any other plagiarism criteria specified in the Coursehandbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr. T.V.Gopal**, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the projectwork.

We are incredibly grateful to our Head of the Department, **Dr. Annapurani Panaiyappan K,** Professor, Department of Networking and Communications, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinator, **Dr. G. Suseela**, Assistant Professor, Panel Head, **Dr. S. Prabakeran**, Associate Professor and members, **Dr. R. Rajkumar,** Assistant Professor, **Dr. T. Karthick,** Assistant Professor and **Dr. S. Jeeva,** Assistant Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor, **Dr. N. Krishnaraj**, Associate Professor, Department of Networking and Communications, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr. Varun Kumar**, Assistant Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under his mentorship. He provided us with the freedom and support to explore the research topics of our interest. His passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank the Networking and Communications Department staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love,constant support, and encouragement.

Rishi Bhatt [RA2011030010185]

Ashutosh Paikaray [RA2011030010196]

TABLE OF CONTENTS

C.NO	*TITLE*	*PAGE NO.*
	Abstract	ix
	List of Figures	x
	List of Symbols and Abbreviations	xi
1.	INTRODUCTION	1
	1.1 General	1
	1.2 Purpose	3
	1.3 Scope	4
	1.4 Challenges and Limitations in existing system	6
2	LITERATURE REVIEW	8
3	PROPOSED METHODOLOGY	10
	3.1 Feature analysis and selection	10
	3.2 Dataset	11
	3.3 Attributes	12
	3.4 Feature Selection	16
	3.4.1 Weight of Evidence	16
	3.4.2 Information Value	17
	3.5 Algorithms	18
	3.5.1 Logistic Regression	18
	3.5.2 K-Nearest Neighbor	19
	3.5.3 Random Forest	19
	3.5.4 Decision Tree	20
	3.5.5 Naive Bayes	21
4	RESULTS	24
	4.1 Random Forest	24
	4.2 KNN	25
	4.3 Decision Tree	26

4.4 Naive Bayes 27
4.5 Comparison Graph 28
4.6. Accuracy of each Model 29
5 CONCLUSION 30
6 FUTURE SCOPE 31
7 REFERENCES 34
APPENDIX 1 37
APPENDIX 2 41
Plagiarism Report 55

ABSTRACT

Ransomware is malware that aims to prevent access to a target's data by encrypting the data and demanding a cryptocurrency ransom in exchange for the decryption key. Traditional signature-based antivirus solutions are often unable to detect evolving ransomware variants. Therefore, machine learning algorithms have proven to be a revolutionary approach to ransomware detection. This study investigated the use of various machine learning algorithms such as K-NearestNeighbors, Random Forest, Decision Tree, Logistic Regression, and Naive Bayes to detect ransomware attacks. The main goal is to develop models that can distinguish between harmless and malicious files.The process includes data collection and data preprocessing, including data cleaning, data transformation and data splitting, followed by feature selection, model selection, model training, model evaluation, model evaluation and model validation. A system architecture diagram, use case diagram, and flowchart are provided to illustrate the key steps in the discovery process. A machine learning model trained on a dataset that covers various features, such as: B. Memory address location, system behavior and network activity. Model performance is evaluated using metrics such as accuracy, precision, confusion matrix, and ROC-AUC.

LIST OF FIGURES

FIG. NO.	Figure Titles	Page No.
3.5.1	UML Diagram	22
3.5.2	Architecture Diagram	22
3.5.3	USE Case Diagram	23
4.1.1	RF Confusion metrics	24
4.1.2	RF ROC Curve	25
4.2.1	KNN Confusion metrics	25
4.2.2	KNN ROC Curve	26
4.3.1	DT Confusion metrics	26
4.3.2	Dt ROC Curve	27
4.4.1	NB Confusion metrics	27
4.4.2	NB ROC Curve	28
4.5	Comparison Graph	28
4.6	Accuracy of each model	29

LIST OF SYMBOLS AND ABBREVIATIONS

KNN	K-Nearest Neighbor
DT	Decision Tree
RF	Random Forest
ML	Machine Learning
NB	Naive Bayes
UML	Unified Modelling Language
e.g.	exempli gratia
Eq	Equation

CHAPTER 1

INTRODUCTION

1.1 General

Fundamentally, ransomware is a type of virus or harmful software that encrypts a victim's files or data, making them unreadable. Ransomware is primarily distinguished by its demand for a ransom, usually in cryptocurrency, in order to open the encrypted files with a decryption key. A computer or network can become infected with ransomware through a number of different channels. Phishing emails, malicious attachments, compromised websites, and software and operating system flaws are among the most typical ways that ransomware spreads.

Financial gain is ransomware's main motivation. This type of malware is used by cybercriminals to demand money from their victims. A ransom note usually appears after the victim's data has been encrypted, with instructions on how to pay the ransom and obtain the decryption key. The ransom payment is highly variable, ranging from several hundred dollars to millions of dollars, contingent upon the target's assessment of the data's worth. The intention is to put the victim in a position where their only option is to pay the ransom in order to get access to their vital data again.

The use of ransomware has evolved over the years, with different strains employing varying tactics and techniques. Some ransomware variants are designed to target individual users, while others focus on organizations and businesses. Additionally, some ransomware families, like Ryuk or Sodinokibi, have gained notoriety for their sophistication and successful extortion campaigns. These attacks can disrupt business operations, result in data loss, and often lead to financial losses and reputational damage.

Detecting ransomware is a complex task due to its adaptability and the continuous emergence of new strains. Traditional antivirus software relies on known signatures or behavioral patterns to identify ransomware, but this method may not be effective against previously unseen variants. More advanced detection systems involve threat hunting, where cybersecurity experts actively search for signs of intrusion and unusual activities in a network. Anomaly detection and behavior

analysis are also essential techniques that help identify ransomware based on its activities, such as rapid encryption of files. Regular data backups are a critical defense strategy, as they enable victims to restore their data without paying a ransom, reducing the incentive for cybercriminals.

Ransomware statistics highlight the growing threat it poses to individuals, businesses, and institutions. The number of ransomware attacks has been on a sharp rise, with a 151% increase in 2021 compared to the previous year. These attacks have targeted various sectors, including healthcare, government, education, and private enterprises. Globally, ransomware attacks have resulted in astounding expenses reaching to billions of dollars, which include ransom payments, data recovery, and downtime. The use of double extortion, where cybercriminals threaten to publish stolen data, has added an additional layer of pressure on victims. As a result, ransomware remains a persistent and concerning issue in the cybersecurity landscape, necessitating robust prevention measures, incident response plans, and ongoing vigilance.

As a compelling solution to counteract this dynamic and pernicious adversary, the application of machine learning has emerged as an exceptionally promising avenue for ransomware detection. Unlike traditional antivirus methods, machine learning leverages the power of data-driven models to adapt and learn from new and previously unseen ransomware strains.
This study delves into the practical application of machine learning techniques with a particular focus on the utilization of the ensemble learning for ransomware detection. The core objective of our investigation is to develop a robust model capable of discerning between benign and malicious files.

The process of creating this model encompasses a series of pivotal steps, which include data creation which is done by scanning malicious samples and benign samples in virtual machines which will lead to extraction of datasets having various features that will determine whether the file is benign or comprising ransomware, meticulous preprocessing to ensure data quality and consistency, feature engineering to extract meaningful insights from raw data, model training to enable the algorithms to recognize ransomware patterns, and finally, deployment in operational systems to provide real-time protection.

This introduction acts as an overview for a thorough investigation of the use of the ensemble learning technique for ransomware detection. The subsequent sections will elucidate the details of the data creation and preprocessing, feature engineering, model training, and the subsequent testing of the ensemble learning model, along with an in-depth discussion of model evaluation,

and the critical aspect of continuous monitoring to ensure a robust and adaptive defense against ransomware threats

1.2 Purpose

The utilization of machine learning for ransomware detection serves a multifaceted purpose within the realm of cybersecurity. Ransomware, as a highly adaptable and continuously evolving threat, requires advanced, proactive, and intelligent methods for its identification and mitigation. Machine learning models, being data-driven and capable of autonomous learning, play a pivotal role in achieving this.

One of the fundamental purposes of integrating machine learning into ransomware detection is to harness the power of predictive analysis. These algorithms can analyze vast datasets, including historical information on previous ransomware attacks, their methodologies, and associated indicators of compromise (IoCs). By learning from these datasets, machine learning models can identify patterns, trends, and subtle deviations in data that may signal a ransomware attack. This predictive capability allows for early detection and response, reducing the potential impact of an attack and preventing the encryption of critical data.

Another crucial purpose of machine learning in ransomware detection is its adaptability. Ransomware strains can quickly mutate to evade traditional signature-based detection methods. Machine learning models, however, can adapt to new, previously unseen ransomware variants by continually updating their knowledge based on the latest threat intelligence. This adaptability is particularly valuable in the context of zero-day attacks, where vulnerabilities are exploited before security patches are available.

Furthermore, machine learning can provide a more comprehensive analysis of various data sources, enhancing detection accuracy. It can examine a multitude of features, including file access patterns, network traffic behaviors, system anomalies, and user interactions, allowing for a holistic view of a potential ransomware threat. The ability to consider multiple factors simultaneously increases the likelihood of early detection and minimizes the risk of false positives or negatives.

Additionally, machine learning contributes to reducing the economic impact of ransomware attacks. Swift identification and containment of ransomware incidents can mitigate potential data

loss, operational downtime, and reputational damage. This, in turn, reduces the urgency and incentive for victims to pay ransoms, ultimately disincentivizing cybercriminals from pursuing such attacks.

In summary, the purpose of employing machine learning in ransomware detection is to enhance the overall efficacy and efficiency of identifying and mitigating this persistent cyber threat. By leveraging predictive analysis, adaptability, and a holistic approach to data analysis, machine learning models bolster an organization's cybersecurity posture, ensuring that ransomware attacks are detected in a timely and accurate manner, thereby reducing their potential impact and financial consequences.

1.3 Scope

The scope of a project focused on implementing machine learning for ransomware detection is extensive and multifaceted, encompassing a range of crucial components.
Introduction and Objectives: The project will commence with a clear introduction that defines its overarching objectives. These objectives will be aimed at improving the efficacy and efficiency of ransomware detection while minimizing potential economic losses associated with such attacks.

Machine Learning Techniques: The study will involve a thorough investigation of supervised, unsupervised, and deep learning machine learning methods. The selection of the most suitable techniques will depend on the specific use case and the available data.

Data Collection and Preprocessing: A significant part of the project will involve identifying and gathering data from diverse sources, including network traffic logs, system behavior data, and historical ransomware attack data. Data preprocessing and feature engineering will be employed to prepare the data for machine learning analysis.

Real-time Monitoring and Response: The heart of the project lies in the development of real-time monitoring systems. These systems will be designed to continuously analyze incoming data streams, looking for potential indicators of ransomware activity. When threats are detected, the system will trigger immediate responses, such as isolating affected systems and notifying the security team.

Behavior-Based Analysis: The project scope will define mechanisms for behavior-based analysis, which is essential in identifying ransomware even when it employs evasion tactics. This approach focuses on monitoring dynamic behaviors, such as rapid file encryption or unusual access patterns, that may signify a ransomware attack.

Threat Intelligence Integration: The project aims to incorporate threat intelligence feeds in order to keep the machine learning models updated with the most recent ransomware threats and strategies. This dynamic aspect is critical for staying ahead of emerging threats.

Adaptability and Continuous Learning: The ability of machine learning models to adjust and acquire knowledge from fresh data will guarantee their efficacy in combating emerging strains of ransomware and tactics of assault.

Reducing Ransom Payments: The project will aim to define strategies to minimize the economic impact of ransomware attacks. This involves early detection and containment, reducing the incentive for victims to pay ransoms.

Customization and Scalability: The scope will allow for customization to cater to the specific needs of different organizations and sectors. Additionally, the project will ensure the system is scalable to accommodate complex and expanding network environments.

Data-Driven Decision-Making: By providing actionable insights through data analysis, the project will empower organizations to make informed decisions regarding cybersecurity strategies, resource allocation, and response tactics.

Testing and Validation: Rigorous testing and validation of machine learning models and detection systems will be a fundamental part of the project. This is essential to ensure the system performs accurately and reliably under various conditions.

Documentation and Training: The project will include the creation of comprehensive documentation and training materials for cybersecurity teams, enabling them to effectively operate, maintain, and understand the machine learning-based ransomware detection system.

Compliance and Legal Considerations: The project will address legal and ethical concerns, including compliance with data protection and privacy regulations, to ensure responsible and lawful data handling practices.

Maintenance and Updates: Ongoing maintenance plans, including regular updates to the system, will be defined to maintain its resilience and effectiveness in the face of evolving threats and changing network landscapes.

In summary, the project's scope is extensive and dynamic, encompassing a wide range of activities from model development and data analysis to real-time monitoring, adaptability, and insights generation. It is a comprehensive initiative that seeks to fortify cybersecurity measures against the persistent and evolving threat of ransomware.

1.4 Challenges and limitations in existing system:

In any existing project that employs machine learning for ransomware detection, there are several significant challenges and limitations that need to be addressed:

Data Quality and Availability: One of the primary challenges is the quality and availability of training data. Machine learning models require large, diverse datasets to be effective. However, acquiring high-quality, labeled data for ransomware incidents can be difficult, as these incidents are relatively rare and may not always be fully documented. Additionally, ensuring that the data is representative of the evolving landscape of ransomware attacks is an ongoing challenge.

False Positives and Negatives: Machine learning models are not infallible and can produce false positives and false negatives. Detecting ransomware with a high degree of accuracy while minimizing false alarms is a delicate balance. A high rate of false positives can lead to alert fatigue and an excessive workload for cybersecurity teams, while false negatives pose a significant risk by allowing actual ransomware attacks to go undetected.
Evasion Techniques: Attackers using ransomware are always creating new ways to evade detection systems. Machine learning models can therefore find it difficult to stay up to date with these changing strategies. Traditional approaches become less successful as attackers might appear innocuous by using techniques like obfuscation or encryption.

Imbalanced Data: The occurrence of ransomware attacks is significantly less frequent than normal network traffic or benign activities. This class imbalance in the data can lead to biases in machine learning models, making them more adept at detecting benign behavior but less effective

at identifying ransomware. Specialized techniques, such as resampling or using different evaluation metrics, are required to address this issue.

Generalization Challenges: Machine learning models developed for ransomware detection in one environment may not generalize well to different network setups, industries, or geographical regions. This limitation poses a challenge when deploying a model across diverse organizations, as customizations and fine-tuning may be necessary for each context.

Resource Intensiveness: For real-time analysis, some machine learning models can be quite computationally demanding and need a lot of resources. This might make them less useful for businesses with little financial resources or computational capacity.

Privacy and Ethical Concerns: Collecting and analyzing network traffic and system data for ransomware detection can raise privacy and ethical concerns. Ensuring that data handling complies with legal and ethical standards, especially in sensitive sectors like healthcare or finance, can be a significant challenge.

Adversarial Attacks: Malicious actors may actively attempt to deceive machine learning models by crafting ransomware attacks that specifically aim to evade detection. This adversarial behavior introduces a cat-and-mouse dynamic, where models need to adapt continuously to emerging evasion tactics.

User Education and Awareness: Even with sophisticated machine learning models in place, human factors remain a critical aspect of ransomware defense. Employees need to be educated and vigilant about phishing attempts and security best practices, as social engineering is often the first step in a ransomware attack.

Regulatory Compliance: Meeting regulatory requirements, such as GDPR or HIPAA, while implementing machine learning for ransomware detection can be challenging. Ensuring that data handling and security practices align with these regulations is an ongoing concern.

In conclusion, addressing these challenges and limitations is essential in developing and maintaining an effective machine learning-based ransomware detection system. A comprehensive and adaptive approach is required to overcome these obstacles and improve the accuracy and reliability of such systems in the ever-evolving landscape of cybersecurity.

CHAPTER 2

LITERATURE REVIEW

The so-called Random Forest Classifier, one of the most dependable machine learning approaches, is introduced in the paper [1] as a solution to the ransomware detection issue utilizing static analysis. This Model 1000 has an F1 measurement of over 97.8% and a ROC of over 99.6%. The best FPR, FNR, TPR, and TNR values for this model's confusion matrix at a size of 1000 features are 0.043, 0.002, 0.998, and 0.957, respectively.

The project[2] develops a random forest model, a Markov model, and a two-stage mixed ransomware detection model.After concentrating on the Windows API call sequence model for some time, he created a Markov model to represent the features of ransomware.Using the remaining data, they then create a random forest machine learning model to account for false positive (FPR) and false negative (FNR) mistakes.The two-stage mixed detection model produced an overall accuracy of 97.28%4.83% FPR and 1.47% FNR at a threshold of 0.2.

The project[3] used a variety of machine learning techniques, such as neural network-based architectures, to categorize a feature selection framework for ransomware detection and prevention according to its security level. They employed a variety of machine learning methods, including Neural Network (NN) based classifiers based on several ransomware traits and Random Forest (RF), Naive Bayes (NB), Decision Tree (DT), and Logistic Regression (LR).grouping together.The Random Forest classifier beat other classifiers, according to experimental results, by attaining the highest F-beta, accuracy, and precision values with a decent degree of consistency in 10-fold cross-validation.

Likewise, identifying the ransomware virus family is the primary objective of the research [4].The online malware file and URL scanning service VirusTotal created the dataset that was used.The methodology uses machine learning methods including Random Forest, AdaBoost, and Modified Decision Tree for malware classification in addition to the standard phases of machine learning, such as feature extraction, dataset segmentation, and dataset processing. A number of criteria, such as accuracy, sensitivity, specificity, and F1 score, were used to evaluate the

model.With an accuracy of 99.56%, the modified decision tree was the most accurate, followed by Random Forest at 99.38% and AdaBoost at 98.37%.

Similarly, the primary objective of the study [4] is to determine the family of malware known as ransomware.VirusTotal, an online malware file and URL detection service, created the dataset that was used.The approach uses machine learning methods including Random Forest, Modified Decision Tree, and AdaBoost for malware classification in addition to the standard machine learning procedures of dataset processing, feature extraction, dataset segmentation, and malware classification. A number of criteria were used to evaluate the model, including F1 score, sensitivity, specificity, and accuracy.Random Forest came in second with 99.38% accuracy, followed by the modified decision tree with 99.56%, and AdaBoost with 98.37% accuracy.

Similarly, the files that are downloaded are sent to the virtual computer using the browser extension and the location of the cloud server. It operates on the very basic tenet that, upon file download, all modifications are tracked within the virtual machine; in the event that the virtual machine fails, the file is erased by the system and is not transferred to the server. System user. The Petya ransomware simply encrypts the Master Boot Record (MBR), whereas its variation NotPetya encrypts both files and the MBR. This study[6] discusses these concepts. NotPetya exploited known vulnerabilities and functioned like a worm.

The literature analysis concludes by highlighting the variety of methods used to identify ransomware, with machine learning algorithms like Random Forest being a popular option because of their ability to achieve high accuracy rates. In order to combat the constantly changing threat landscape of ransomware, these strategies emphasize a number of different areas, such as feature selection, behavioral profiling, and static analysis. Furthermore, understanding the characteristics of ransomware variations such as Petya and NotPetya illuminates the strategies and powers of these malevolent software strains.

CHAPTER 3

PROPOSED METHODOLOGY

With the use of several machine learning methods, such as Logistic Regression, K-Nearest Neighbors (KNN), Random Forest (RF), Decision Trees (DT), and Naive Bayes (NB), we want to create an efficient ransomware detection system in this project. To guarantee the robustness and dependability of the model, we start by addressing problems with data distribution and imbalance.

Our methodology begins with an in-depth examination of ransomware, with a primary focus on dissecting its static data. The static analysis phase focuses on analyzing the binary code of ransomware, often derived from the popular Portable Executable (PE) file format commonly found in Windows executables and applications. This initial analysis serves as a cornerstone of our methodology by uncovering critical insights into the internal architecture and operational characteristics of ransomware. Understanding ransomware behavior necessitates a thorough understanding of the first phase of static analysis. This phase involves disassembling the ransomware binary, decompiling it, and scrutinizing each component individually. It plays a crucial role in unveiling the ransomware's encryption algorithms, evasion tactics, and other vital operational indicators.

Data Distribution Analysis

First, let's examine the distribution of labeled data:

Number of legitimate samples: 96,724

Number of ransomware samples: 41,323

We can observe a significant class imbalance with a larger number of legitimate samples compared to ransomware samples. To address this imbalance, we will use the Synthetic Minority Over-sampling Technique (SMOTE) and the Tomek links technique (SMOTE-TOMEK) to create a more balanced dataset.

3.1 Feature Analysis and Selection

Next, we perform an analysis of the dataset's features to select the most relevant ones for our model. To evaluate the predictive ability of each feature, we employ the Information Value (IV) and Weight of Evidence (WOE).

The top 15 features, ranked by IV, are:

1. ImageBase
2. VersionInformationSize
3. SectionsMaxEntropy
4. MajorOperatingSystemVersion
5. ResourcesMinSize
6. SizeOfStackReserve
7. Characteristics
8. SizeOfInitializedData
9. MajorSubsystemVersion
10. ResourcesNb
11. Subsystem
12. ResourcesMinEntropy
13. BaseOfData
14. SizeOfImage
15. MajorLinkerVersion

These features will be used for model training. With a well-performing model in place, you can further fine-tune the parameters, assess model robustness, and continue monitoring for new ransomware threats. Additionally, you can deploy this model in real-world scenarios to enhance cybersecurity efforts.

This study offers a strong basis for next advancements and research in the field of cybersecurity while showcasing the efficacy of machine learning in combating ransomware threats.

3.2 Dataset:

For the preparation of the dataset we used an open source software to execute ransomware in a safe environment which is- Any.Run is an interactive online sandbox for malware analysis. Cybersecurity threats are identified, looked into, and tracked by the service. Globally, clients may conduct efficient and qualitative investigations thanks to an intuitive interface. The user is in full control of the analysis flow. With ANY.RUN’s interactivity, you can: - investigate complex and unknown threats - customize emulation for each sample - directly interact with a

malicious object in real-time - get instant results and convenient reports Features Interactive approach to malware analysis You can work with a suspicious sample directly using ANY.RUN by clicking, opening, printing, and rebooting it just like you would on your own computer. Instant access Flexible configuration of a simulation Choose the locale (country, language, currency), OS version, and redirect traffic through other countries. ANY.RUN does not pass data to third parties, and only you control the access level to tasks.

My research involved the careful selection of ransomware samples .When these samples were executed within Any.Run, we observed and recorded valuable insights. These included file metadata, highlighting which file types were targeted for encryption and the specific encryption techniques employed. Furthermore, We closely examined system activities, monitoring registry modifications, process creation, and persistence mechanisms initiated by the ransomware. The environment also aided in the analysis of network traffic patterns generated by ransomware, revealing details about communication protocols, domains used for command and control, and data exfiltration routes.

This categorization ensured that the resulting dataset was organized and structured, allowing for the distinction between various ransomware strains. In preparation for machine learning model development, We undertook the critical task of feature extraction, isolating meaningful attributes that could empower the detection model. These features included insights into encryption methods, communication patterns, and attack vectors. Any.Run not only facilitated the development and training of the machine learning model but also offered a secure space for evaluation, model refinement, and adaptation to emerging ransomware threats.

3.3 Attributes:

Here are the attributes used in the dataset and their significance-

1. *Name*: This attribute likely contains the name or identifier of the binary file. While not directly used in machine learning models, it can be valuable for data management and tracking.

2. *md5*: This is a hash of the binary file. It's a unique identifier for the file and is employed to confirm the file's integrity. In ransomware detection, this attribute can be useful for quickly identifying known malicious files.

3. *Machine*: This attribute specifies the target architecture for which the binary is compiled (e.g., x86, x64). It can be important in understanding the compatibility of the file with the host system.

4. *SizeOfOptionalHeader*: This is the size, in bytes, of the optional header of the PE (Portable Executable) file. It's essential for parsing the PE file format.

5. *Characteristics*: These are flags that indicate various attributes of the binary, such as whether it is a DLL (Dynamic Link Library), whether it's executable, or whether it's a system file. These flags help in identifying the nature and purpose of the file.

6. *MajorLinkerVersion* and *MinorLinkerVersion*: These attributes indicate the version of the linker used during the compilation of the binary. They may provide insights into the file's origin and history.

7. *SizeOfCode*: This attribute represents the size of the code section within the binary. Understanding the code section's size is crucial for analyzing its functionality.

8. *SizeOfInitializedData* and *SizeOfUninitializedData*: These attributes specify the sizes of initialized and uninitialized data sections, respectively, within the binary. They are important for understanding the data structures and variables used by the file.

9. *AddressOfEntryPoint*: This is the address in memory where the execution of the binary begins. In ransomware detection, it helps identify the starting point of malicious code.
10. *BaseOfCode* and *BaseOfData*: These attributes indicate the base address in memory where the code and data sections are loaded. They provide information about the binary's memory layout.

11. *ImageBase*: This is the preferred base address for the binary when loaded into memory. It's crucial for understanding the binary's memory allocation and possible interactions with other files.

12. *SectionAlignment* and *FileAlignment*: These attributes determine how sections within the binary are aligned in memory and on disk. Proper alignment is necessary for efficient execution and loading.

13. *MajorOperatingSystemVersion* and *MinorOperatingSystemVersion*: These indicate the minimum operating system version required to run the binary. It can help determine the compatibility and target environment of the file.

14. *MajorImageVersion* and *MinorImageVersion*: Similar to the operating system version, these indicate the minimum Windows version required to run the binary.

15. *MajorSubsystemVersion* and *MinorSubsystemVersion*: These attributes indicate the version of the Windows subsystem the binary is intended to run on.

16. *SizeOfImage*: This is the size of the image in memory when loaded. It provides insights into the memory footprint of the binary.

17. *SizeOfHeaders*: It indicates the size of the headers in the PE file, which is important for various file operations and memory management.

18. *CheckSum*: This attribute is used for integrity checking of the file. It's important for detecting any tampering with the binary.

19. *Subsystem*: It specifies the subsystem required to run the binary, such as the console, graphical user interface, or native.

20. *DllCharacteristics*: These flags specify characteristics of the binary, like whether it is a dynamic link library and how it should be handled.
21. *SizeOfStackReserve, **SizeOfStackCommit, **SizeOfHeapReserve, and **SizeOfHeapCommit*: These attributes define the sizes of stack and heap memory reserved and committed for the binary. They are essential for understanding memory requirements.

22. *LoaderFlags*: Flags used by the loader, which can provide information about how the binary interacts with the loader.

23. *NumberOfRvaAndSizes*: This indicates the number of data-directory entries in the optional header. These entries point to various data structures in the PE file.

24. *SectionsNb*: The number of sections in the binary. Each section may have a different role or contain different types of code or data.

25. *SectionsMeanEntropy, **SectionsMinEntropy, **SectionsMaxEntropy*: These attributes describe the entropy (randomness) of sections. High entropy may indicate encrypted or compressed data often seen in malware.

26. *SectionsMeanRawsize, **SectionsMinRawsize, **SectionMaxRawsize*: These attributes provide statistics about the raw size of sections. They can help identify unusually large or small sections.

27. *SectionsMeanVirtualsize, **SectionsMinVirtualsize, **SectionMaxVirtualsize*: These attributes offer statistics about the virtual size of sections, helping understand the memory allocation pattern of the binary.

28. *ImportsNbDLL, **ImportsNb, and **ImportsNbOrdinal*: These attributes count the number of DLLs imported, total imports, and imports by ordinal number. They provide insights into the binary's dependencies and external functions it uses.

29. *ExportNb*: The number of functions exported by the binary, which can indicate its capabilities and functionality.

30. *ResourcesNb*: The number of resources embedded in the binary, such as icons, strings, or other data. This is important for identifying additional payloads or information.

31. *ResourcesMeanEntropy, **ResourcesMinEntropy, **ResourcesMaxEntropy*: These attributes describe the entropy of embedded resources, which can help in identifying encrypted or obfuscated data.

32. *ResourcesMeanSize, **ResourcesMinSize, **ResourcesMaxSize*: These provide statistics about the sizes of embedded resources, which can reveal information about the binary's functionality and payloads.

33. *LoadConfigurationSize*: This attribute is related to the load configuration data directory, which contains information about how the binary should be loaded. It can be crucial for understanding the binary's execution environment.

34. *VersionInformationSize*: The size of version information data. This data can include details about the file, its manufacturer, and version number, which can be useful for identifying the source of the binary.

35. *legitimate*: This is the target attribute used in machine learning, indicating whether the binary is legitimate (non-malicious) or not. It serves as the ground truth for training and evaluating models.

These attributes collectively provide a comprehensive set of features that enable machine learning models to analyze and differentiate between legitimate software and potential ransomware, making them invaluable for building effective ransomware detection systems.

3.4 Feature selection:

In the world of predictive modeling, the Weight of Evidence (WOE) and Information Value (IV) methods serve as indispensable tools, often finding their home in domains as diverse as credit scoring, marketing analytics, and risk assessment. In the following discussion, we'll elaborate on these methods in the context of your extensive dataset, composed of 52 attributes meticulously curated for ransomware analysis.

3.4.1 Weight of Evidence (WOE):

In our intricate world of predictive modeling, the Weight of Evidence (WOE) method assumes a role that can be likened to a magnifying glass through which we scrutinize the intricate patterns and connections woven into our dataset. Imagine for a moment that your dataset encompasses 52 attributes, each brimming with information concerning ransomware activities. Among these attributes, one might be the age of the user, another could be the geographical location of the target system, and yet another might capture the complexity of the ransomware code itself.

Now, WOE offers a unique perspective. It's essentially a gauge, a metric, that assists us in determining how well a particular attribute predicts a specific outcome. In your case, this outcome could be whether a ransomware attack proves successful or not. WOE goes beyond mere speculation, it's a quantitative measure that puts a number on the 'evidence' or the strength of the relationship between an attribute and the anticipated result.

Consider an example where we're assessing whether the age of a user is indicative of the likelihood of a ransomware attack's success. If a high WOE value emerges for this attribute, it indicates that age is a significant factor in predicting ransomware outcomes. It essentially tells

us that, statistically speaking, there's a strong connection between age and the success of ransomware attacks. Conversely, if the WOE value is low or negative, it means the attribute, in this case, age, isn't particularly helpful in forecasting ransomware outcomes.

Moreover, the sign of the WOE value adds nuance to the story. A positive WOE implies that the attribute makes the predicted outcome more likely. In our example, this would mean that higher age might be associated with a greater likelihood of ransomware success. On the other hand, a negative WOE suggests that the attribute diminishes the probability of the outcome. So, if the WOE is negative in our age scenario, it would mean that older users might actually be less susceptible to ransomware success.

3.4.2 Information Value (IV):

The Information Value (IV) method complements WOE by offering a holistic assessment of an attribute's predictive prowess. It looks at the bigger picture by aggregating the WOE values of all the categories or values within a given attribute.

We apply this to our dataset with the 52 attributes tailored for ransomware analysis. Suppose one of the attributes involves the geographical location of the target system. Within this attribute, there might be several categories or values representing different locations. IV takes all the WOE values associated with these categories and consolidates them into a single figure. This figure is, in essence, the grand indicator of how powerful this attribute is in predicting ransomware-related events.

In our ransomware dataset, high IV signifies that the attribute, such as geographical location, holds substantial predictive power. It implies that variations in location are strongly tied to ransomware outcomes, be it success or failure. Conversely, a low IV points to an attribute that doesn't significantly contribute to predictions regarding ransomware-related events.

So, to put it all in a broader context, when we dive into our extensive dataset with 52 attributes meticulously tailored for ransomware analysis, WOE and IV serve as discerning guides. They help us navigate through the labyrinth of data and statistical relationships, pinpointing attributes that bear the most significance in predicting ransomware-related outcomes.

By quantifying the strength of the relationship between each attribute and the predicted outcome (whether a ransomware attack succeeds or not), WOE and IV provide a structured

approach to attribute selection. It's akin to letting the data itself reveal which attributes have the strongest ties to ransomware outcomes. Armed with this knowledge, you can refine your predictive model, ensuring that it incorporates the attributes that truly matter in the realm of ransomware analysis. The result? More accurate predictions, a deeper understanding of the factors at play, and greater success in combating ransomware threats.

3.5 Algorithmss:

3.5.1 Logistic Regresion:

One of the fundamental algorithms for binary classification is logistic regression. By estimating probabilities using the logistic function, it simulates the relationship between a binary dependent variable (ransomware or not) and one or multiple independent variables (attributes or features). The algorithm is simple and easy to understand.

Significance in Ransomware Detection:
Interpretability: In ransomware detection, understanding why and how the model makes predictions is vital. Logistic Regression provides clear coefficients for each feature, allowing cybersecurity experts to identify which attributes are significant in distinguishing ransomware from legitimate software.

Low Complexity: Logistic Regression is computationally efficient and lightweight. In a real-time ransomware detection system, this efficiency is crucial. It can make quick predictions without overloading the system's resources.

Scalability: It can handle large datasets effectively. In the context of ransomware detection, where vast amounts of data need to be processed and analysed, this scalability is a significant advantage.

Early Warning: Logistic Regression is valuable in creating an early warning system. It can alert users or administrators when it detects patterns associated with ransomware-like behavior, helping mitigate potential damage before it spreads.

Feedback Loops: In a dynamic threat environment, where ransomware evolves constantly, models need to adapt. Logistic Regression can easily accommodate feedback loops and retraining with new data, making it relevant for ongoing ransomware detection efforts.

3.5.2 K-Nearest Neighborss:

An instance-based, non-parametric learning algorithm is called K-Narest Neighborss. Data points are categorized according to how similar they are to one another. When it comes to classification, the "K" stands for the number of nearest neighbors. The foundation of KNN is the hypothesis that related data points are probably members of the same class.

Significance in Ransomware Detection:

Local Patterns: KNN is excellent at identifying local patterns within data. In the context of ransomware detection, this is crucial because ransomware may exhibit specific, localized behaviors that are different from global patterns.

Flexibility: KNN is adaptable to the dynamic nature of ransomware. As new strains and variants emerge, KNN can learn from the latest local patterns, making it suitable for evolving threats.

No Assumptions: KNN makes no assumptions about the underlying data distribution. This means it can handle a wide range of ransomware behavior without requiring prior assumptions or complex model structures.

Ensemble Potential: KNN can be incorporated into ensemble methods. Ensemble models combining KNN with other algorithms can harness the strengths of both for improved ransomware detection.

Anomaly Detection: KNN can also be employed for anomaly detection. As ransomware behavior is often anomalous, KNN can flag unusual patterns, which is crucial for detecting previously unseen ransomware strains.

3.5.3 Random Forest:

An ensemble learning system called Random Forest uses several decision trees combined to provide predictions. It uses the knowledge of the crowd and works on the bagging (Bootstrap Aggregating) principle. To reduce overfitting and boost the model's resilience, each decision tree is trained using a random subset of the features and a random subset of the data.

Significance in Ransomware Detection:

Feature Importance: Random Forest excels at feature selection. When applied to ransomware detection, it helps identify the most relevant attributes that differentiate ransomware from benign software. For example, it can highlight the importance of attributes related to encryption behavior or malicious network communication patterns.

High Accuracy: Random Forest typically delivers high accuracy due to its ensemble nature. Detecting ransomware accurately is paramount, as misclassifications can lead to data loss and financial damage. The ability to combine multiple decision trees enhances the model's predictive capabilities.

Resistance to Overfitting: The randomization in feature selection and data sampling makes Random Forest resilient against overfitting, which is essential for robust ransomware detection models.

Robust to Noise: Random Forest can handle noisy data. In the context of cybersecurity, where data may be incomplete or noisy, this robustness is advantageous.

Real-Time Detection: By efficiently processing data, Random Forest can make real-time predictions. This is crucial for rapidly identifying ransomware threats and taking timely action.

3.5.4 Docision Trees:

Understanding Docision Trees:
An effective yet straightforward technique for classification problems is the decision tree. They divide the data according to attribute conditions to produce a structure resembling a tree. Decision-making is transparent and comprehensible thanks to the ensuing tree.

Significance in Ransomware Detection:
Interpretability: Decision Trees are highly interpretable. This is crucial for cybersecurity experts to understand why the model classified an instance as ransomware, aiding in the decision-making process.

Rule Extraction: Decision Trees allow for the extraction of rules that define ransomware behaviour. These rules can be used for rule-based ransomware detection systems and for educating security analysts.

Local Patterns: Decision Trees can capture local patterns in data. As ransomware behaviour can vary significantly, particularly for different ransomware families, Decision Trees can identify specific characteristics of various ransomware strains.

Early Detection: Decision Trees can be used for early detection, identifying potential ransomware-like behavior based on a sequence of attribute conditions. This can be crucial in stopping ransomware attacks before they fully manifest.

Human-Centric: Decision Trees are designed to mimic human decision-making processes. This aligns well with the need for human analysts to comprehend and act upon ransomware detection outcomes**.**

3.5.5 Naive Bayes:

Understanding Naive Bayes:
The probabilistic algorithm Naive Bayes is based on the Bayes theorem. Because it assumes that characteristics are conditionally independent of one another given the class name, it is referred to as "naive" because it makes a strong independence assumption. In practice, Naive Bayes frequently works well despite this simplification.

Significance in Ransomware Detection:

Speed and Efficiency: Naive Bayes is incredibly fast and resource-efficient. In a security context, where timely decisions are critical, Naive Bayes can provide rapid assessments of whether a file or network activity is related to ransomware.

Adaptability: Naive Bayes is adaptable and can handle dynamic environments. This adaptability is essential in ransomware detection, where threats constantly evolve.

Good Baseline: Naive Bayes serves as a useful baseline model for ransomware detection. It's quick to implement and can provide a decent level of accuracy, especially when paired with other algorithms in an ensemble approach.

Probabilistic Output: Naive Bayes provides probabilistic outputs, allowing for the ranking of potential ransomware threats. This probabilistic information can guide the allocation.

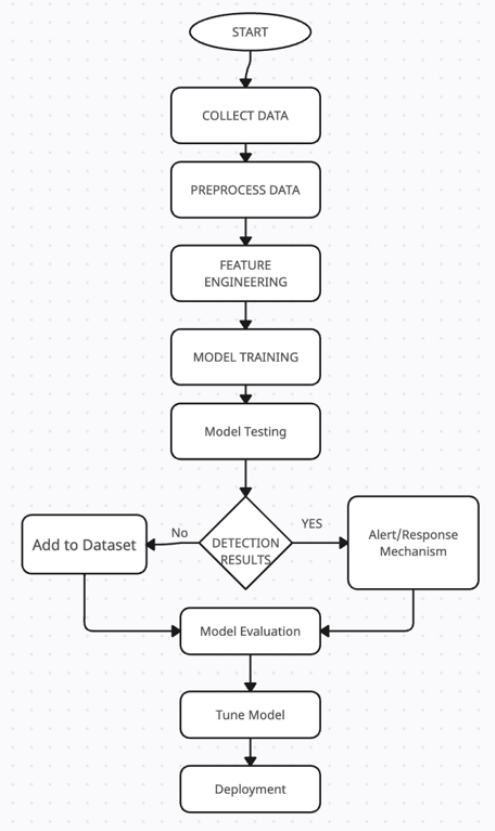


Fig 3.5.1: UML Diagram

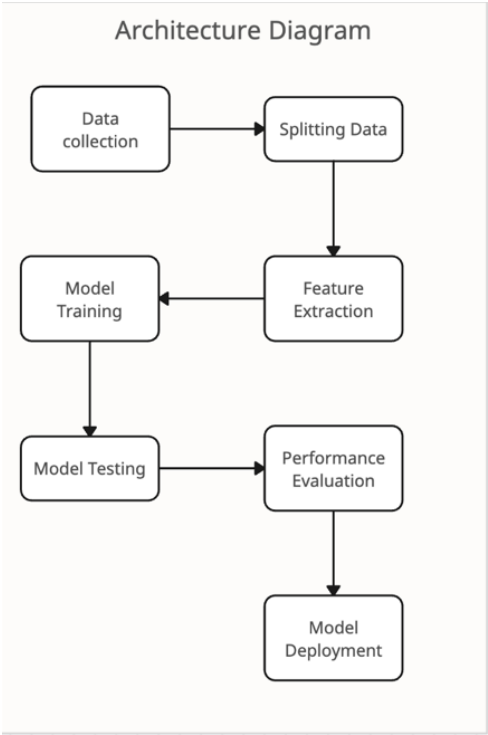


Fig 3.5.2: Architecture Diagram

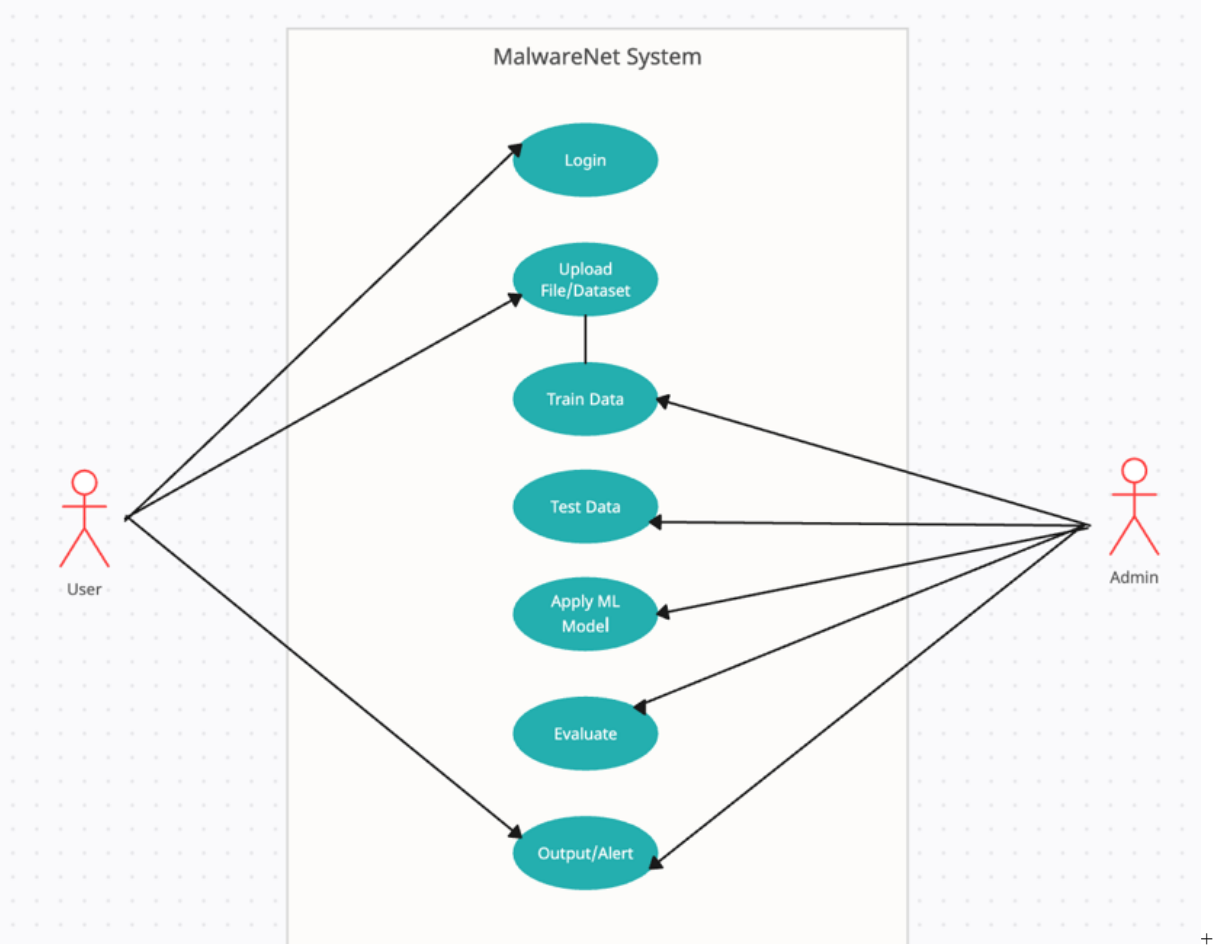


Fig 3.5.3: Usecase diagram

CHAPTER 4

RESULTS

Our model was evaluated by Logistic Regresion (LR), K- Nearest Neighbor (KNN), Naive Bayes(NB), Random Forest(RF) algorithms and Decision Tree(DT). Like every other machine learning process we have also followed the same methodology for training the data through train data and for evaluating its performance we have used testing data. To ensure that each of the sample of dataset is considered for both training and testing data, 10-fold cross validation were experimented with all model so that consistency of the model can be verified. Here are the confusion matrix of each model confusion matrix and ROC graph:

4.1 Random Forest:

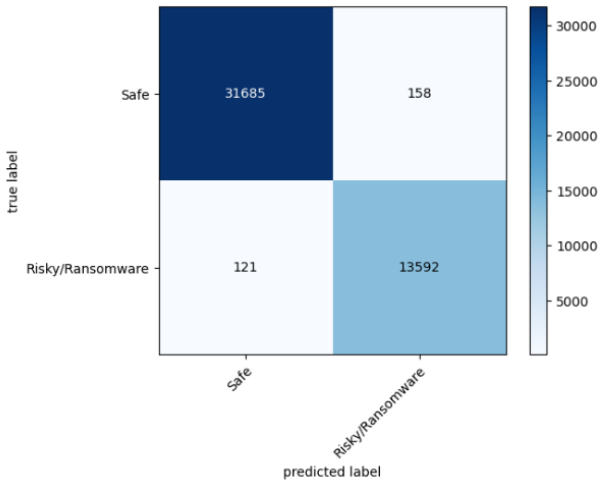


Figure 4.1.1: RF confusion metrics

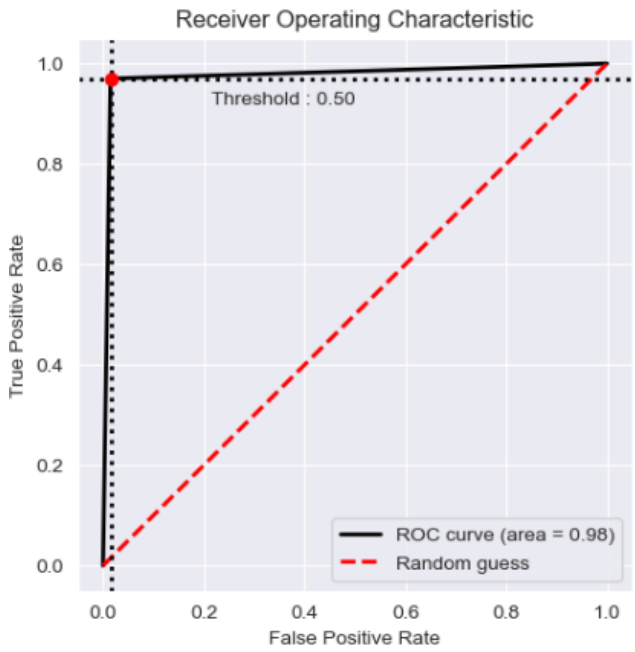


Figure 4.1.2 RF ROC curve

4.2 KNN:

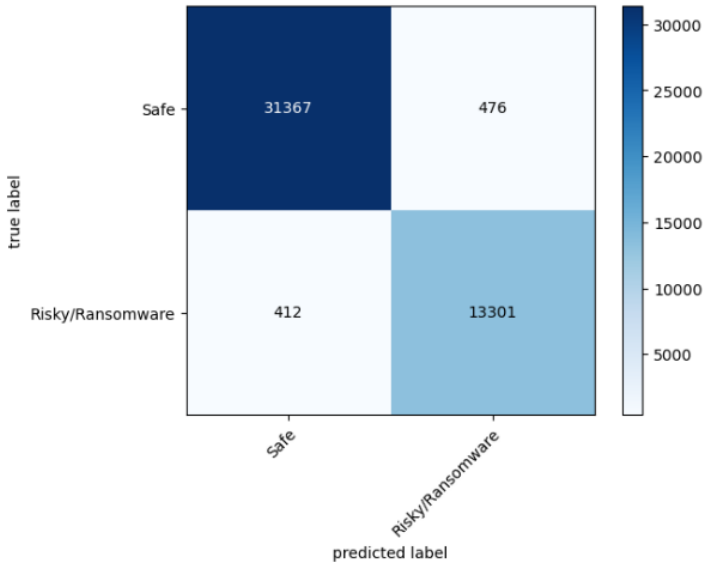


Figure 4.2.1 KNN confusion metrics

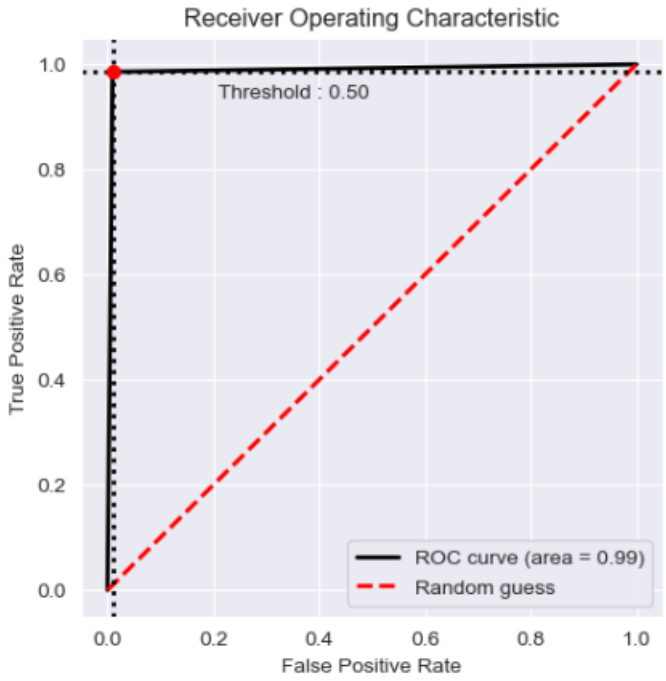


Figure 4.2.2 KNN ROC curve

4.3 Decision Tree

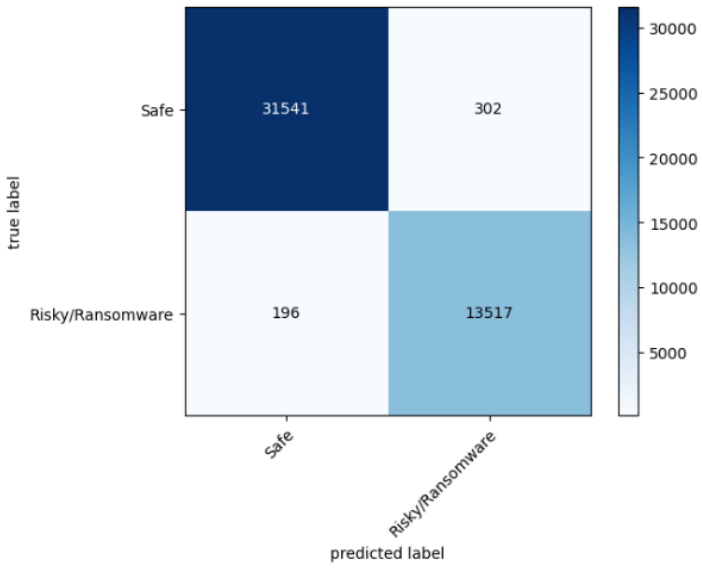


Figure 4.3.1 DT confusion metrics

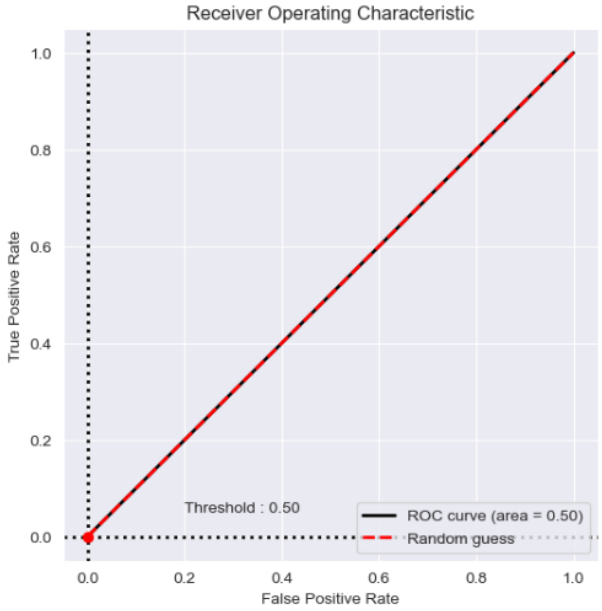


Figure 4.3.2 DT ROC curve

4.4 Naïve Bayers

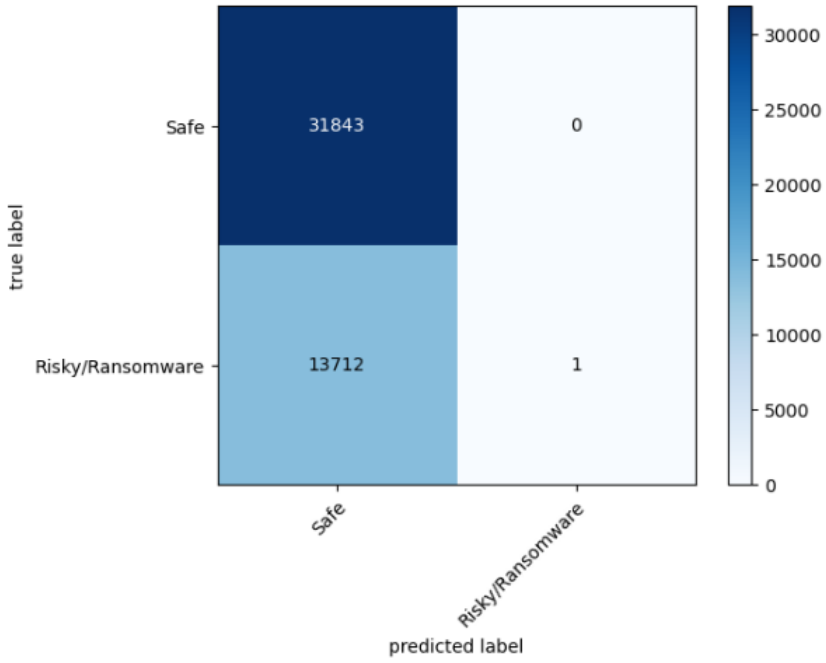


Figure 4.4.1 NB confusion metrics

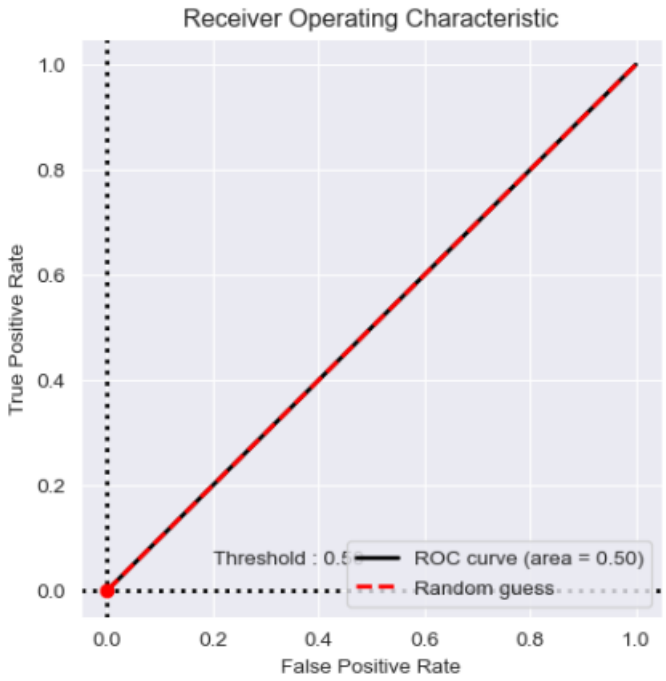


Figure 4.4.2: NB ROC curve

4.5 Comparison graph

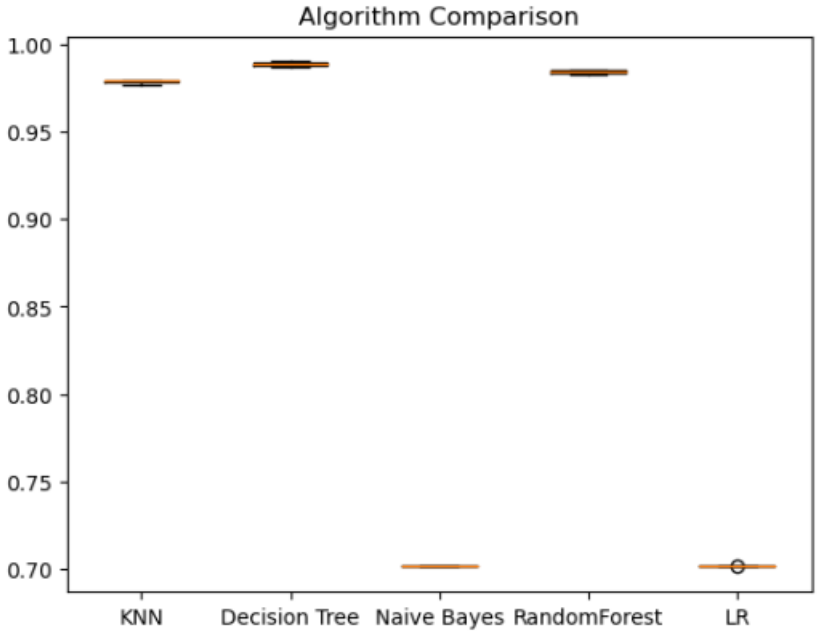


Figure 4.5.1: Comparison Graph

4.6 Accuracy of each model

```
KNN: 0.978484 (0.000941)
Decision Tree: 0.988377 (0.001060)
Naive Bayes: 0.701517 (0.000030)
RandomForest: 0.983944 (0.000913)
LR: 0.701484 (0.000006)
```

Figure 4.6.1: Accuracy model

CHAPTER 5

CONCLUSION

In the evaluation of ransomware detection methods, we assessed five different machine learning models: Logistic Regression, K- Nearest Neighbors (KNN), Randem Forest (RF), Docision Trees (DT), and Naive Bayes (NB). Each of these models has distinct strengths and weaknesses. Logistic Regression though has the lowest accuracy, stood out for its simplicity and interpretability, making it a viable choice when the relationship between features and the target variable is approximately linear. KNN has a reasonable performance demonstrating its ability to adapt to diverse data distributions, particularly when dealing with non-linear decision boundaries and localized patterns. Random Forest, an ensemble method, having a high accuracy rate excelled in handling high-dimensional data and capturing complex correlations, even though with reduced interpretability. Decision Trees is the best performing model with the highest accuracy Naive Bayes, a probabilistic model, showed one of the least efficiency and relied on the strong independence assumption between features.

In actuality, the choice of algorithm may be influenced by the particular demands and limitations of the ransomware detection assignment. Furthermore, feature engineering, the capacity to adjust to changing ransomware strategies, and the caliber and applicability of the training data all have a substantial impact on how well these algorithms function.

It is important to note that in real-world applications, a combination of these algorithms, possibly as an ensemble, might offer the best results. Additionally, staying current with threat intelligence and continuously updating and refining the models are crucial for effective ransomware detection. Ultimately, the capacity to change and adapt in the face of a constantly shifting cybersecurity scenario will determine how successful machine learning approaches are at detecting ransomware.

CHAPTER 6

FUTURE SCOPE

The future scope of a project aimed at developing a system for the live detection of ransomware in real time is a complex and dynamic landscape that requires both technological advancements and adaptive strategies.

First and foremost, the project can explore the potential of advanced machine learning models, such as deep learning tactics like convulutional neurael networks (CNNs) and recurrent neural network (RNNs). These models have the capability to understand intricate patterns in network and system behavior, allowing for more accurate and granular ransomware detection. Additionally, transfer learning and federated learning could be employed to leverage pre-trained models and to enhance the system's performance further.

The project's future scope should also encompass real-time threat intelligence integration, ensuring that the system remains current on the newest techniques and threats used by ransomware. This involves incorporating dynamic feeds of threat information and integrating with security information and event management (SIEM) systems, allowing the system to respond proactively to emerging threats.

As ransomware attackers become increasingly sophisticated, the project should explore adversarial machine learning techniques. These strategies are essential in countering attackers who actively attempt to evade detection systems by crafting ransomware attacks specifically designed to bypass defenses.

Moreover, extending the system's protection to the burgeoning landscape of Internet of Things (IoT) devices and bolstering endpoint security is imperative. The future scope involves adapting the system to monitor and safeguard these devices, which often lack robust security measures.
The analysis of blockchain transactions to trace ransom payments and identify perpetrators presents an emerging frontier in the fight against ransomware. By developing methods to analyze

cryptocurrency transactions on the blockchain, the system can potentially aid in the identification and prosecution of ransomware operators.

Enhanced user training and awareness programs represent a critical aspect of future scope. Preventing ransomware attacks from occurring in the first place through user education is pivotal, and the project can involve developing user-friendly tools and resources to educate employees and individuals about the risks and how to recognize phishing attempts.

The future scope should also include automation and orchestration of incident response. As ransomware threats evolve, automating responses based on real-time detection will help contain threats rapidly, reducing potential damage and response time.

Furthermore, as organizations increasingly move to cloud and hybrid environments, ensuring seamless integration with these platforms will be crucial. The system should be scalable to accommodate a variety of network environments, from traditional on-premises setups to complex, multi-cloud infrastructures.

Strengthening privacy-preserving capabilities to address compliance and privacy concerns is paramount. The project's future iterations should consider compliance with emerging data protection regulations and ethical considerations, ensuring that data handling aligns with international standards.

With the advent of quantum computing, preparing the system to withstand quantum threats is a forward-looking consideration. Quantum-safe encryption techniques should be explored and integrated to protect against quantum-based ransomware attacks.

Proactive threat hunting and red teaming exercises represent an essential element of the future scope. These exercises help continuously evaluate the system's effectiveness and readiness against evolving ransomware tactics, ensuring that it remains resilient and adaptive.

Finally, fostering global collaboration and sharing of threat intelligence among organizations, industries, and countries is imperative. In an interconnected digital world, working together to combat ransomware threats and share insights is critical for collective defense against these evolving cyber threats.

In summary, the future scope of the project is characterized by innovation, adaptability, and continuous vigilance. It encompasses technological advancements, user education, global

collaboration, and an unwavering commitment to staying ahead of ransomware threats in an ever-evolving cybersecurity landscape.

CHAPTER 7

REFERENCES

1. M. Masum, M. J. Hossain Faruk, H. Shahriar, K. Qian, D. Lo and M. I. Adnan, "Ransomware Classification and Detection With Machine Learning Algorithms," *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, 2022, pp. 0316-0322, doi: 10.1109/CCWC54503.2022.9720869.
2. Mohammed, Ban. (2020). Ransomware Detection using Random Forest Technique. ICT Express. 6. 325-331. 10.1016/j.icte.2020.11.001.
3. Hwang, J., Kim, J., Lee, S. *et al.* Two-Stage Ransomware Detection Using Dynamic Analysis and Machine Learning Techniques. *Wireless Pers Commun* **112**, 2597–2609 (2020).
4. F. Khan, C. Ncube, L. K. Ramasamy, S. Kadry and Y. Nam, "A Digital DNA Sequencing Engine for Ransomware Detection Using Machine Learning," in IEEE Access, vol. 8, pp. 119710-119719, 2020, doi: 10.1109/ACCESS.2020.3003785.
5. Ö. Aslan, M. Ozkan-Okay and D. Gupta, "Intelligent Behavior-Based Malware Detection System on Cloud Computing Environment," in IEEE Access, vol. 9, pp. 83252-83271, 2021, doi: 10.1109/ACCESS.2021.3087316.
6. R. Kumar, K. Sethi, N. Prajapati, R. R. Rout and P. Bera, "Machine Learning based Malware Detection in Cloud Environment using Clustering Approach," 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kharagpur, India, 2020, pp. 1-7, doi: 10.1109/ICCCNT49239.2020.9225627.
7. J. Kimmell, M. Abdelsalam and M. Gupta, "Analyzing Machine Learning Approaches for Online Malware Detection in Cloud," in 2021 IEEE International Conference on Smart Computing (SMARTCOMP), Irvine, CA, USA, 2021 pp. 189-196.

8. Mohammed K. Alzaylaee, Suleiman Y. Yerima, Sakir Sezer,DL-Droid: Deep learning based android malware detection using real devices,Computers & Security,Volume 89,2020,101663,

9. Gomez-Hernandez, Jose & Sánchez-Fernández, Raúl & García-Teodoro, Pedro. (2021). Inhibiting crypto-ransomware on windows platforms through a honeyfile-based approach with R-Locker. IET Information Security. 16. 10.1049/ise2.12042.

10. K. P. Subedi, D. R. Budhathoki and D. Dasgupta, "Forensic Analysis of Ransomware Families Using Static and Dynamic Analysis," 2018 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 2018, pp. 180-185, doi: 10.1109/SPW.2018.00033.

11. Jin-Young Kim, Sung-Bae Cho,Obfuscated Malware Detection Using Deep Generative Model based on Global/Local Features,Computers & Security,2022,102501,

12. Ullah, Faizan & Javaid, Qaisar & Salam, Abdu & Ahmed, Masood & Sarwar, Nadeem & Shah, Dilawar & Abrar, Muhammad. (2020). Modified Decision Tree Technique for Ransomware Detection at Runtime through API Calls. Scientific Programming. 2020. 1-10. 10.1155/2020/8845833.

13. Ren, Amos & Liang, Chong & Hyug, Im & Brohi, Sarfraz & Jhanjhi, Noor. (2018). A Three-Level Ransomware Detection and Prevention Mechanism. EAI Endorsed Transactions on Energy Web. 7. 162691. 10.4108/eai.13-7-2018.162691.

14. Jian Du, Sajid Hussain Raza, Mudassar Ahmad, Iqbal Alam, Saadat Hanif Dar, Muhammad Asif Habib, "Digital Forensics as Advanced Ransomware Pre-Attack Detection Algorithm for Endpoint Data Protection", *Security and Communication Networks*, vol. 2022, Article ID 1424638, 16 pages, 2022.

15. Yang, C., Xu, J., Liang, S. *et al.* DeepMal: maliciousness-Preserving adversarial instruction learning against static malware detection. *Cybersecur* **4**, 16 (2021). https://doi.org/10.1186/s42400-021-00079-5

16. Sihwail, Rami & Omar, Khairuddin & Zainol Ariffin, Khairul Akram. (2021). An Effective Memory Analysis for Malware Detection and Classification. Computers, Materials and Continua. 67. 2301-2320. 10.32604/cmc.2021.014510.

17. Hitaj, Dorjan & Pagnotta, Giulio & De Gaspari, Fabio & De Carli, Lorenzo & Mancini, Luigi. (2023). Minerva: A File-Based Ransomware Detector.

18. Azeez, N.A.; Odufuwa, O.E.; Misra, S.; Oluranti, J.; Damaševičius, R. Windows PE Malware Detection Using Ensemble Learning. *Informatics* **2021**, *8*, 10. https://doi.org/10.3390/informatics8010010
19. Markus Ring, Daniel Schlör, Sarah Wunderlich, Dieter Landes, Andreas Hotho,Malware detection on windows audit logs using LSTMs,Computers & Security,Volume 109,2021,102389,
20. ISSN 0167-4048,Z. Pan, C. Feng and C. Tang, "Malware Classification Based on the Behavior Analysis and Back Propagation Neural Network", ITM Web of Conferences, vol. 7, p. 02001

A. Sung, J. Xu, P. Chavez and S. Mukkamala, "Static Analyzer of Vicious Executables (SAVE)", 20th Annual Computer Security Applications Conference.

APPENDIX 1

This section contains details on the language, software and packages used in our project.

This project is developed in Python, which is a general-purpose interpreted, interactive, object oriented and high-level programming language. It offers concise and readable code. Despite being highly complex with versatile workflows, the AI and ML algorithms, when written in Python, can help the developers create robust and reliable machine intelligent systems. The list of Python packages used in our project are:

- **Numpy:** It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O.
    - Multidimensional Arrays: Large datasets can be efficiently manipulated with Numpy's ndarray, a homogeneously typed array. The foundation of many Python libraries for science and mathematics is this array object.
    - NumPy offers an extensive set of functions for mathematical, logical, and statistical operations on arrays. Because these operations are performance-optimized, NumPy is an essential part of numerical computations.

- **Pandas:** It is a Python package that provides fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive.
    - Data Structures: Series and DataFrame are the two main data structures introduced by Pandas. A series is an object that resembles a one-dimensional array and can hold any kind of data. A DataFrame is a tabular structure that is two-dimensional and has labelled axes. It is similar to a spreadsheet in that it may hold data.
    - Data Alignment: Data alignment is a fundamental component of Pandas. It removes the requirement for explicit data alignment and streamlines data operations by automatically aligning data based on label values.

- **Random:** The random module is a built-in module that provides functions for generating random numbers and performing various randomization tasks. It is commonly

used in applications such as simulations, games, statistical sampling, cryptography, and more.

- Random Number Generation: Functions to produce random floating-point and integer numbers are available in the random module. To regulate how random the generated values are, you can set ranges, seed values, and other settings.
- Random Sequences: It offers routines for selecting random items at random from a sequence and for rearranging sequences (such as lists). These features are in handy when you want to randomly choose things from a list or shuffle cards in a card game.

- **Matplotlib:** It is a comprehensive library for creating static, animated, and interactive visualizations in Python. It can create publication quality plots, make interactive figures that can zoom, pan, update, customize visual style and layout, export to many file formats and can be embedded in JupyterLab and Graphical User Interfaces.
  - Support for Backends: Matplotlib offers support for a variety of rendering backends. It can be set up to function with a number of graphical environments, such as Qt, Agg, GTK, and others. This enables it to produce charts for a variety of scenarios, including standalone GUI apps and inline rendering in Jupyter notebooks.
  - Plot kinds: A multitude of plot kinds, such as line plots, scatter plots, bar charts, histograms, and more, are supported by Matplotlib. There are choices for colour, markers, labels, and axes customization.

- **Keras:** It is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Up until version 2.3, Keras supported multiple backends, including TensorFlow, Microsoft Cognitive Toolkit, Theano, and PlaidML.
  - Modularity: The design of Keras prioritises modularity. It offers an easy-to-use, high-level API for creating neural network designs. Layers can be stacked to construct complicated models, and each layer has own customization options.
  - Backends: TensorFlow, Theano, Microsoft Cognitive Toolkit (CNTK), and PlaidML were among the few backends that Keras originally supported. Nevertheless, Keras became an essential component of TensorFlow with TensorFlow 2.0, streamlining the machine learning ecosystem.

- **Pickle:** The pickle library in Python is a module used for serializing and deserializing Python objects. Serialization is the process of converting a Python object into a format that can be easily stored or transmitted, while deserialization is the process of reconstructing the original Python object from the serialized data. Pickle is a common way to save and load data in Python, particularly when working with complex data structures and custom objects.Here are some key features and functionalities of the pickle library:
    - Serialization and Deserialization: Pickle allows you to serialize Python objects (convert them into a byte stream) using the pickle.dump() function, and then later deserialize them (convert the byte stream back into Python objects) using pickle.load().
    - Handling Various Data Types: Pickle can handle a wide variety of Python data types, including basic data types like numbers and strings, as well as more complex data structures like lists, dictionaries, classes, and even functions.
    - Cross-Version Compatibility: Pickle is designed to be compatible across different Python versions, which means you can serialize an object in one Python version and deserialize it in another version without any major issues. However, there are some limitations and compatibility concerns, especially when moving between major Python versions.
    - Security Concerns: Be cautious when unpickling data from untrusted or unauthenticated sources, as pickle can execute arbitrary code during deserialization. This can pose a security risk if you unpickle data from untrusted sources, so it's generally recommended to avoid unpickling data from untrusted sources or to use alternative serialization formats when security is a concern.

- **Sklearn:** scikit-learn, often abbreviated as sklearn, is a popular and widely-used machine learning library in Python. It provides a range of tools and algorithms for machine learning, data preprocessing, feature selection, and model evaluation. Here's a brief overview of the key features and functionalities of scikit-learn:
    - Simple and Consistent API: Scikit-learn offers a simple and consistent API that makes it easy to use various machine learning algorithms. This consistency simplifies the process of switching between different algorithms and models.
    - Supervised and Unsupervised Learning: Scikit-learn supports both supervised learning (classification and regression) and unsupervised learning (clustering, dimensionality reduction, etc.).

- Preprocessing and Feature Extraction: It provides tools for data preprocessing, including data scaling, encoding categorical variables, and feature selection.
- Rich Collection of Algorithms: Scikit-learn includes a wide variety of machine learning algorithms, such as linear and logistic regression, decision trees, support vector machines, k-nearest neighbors, random forests, and more.
- Model Evaluation and Selection: The library offers tools for model evaluation, cross-validation, hyperparameter tuning, and model selection, making it easier to assess and compare different models.
- Integration with NumPy and Pandas: Scikit-learn seamlessly integrates with popular data manipulation libraries like NumPy and Pandas, allowing you to work with data in a familiar format.
- Community and Ecosystem: It has a large and active community, with extensive documentation and resources available. Scikit-learn is often used in conjunction with other data science libraries like NumPy, Pandas, and Matplotlib.
- Open Source and Free: Scikit-learn is an open-source library distributed under a permissive license, making it free to use and modify.

APPENDIX 2

The code used in this section develops the algorithm and trains the model utilized in this project.

```
[11]: import os
      import pandas as pd
      import numpy as np
      from matplotlib import pyplot as plt
      import pickle
      import sklearn.ensemble as ek
      from sklearn import tree, linear_model
      from sklearn.feature_selection import SelectFromModel
      import joblib
      from sklearn.naive_bayes import GaussianNB
      from sklearn.metrics import confusion_matrix
      from sklearn.pipeline import make_pipeline
      from sklearn import preprocessing
      from sklearn import svm
      from sklearn.linear_model import LogisticRegression
      from statsmodels.stats.outliers_influence import variance_inflation_factor as␣
       ↪vif
      from sklearn.model_selection import train_test_split
      from mlxtend.plotting import plot_confusion_matrix
      import seaborn as sns

      import warnings
      warnings.filterwarnings("ignore")

[12]: df=pd.read_csv("Ransomware.csv",sep='|')

[13]: df

[13]:                                          Name  \
      0                                  memtest.exe
      1                                      ose.exe
      2                                    setup.exe
      3                                     DW20.EXE
      4                                  dwtrig20.exe
      ...                                         ...
      138042  VirusShare_8e292b418568d6e7b87f2a32aee7074b
      138043  VirusShare_260d9e2258aed4c8a3bbd703ec895822

                                                    1
```

```
138044  VirusShare_8d088a51b7d225c9f5d11d239791ec3f
138045  VirusShare_4286dccf67ca220fe67635388229a9f3
138046  VirusShare_d7648eae45f09b3adb75127f43be6d11

                                md5  Machine  SizeOfOptionalHeader  \
0       631ea355665f28d4707448e442fbf5b8      332                   224
1       9d10f99a6712e28f8acd5641e3a7ea6b      332                   224
2       4d92f518527353c0db88a70fddcfd390      332                   224
3       a41e524f8d45f0074fd07805ff0c9b12      332                   224
4       c87e561258f2f8650cef999bf643a731      332                   224
...                                  ...      ...                   ...
138042  8e292b418568d6e7b87f2a32aee7074b      332                   224
138043  260d9e2258aed4c8a3bbd703ec895822      332                   224
138044  8d088a51b7d225c9f5d11d239791ec3f      332                   224
138045  4286dccf67ca220fe67635388229a9f3      332                   224
138046  d7648eae45f09b3adb75127f43be6d11      332                   224

        Characteristics  MajorLinkerVersion  MinorLinkerVersion  SizeOfCode  \
0                   258                   9                   0      361984
1                  3330                   9                   0      130560
2                  3330                   9                   0      517120
3                   258                   9                   0      585728
4                   258                   9                   0      294912
...                 ...                 ...                 ...         ...
138042              258                  11                   0      205824
```

[0 rows x 57 columns]

[15]:
```
from sklearn.metrics import f1_score
import sklearn.metrics
```

[16]:
```
df.describe()
```


[16]:

	Machine	SizeOfOptionalHeader	Characteristics	\
count	138047.000000	138047.000000	138047.000000	
mean	4259.069274	225.845632	4444.145994	
std	10880.347245	5.121399	8186.782524	
min	332.000000	224.000000	2.000000	
25%	332.000000	224.000000	258.000000	
50%	332.000000	224.000000	258.000000	
75%	332.000000	224.000000	8226.000000	
max	34404.000000	352.000000	49551.000000	

	MajorLinkerVersion	MinorLinkerVersion	SizeOfCode	\
count	138047.000000	138047.000000	1.380470e+05	
mean	8.619774	3.819286	2.425956e+05	
std	4.088757	11.862675	5.754485e+06	
min	0.000000	0.000000	0.000000e+00	
25%	8.000000	0.000000	3.020800e+04	
50%	9.000000	0.000000	1.136640e+05	
75%	10.000000	0.000000	1.203200e+05	
max	255.000000	255.000000	1.818587e+09	

	SizeOfInitializedData	SizeOfUninitializedData	AddressOfEntryPoint	\
count	1.380470e+05	1.380470e+05	1.380470e+05	
mean	4.504867e+05	1.009525e+05	1.719561e+05	
std	2.101599e+07	1.635288e+07	3.430553e+06	
min	0.000000e+00	0.000000e+00	0.000000e+00	
25%	2.457600e+04	0.000000e+00	1.272100e+04	
50%	2.631680e+05	0.000000e+00	5.288300e+04	
75%	3.850240e+05	0.000000e+00	6.157800e+04	
max	4.294966e+09	4.294941e+09	1.074484e+09	

	BaseOfCode	…	ResourcesNb	ResourcesMeanEntropy	\
count	1.380470e+05	…	138047.000000	138047.000000	
mean	5.779845e+04	…	22.050700	4.000127	
std	5.527658e+06	…	136.494244	1.112981	
min	0.000000e+00	…	0.000000	0.000000	
25%	4.096000e+03	…	5.000000	3.458505	
50%	4.096000e+03	…	6.000000	3.729824	
75%	4.096000e+03	…	13.000000	4.233051	
max	2.028711e+09	…	7694.000000	7.999723	

[17]:
```
# Checking the size of dataframe
from sys import getsizeof
initial_size = getsizeof(df)/(1024.0**3)
print("Size of DataFrame: {} GB".format(initial_size))
```

```
Size of DataFrame: 0.07982608210295439 GB
```

[18]:
```
df.isnull().sum()
```

[18]:
```
Name                        0
md5                         0
Machine                     0
SizeOfOptionalHeader        0
Characteristics             0
MajorLinkerVersion          0
MinorLinkerVersion          0
SizeOfCode                  0
SizeOfInitializedData       0
SizeOfUninitializedData     0
AddressOfEntryPoint         0
BaseOfCode                  0
BaseOfData                  0
```



```
ImageBase                   0
SectionAlignment            0
FileAlignment               0
MajorOperatingSystemVersion 0
MinorOperatingSystemVersion 0
MajorImageVersion           0
MinorImageVersion           0
MajorSubsystemVersion       0
MinorSubsystemVersion       0
SizeOfImage                 0
SizeOfHeaders               0
CheckSum                    0
Subsystem                   0
DllCharacteristics          0
SizeOfStackReserve          0
SizeOfStackCommit           0
SizeOfHeapReserve           0
SizeOfHeapCommit            0
LoaderFlags                 0
NumberOfRvaAndSizes         0
SectionsNb                  0
SectionsMeanEntropy         0
SectionsMinEntropy          0
SectionsMaxEntropy          0
SectionsMeanRawsize         0
SectionsMinRawsize          0
SectionMaxRawsize           0
SectionsMeanVirtualsize     0
SectionsMinVirtualsize      0
SectionMaxVirtualsize       0
ImportsNbDLL                0
ImportsNb                   0
ImportsNbOrdinal            0
ExportNb                    0
ResourcesNb                 0
ResourcesMeanEntropy        0
ResourcesMinEntropy         0
```

```
[19]: df.legitimate.value_counts() #1 means legitimate, 0 means malware
```



```
[19]: legitimate
      0    96724
      1    41323
      Name: count, dtype: int64

[20]: # Converting labelled data in categories datatype
      df.legitimate = df.legitimate.astype('category')
      df.legitimate

[20]: 0         1
      1         1
      2         1
      3         1
      4         1
               ..
      138042    0
      138043    0
      138044    0
      138045    0
      138046    0
      Name: legitimate, Length: 138047, dtype: category
      Categories (2, int64): [0, 1]

[21]: df.md5.nunique()

[21]: 138047

[22]: df.md5.shape[0]

[22]: 138047

[23]: df.shape[1]

[23]: 57

[24]: df.columns

[24]: Index(['Name', 'md5', 'Machine', 'SizeOfOptionalHeader', 'Characteristics',
             'MajorLinkerVersion', 'MinorLinkerVersion', 'SizeOfCode',
             'SizeOfInitializedData', 'SizeOfUninitializedData',
             'AddressOfEntryPoint', 'BaseOfCode', 'BaseOfData', 'ImageBase',
             'SectionAlignment', 'FileAlignment', 'MajorOperatingSystemVersion',
             'MinorOperatingSystemVersion', 'MajorImageVersion', 'MinorImageVersion',
             'MajorSubsystemVersion', 'MinorSubsystemVersion', 'SizeOfImage',
             'SizeOfHeaders', 'CheckSum', 'Subsystem', 'DllCharacteristics',
             'SizeOfStackReserve', 'SizeOfStackCommit', 'SizeOfHeapReserve',
             'SizeOfHeapCommit', 'LoaderFlags', 'NumberOfRvaAndSizes', 'SectionsNb',
```

[25]:
```
df.dtypes
```

[25]:
```
Name                        object
md5                         object
Machine                      int64
SizeOfOptionalHeader         int64
Characteristics              int64
MajorLinkerVersion           int64
MinorLinkerVersion           int64
SizeOfCode                   int64
SizeOfInitializedData        int64
SizeOfUninitializedData      int64
AddressOfEntryPoint          int64
BaseOfCode                   int64
BaseOfData                   int64
ImageBase                  float64
SectionAlignment             int64
FileAlignment                int64
MajorOperatingSystemVersion  int64
MinorOperatingSystemVersion  int64
MajorImageVersion            int64
MinorImageVersion            int64
MajorSubsystemVersion        int64
MinorSubsystemVersion        int64
SizeOfImage                  int64
SizeOfHeaders                int64
CheckSum                     int64
Subsystem                    int64
DllCharacteristics           int64
SizeOfStackReserve           int64
SizeOfStackCommit            int64
SizeOfHeapReserve            int64
SizeOfHeapCommit             int64
LoaderFlags                  int64
NumberOfRvaAndSizes          int64
SectionsNb                   int64
SectionsMeanEntropy        float64
```



```
SectionsMinEntropy         float64
SectionsMaxEntropy         float64
SectionsMeanRawsize        float64
SectionsMinRawsize           int64
SectionMaxRawsize            int64
SectionsMeanVirtualsize    float64
SectionsMinVirtualsize       int64
SectionMaxVirtualsize        int64
ImportsNbDLL                 int64
ImportsNb                    int64
ImportsNbOrdinal             int64
ExportNb                     int64
ResourcesNb                  int64
ResourcesMeanEntropy       float64
ResourcesMinEntropy        float64
ResourcesMaxEntropy        float64
ResourcesMeanSize          float64
ResourcesMinSize             int64
```

[26]:
```
# Using VIF to remove highly correlated columns
from statsmodels.stats.outliers_influence import variance_inflation_factor


cols_vif = df.columns.tolist()
cols_vif.remove('legitimate')
cols_vif.remove('md5')
cols_vif.remove('Name')
cols_vif


# VIF dataframe
vif_data = pd.DataFrame()
vif_data["feature"] = cols_vif
```

[27]:
```
df = df.dropna()
```

[28]:
```
df.drop(['Name','md5'],axis=1)
```

[28]:

	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	\
0	332	224	258	9	
1	332	224	3330	9	
2	332	224	3330	9	

3	332	224	258	9	
4	332	224	258	9	
…	…	…	…	…	
138042	332	224	258	11	
138043	332	224	33167	2	
138044	332	224	258	10	
138045	332	224	33166	2	
138046	332	224	258	11	

	MinorLinkerVersion	SizeOfCode	SizeOfInitializedData	\
0	0	361984	115712	
1	0	130560	19968	
2	0	517120	621568	
3	0	585728	369152	
4	0	294912	247296	
…	…	…	…	
138042	0	205824	223744	
138043	25	37888	185344	
138044	0	118272	380416	
138045	25	49152	16896	
138046	0	111616	468480	

	SizeOfUninitializedData	AddressOfEntryPoint	BaseOfCode	…	\
0	0	6135	4096	…	
1	0	81778	4096	…	
2	0	350896	4096	…	
3	0	451258	4096	…	
4	0	217381	4096	…	
…	…	…	…	…	
138042	0	123291	4096	…	
138043	0	40000	4096	…	
138044	0	59610	4096	…	
138045	0	51216	4096	…	
138046	0	22731	4096	…	

	ResourcesNb	ResourcesMeanEntropy	ResourcesMinEntropy	\
0	4	3.262823	2.568844	
1	2	4.250461	3.420744	
2	11	4.426324	2.846449	
3	10	4.364291	2.669314	
4	2	4.306100	3.421598	

```
[ ]: # Using VIF to remove highly correlated columns
from statsmodels.stats.outliers_influence import variance_inflation_factor


cols_vif = df.columns.tolist()
cols_vif.remove('legitimate')
cols_vif.remove('md5')
cols_vif.remove('Name')
cols_vif
```



```
# VIF dataframe
vif_data = pd.DataFrame()
vif_data["feature"] = cols_vif
```

```
[ ]: df = df.dropna()
```

```
[ ]: # calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(df[cols_vif].values, i)
                          for i in range(len(cols_vif))]

print(vif_data)
```

```
[ ]: df.
↪drop(['MinorImageVersion','MinorSubsystemVersion','SizeOfHeapCommit','SectionsMinRawsiz
↪'],axis=1)
```

```
[69]: def iv_woe(data, target, bins=10, show_woe=False):

    #Empty Dataframe
    newDF,woeDF = pd.DataFrame(), pd.DataFrame()

    #Extract Column Names
    cols = data.columns

    #Run WOE and IV on all the independent variables
    for ivars in cols[~cols.isin([target])]:
        if (data[ivars].dtype.kind in 'bifc') and (len(np.
↪unique(data[ivars]))>10):
            binned_x = pd.qcut(data[ivars], bins,  duplicates='drop')
            d0 = pd.DataFrame({'x': binned_x, 'y': data[target]})
        else:
            d0 = pd.DataFrame({'x': data[ivars], 'y': data[target]})
        d = d0.groupby("x", as_index=False).agg({"y": ["count", "sum"]})
        d.columns = ['Cutoff', 'N', 'Events']
        d['% of Events'] = np.maximum(d['Events'], 0.5) / d['Events'].sum()
        d['Non-Events'] = d['N'] - d['Events']
        d['% of Non-Events'] = np.maximum(d['Non-Events'], 0.5) /␣
↪d['Non-Events'].sum()
        d['WoE'] = np.log(d['% of Events']/d['% of Non-Events'])
        d['IV'] = d['WoE'] * (d['% of Events'] - d['% of Non-Events'])
        d.insert(loc=0, column='Variable', value=ivars)
        print("Information value of " + ivars + " is " + str(round(d['IV'].
↪sum(),6)))
        temp =pd.DataFrame({"Variable" : [ivars], "IV" : [d['IV'].sum()]},␣
↪columns = ["Variable", "IV"])
        newDF=pd.concat([newDF,temp], axis=0)
```

```
        woeDF=pd.concat([woeDF,d], axis=0)

        #Show WOE Table
        if show_woe == True:
            print(d)
    return newDF, woeDF
```

[70]:
```
df.legitimate = df.legitimate.dropna()
df.legitimate = df.legitimate.fillna(0)
```

[71]:
```
df.legitimate = df.legitimate.astype('int64')
```

[72]:
```
iv, woe = iv_woe(df.drop(['Name'],axis=1), 'legitimate')
```

```
Information value of md5 is 1.240653
Information value of Machine is 2.596527
Information value of SizeOfOptionalHeader is 2.596853
Information value of Characteristics is 3.823743
Information value of MajorLinkerVersion is 2.787002
Information value of MinorLinkerVersion is 0.583745
Information value of SizeOfCode is 2.5136
Information value of SizeOfInitializedData is 3.569039
Information value of SizeOfUninitializedData is 0.328101
Information value of AddressOfEntryPoint is 2.351206
Information value of BaseOfCode is 0.031301
Information value of BaseOfData is 2.974971
Information value of ImageBase is 6.097249
Information value of SectionAlignment is 0.163236
Information value of FileAlignment is 0.153303
Information value of MajorOperatingSystemVersion is 4.34342
Information value of MinorOperatingSystemVersion is 0.475603
Information value of MajorImageVersion is 0.181241
Information value of MinorImageVersion is 0.17365
Information value of MajorSubsystemVersion is 3.566766
Information value of MinorSubsystemVersion is 0.872094
Information value of SizeOfImage is 2.965568
Information value of SizeOfHeaders is 0.117683
Information value of CheckSum is 1.991779
Information value of Subsystem is 3.071376
Information value of DllCharacteristics is 2.484924
Information value of SizeOfStackReserve is 4.161116
Information value of SizeOfStackCommit is 0.000917
Information value of SizeOfHeapReserve is 0.001089
Information value of SizeOfHeapCommit is 0.006985
Information value of LoaderFlags is 0.0
Information value of NumberOfRvaAndSizes is 0.000592
Information value of SectionsNb is 1.298768
```

[35]:
```
features = iv.sort_values(by = 'IV', ascending=False)['Variable'][:15].values.
↪tolist()
```

[36]:
```
features
```

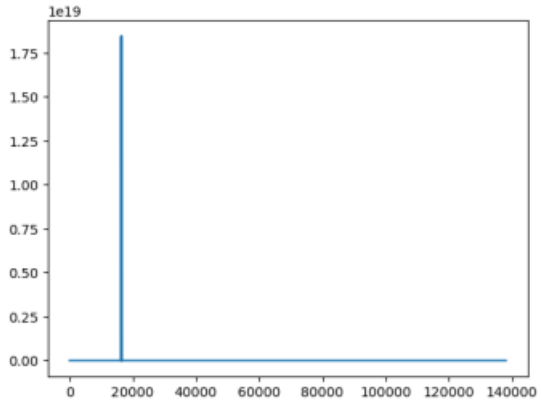
[36]:
```
['ImageBase',
 'VersionInformationSize',
 'SectionsMaxEntropy',
 'MajorOperatingSystemVersion',
 'ResourcesMinSize',
 'SizeOfStackReserve',
 'Characteristics',
```



```
 'SizeOfInitializedData',
 'MajorSubsystemVersion',
 'SectionsMinVirtualsize',
 'ResourcesNb',
 'Subsystem',
 'ResourcesMinEntropy',
 'BaseOfData',
 'SizeOfImage']
```

[37]:
```
plt.plot(df.ImageBase)
```

[37]:
```
[<matplotlib.lines.Line2D at 0x1ee86a4c390>]
```



[38]:
```
X=df[features]
y=df['legitimate']
```

[39]:
```
randomseed = 42
```

[40]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
↪random_state=42)
```

[41]:
```
print(X_test.shape[0] + X_train.shape[0])
print('Training labels shape:', y_train.shape)
print('Test labels shape:', y_test.shape)
print('Training features shape:', X_train.shape)
print('Test features shape:', X_test.shape)
```

```
138047
Training labels shape: (92491,)
Test labels shape: (45556,)
Training features shape: (92491, 15)
Test features shape: (45556, 15)
```

[42]:
```
from collections import Counter
#import imblearn
```

[43]:
```
from sklearn.linear_model import LogisticRegression
```

[44]:
```
df.dtypes
```

[44]:
```
Name                         object
md5                          object
Machine                       int64
SizeOfOptionalHeader          int64
Characteristics               int64
MajorLinkerVersion            int64
MinorLinkerVersion            int64
SizeOfCode                    int64
SizeOfInitializedData         int64
SizeOfUninitializedData       int64
AddressOfEntryPoint           int64
BaseOfCode                    int64
BaseOfData                    int64
ImageBase                   float64
SectionAlignment              int64
FileAlignment                 int64
MajorOperatingSystemVersion   int64
MinorOperatingSystemVersion   int64
MajorImageVersion             int64
MinorImageVersion             int64
MajorSubsystemVersion         int64
MinorSubsystemVersion         int64
SizeOfImage                   int64
SizeOfHeaders                 int64
CheckSum                      int64
Subsystem                     int64
DllCharacteristics            int64
SizeOfStackReserve            int64
```



```
SizeOfStackCommit             int64
SizeOfHeapReserve             int64
SizeOfHeapCommit              int64
LoaderFlags                   int64
NumberOfRvaAndSizes           int64
SectionsNb                    int64
SectionsMeanEntropy         float64
SectionsMinEntropy          float64
SectionsMaxEntropy          float64
SectionsMeanRawsize         float64
SectionsMinRawsize            int64
SectionMaxRawsize             int64
SectionsMeanVirtualsize     float64
SectionsMinVirtualsize        int64
SectionMaxVirtualsize         int64
ImportsNbDLL                  int64
```

```
dtype: object
```

[45]:
```
df.drop('md5',axis=1)
```

[45]:
```
                                        Name  Machine  \
0                                  memtest.exe      332
1                                     ose.exe      332
2                                   setup.exe      332
3                                    DW20.EXE      332
4                                  dwtrig20.exe      332
...                                         ...      ...
138042  VirusShare_8e292b418568d6e7b87f2a32aee7074b      332
138043  VirusShare_260d9e2258aed4c8a3bbd703ec895822      332
138044  VirusShare_8d088a51b7d225c9f5d11d239791ec3f      332
138045  VirusShare_4286dccf67ca220fe67635388229a9f3      332
138046  VirusShare_d7648eae45f09b3adb75127f43be6d11      332

        SizeOfOptionalHeader  Characteristics  MajorLinkerVersion  \
```



```
0                        224              258                   9
1                        224             3330                   9
2                        224             3330                   9
3                        224              258                   9
4                        224              258                   9
...                      ...              ...                 ...
138042                   224              258                  11
138043                   224            33167                   2
138044                   224              258                  10
138045                   224            33166                   2
138046                   224              258                  11

        MinorLinkerVersion  SizeOfCode  SizeOfInitializedData  \
0                        0      361984                 115712
1                        0      130560                  19968
2                        0      517120                 621568
3                        0      585728                 369152
4                        0      294912                 247296
...                    ...         ...                    ...
138042                   0      205824                 223744
138043                  25       37888                 185344
138044                   0      118272                 380416
138045                  25       49152                  16896
138046                   0      111616                 468480

        SizeOfUninitializedData  AddressOfEntryPoint  ...  ResourcesNb  \
0                             0                6135  ...            4
1                             0               81778  ...            2
2                             0              350896  ...           11
3                             0              451258  ...           10
4                             0              217381  ...            2
...                         ...                 ...  ...          ...
138042                        0              123291  ...            7
138043                        0               40000  ...           26
138044                        0               59610  ...           22
138045                        0               51216  ...           10
138046                        0               22731  ...            4

        ResourcesMeanEntropy  ResourcesMinEntropy  ResourcesMaxEntropy  \
0                   3.262823             2.568844             3.537939
1                   4.250461             3.420744             5.080177
2                   4.426324             2.846449             5.271813
```

[46]:
```
"""
Before SMOTE_Tomek
"""
counter_train = Counter(y_train)
counter_test = Counter(y_test)
print(counter_train, counter_test)

# Create the classifier object
classifier = LogisticRegression(class_weight = 'balanced')

# Train the classifier on the resampled data
classifier.fit(X_train, y_train)

# Check the new counts of the train and test sets
counter_train = Counter(y_train)
```



```
counter_test = Counter(y_test)
print(counter_train, counter_test)
```

```
Counter({0: 64881, 1: 27610}) Counter({0: 31843, 1: 13713})
Counter({0: 64881, 1: 27610}) Counter({0: 31843, 1: 13713})
```

[47]:
```
print(X_test.shape[0] + X_train.shape[0])
print('Training labels shape:', y_train.shape)
print('Test labels shape:', y_test.shape)
print('Training features shape:', X_train.shape)
print('Test features shape:', X_test.shape)
```

```
138047
Training labels shape: (92491,)
Test labels shape: (45556,)
Training features shape: (92491, 15)
Test features shape: (45556, 15)
```

[48]:
```
import sys
!{sys.executable} -m pip install lazypredict
```

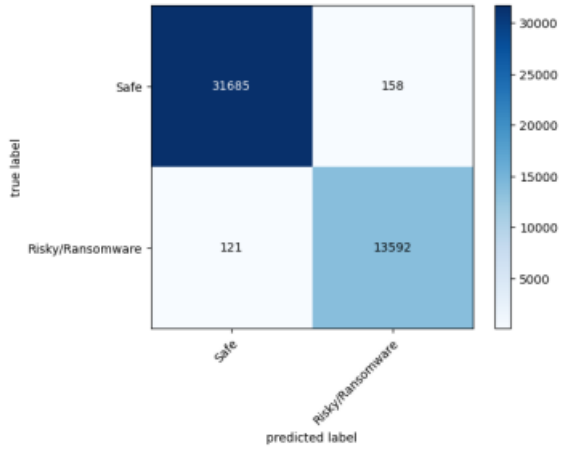
```
Requirement already satisfied: lazypredict in c:\users\ashut\anaconda3\lib\site-
packages (0.2.12)
Requirement already satisfied: click in c:\users\ashut\anaconda3\lib\site-
packages (from lazypredict) (8.0.4)
Requirement already satisfied: scikit-learn in
c:\users\ashut\anaconda3\lib\site-packages (from lazypredict) (1.2.2)
Requirement already satisfied: pandas in c:\users\ashut\anaconda3\lib\site-
packages (from lazypredict) (2.0.3)
Requirement already satisfied: tqdm in c:\users\ashut\anaconda3\lib\site-
packages (from lazypredict) (4.65.0)
Requirement already satisfied: joblib in c:\users\ashut\anaconda3\lib\site-
packages (from lazypredict) (1.2.0)
Requirement already satisfied: lightgbm in c:\users\ashut\anaconda3\lib\site-
packages (from lazypredict) (4.1.0)
Requirement already satisfied: xgboost in c:\users\ashut\anaconda3\lib\site-
packages (from lazypredict) (2.0.1)
Requirement already satisfied: colorama in c:\users\ashut\anaconda3\lib\site-
packages (from click->lazypredict) (0.4.6)
Requirement already satisfied: numpy in c:\users\ashut\anaconda3\lib\site-
packages (from lightgbm->lazypredict) (1.24.3)
Requirement already satisfied: scipy in c:\users\ashut\anaconda3\lib\site-
packages (from lightgbm->lazypredict) (1.11.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
c:\users\ashut\anaconda3\lib\site-packages (from pandas->lazypredict) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
c:\users\ashut\anaconda3\lib\site-packages (from pandas->lazypredict)
(2023.3.post1)
```

```
[55]: rf.fit(X_train,y_train)

[55]: RandomForestClassifier(random_state=42)

[56]: pred = rf.predict(X_test)

[57]: cm=confusion_matrix(y_test,pred)
      cm

[57]: array([[31685,   158],
             [  121, 13592]], dtype=int64)

[58]: rf.score(X_test,y_test)

[58]: 0.9938756695056633

[62]: kn = KNeighborsClassifier(n_neighbors=29)

[63]: kn.fit(X_train,y_train)
```



```
[63]: KNeighborsClassifier(n_neighbors=29)

[64]: pred1= kn.predict(X_test)

[65]: cm1 = confusion_matrix(y_test,pred1)
      cm1

[65]: array([[31367,   476],
             [  412, 13301]], dtype=int64)

[66]: kn.score(X_test,y_test)

[66]: 0.9805075072438317

[67]: type_classify=['Legitimate','Malware']
      count_classify=[41323,96724]
      plt.pie(count_classify, labels=type_classify, autopct='%0.f%%')

[67]: ([<matplotlib.patches.Wedge at 0x1ee8a822290>,
         <matplotlib.patches.Wedge at 0x1ee8a7baad0>],
        [Text(0.6484073958497663, 0.8885763045497695, 'Legitimate'),
         Text(-0.6484073958497659, -0.8885763045497698, 'Malware')],
        [Text(0.35367676137259974, 0.4846779842998742, '30%'),
         Text(-0.35367676137259957, -0.4846779842998743, '70%')])
```

[254]:
```
# Classes
classes = ['Safe', 'Risky/Ransomware']

figure, ax = plot_confusion_matrix(conf_mat = cm,
                                   class_names = classes,
                                   colorbar = True)

plt.show()
```



[61]:
```
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```



```
from sklearn.naive_bayes import GaussianNB
from  sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

[270]:
```
# Spot Check Algorithms
models = []
models.append(('KNN', KNeighborsClassifier(n_neighbors=29)))
models.append(('Decision Tree', DecisionTreeClassifier(min_samples_leaf=30)))
models.append(('Naive Bayes', GaussianNB()))
models.append(('RandomForest', RandomForestClassifier(n_estimators=80,␣
 ↪max_depth=3,random_state=0,min_samples_leaf=9)))
models.append(("LR",LogisticRegression()))

# evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = StratifiedKFold(n_splits=5, random_state=1, shuffle=True)
    cv_results = cross_val_score(model, X_train, y_train.values.ravel(),␣
 ↪cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))
# Compare Algorithms
pyplot.boxplot(results, labels=names)
pyplot.title('Algorithm Comparison')
pyplot.show()
```

```
KNN: 0.978484 (0.000941)
Decision Tree: 0.988377 (0.001060)
Naive Bayes: 0.701517 (0.000030)
RandomForest: 0.983944 (0.000913)
LR: 0.701484 (0.000006)
```

Plagiarism Report

ashutosh

ORIGINALITY REPORT

3%	2%	2%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	ebin.pub Internet Source	<1%
2	www.researchgate.net Internet Source	<1%
3	www.mdpi.com Internet Source	<1%
4	Jinsoo Hwang, Jeankyung Kim, Seunghwan Lee, Kichang Kim. "Two-Stage Ransomware Detection Using Dynamic Analysis and Machine Learning Techniques", Wireless Personal Communications, 2020 Publication	<1%
5	Rajat Shri Shrimal, Jyoti Gajrani, Vinesh Kumar Jain, Meenakshi Tripathi, Dharm Singh Jat. "Detection of Ransomware Attacks Using Weight of Evidence Technique", 2023 International Conference on Emerging Trends in Networks and Computer Communications (ETNCC), 2023 Publication	<1%

6	Submitted to University of Stirling Student Paper	<1 %
7	Jaskaran Singh, Keshav Sharma, Mohammad Wazid, Ashok Kumar Das. "SINN-RD: Spline interpolation-envisioned neural network-based ransomware detection scheme", Computers and Electrical Engineering, 2023 Publication	<1 %
8	Submitted to University of Alabama Student Paper	<1 %
9	Mohammad Masum, Md Jobair Hossain Faruk, Hossain Shahriar, Kai Qian, Dan Lo, Muhaiminul Islam Adnan. "Ransomware Classification and Detection With Machine Learning Algorithms", 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC), 2022 Publication	<1 %
10	www.coursehero.com Internet Source	<1 %
11	www.e3s-conferences.org Internet Source	<1 %
12	koreascience.kr Internet Source	<1 %
13	hrcak.srce.hr Internet Source	<1 %