A Comparative Investigation on the Application of Machine Learning for Ransomware Detection

A
Project Report
Submitted in partial fulfilment of the
Requirements for the award of the Degree of

BACHELOR OF ENGINEERING

IN

INFORMATION TECHNOLOGY

By

Nikitha Maramraju, 1602-19-737-145

Guggilla Shri Pallavi, 1602-19-737-171

Under the guidance of

D.R.L. Prasanna

Assistant Professor, Dept. of IT

Department of Information Technology

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad-31 2023

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Information Technology

DECLARATION BY THE CANDIDATE

We, **Nikitha Maramraju** and **Guggilla Shri Pallavi** bearing hall ticket number, **1602-19-737-145** and **1602-19-737-171** hereby declare that the project report entitled **A Comparative Investigation on the Application of Machine Learning for Ransomware Detection** under the guidance of **D.R.L. Prasanna, Assistant Professor**, Department of Information Technology, Vasavi College of Engineering, Hyderabad, is submitted in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering** in **Information Technology**

This is a record of bonafide work carried out by us and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

Nikitha Maramraju
1602-19-737-145

Guggilla Shri Pallavi
1602-19-737-171

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Information Technology

BONAFIDE CERTIFICATE

This is to certify that the project entitled **A Comparative Investigation on the Application of Machine Learning for Ransomware Detection** being submitted by **Nikitha Maramraju** and **Guggilla Shri Pallavi** bearing **1602-19-737-145** and **1602-19-737-171** in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Information Technology is a record of bonafide work carried out by them under my guidance.

D.R.L. Prasanna	**Dr. K. Ram Mohan Rao**	**External Examiner**
Assistant Professor, IT	**Professor & HOD, IT**	
Internal Guide		

ACKNOWLEDGEMENT

The satisfaction that accompanies that the successful completion of the Main project would not have been possible without the kind support and help of many individuals. We would like to extend our sincere thanks to all of them.

It is with immense pleasure that we would like to take the opportunity to express our humble gratitude to **D.R.L. Prasanna, Assistant Professor, Dept. of Information Technology** under whom we executed this project. We are also grateful to **Sathya Devi Maranganti, Assistant Professor, Dept. of Information Technology** for her guidance. Their constant guidance and willingness to share their vast knowledge made us understand this project and its manifestations in great depths and helped us to complete the assigned tasks.

We are very much thankful to **Dr. K. Ram Mohan Rao, Professor and HOD, Dept. of Information Technology,** for his kind support and for providing necessary facilities to carry out the work.

We wish to convey our special thanks to **Dr. S. V. Ramana, Principal of Vasavi College of Engineering and Management** for providing facilities. Not to forget, we thank all other faculty and non-teaching staff, and my friends who had directly or indirectly helped and supported me in completing my project in time.

ABSTRACT

Ransomware refers to a type of malicious software designed to encrypt a user's data and prevent them from accessing it until a ransom payment is made. The encrypted data is held hostage until the payment is made, with the threat of permanent loss or destruction of the data if the payment is not received. In order to evade detection, ransomware employs various techniques during its attack flow, including the use of Machine Learning (ML) algorithms. Therefore, it is important to comprehend the techniques used in the development of ransomware, as well as their deployment strategies, in order to gain a better understanding of their attack flow and develop suitable countermeasures. The project aims to provide insights into the effectiveness of machine learning techniques in detecting ransomware.

Our proposal is to design a Ransomware detection system using machine learning. The dataset is preprocessed to ensure its quality and suitability for analysis. To address potential multicollinearity issues, the Variance Inflation Factor (VIF) is applied to identify and mitigate high correlations between independent variables. This process helps enhance the accuracy and stability of subsequent machine learning models. Next, we employed a hybridized resampling technique – Synthetic Minority Over Sampling Techniques with Tomek Links (SMOTE + Tomek Links) to handle the imbalanced dataset. Next, various machine learning models are implemented for ransomware detection. The Random Forest algorithm is utilized due to its ability to handle high-dimensional data and capture complex patterns. Additionally, Linear Support Vector Classification and Bernoulli Naive Bayes models are applied to compare their performance against Random Forest. The implemented models are trained using the balanced dataset and also the unbalanced dataset. Their performance is evaluated using appropriate evaluation metrics such as accuracy, precision, recall, and F1-score. Experimental results have shown us that random forest can predict and detect ransomware accurately and effectively with an accuracy of 99%.

TABLE OF CONTENTS

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Full Form
SMOTE	Synthetic Minority Oversampling Techniques
VIF	Variance Inflation Factor
RF	Random Forest
NB	Naive Bayes
SVC	Support Vector Classifiction
TPR	True Positive Rate
FPR	False Positive Rate
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative

1. INTRODUCTION

Ransomware attacks have become one of the most prevalent and disruptive forms of cybercrime in recent years. These attacks involve encrypting a victim's files or system and demanding a ransom in exchange for the decryption key. Ransomware attacks can cause significant damage to individuals and organizations, resulting in data loss, financial losses, and reputational damage. In recent years, ransomware attacks have become more common and complex. As per a report by Cybersecurity Ventures, it is predicted that the global costs of damage caused by ransomware will reach up to $20 billion by the year 2021.

As ransomware continues to evolve and adapt to defensive measures, there is a pressing need to develop advanced detection techniques to effectively combat this menace. Traditional signature-based antivirus systems are no longer sufficient to protect against ransomware attacks. This is due to the rapidly evolving nature of ransomware, with attackers frequently modifying their tactics to evade detection. Therefore, there is a need for more sophisticated and adaptive techniques for detecting ransomware attacks. Machine learning, with its ability to learn patterns and detect anomalies, offers promising solutions for ransomware detection.

1.1. PROBLEM STATEMENT

Machine learning has emerged as a promising technique for detecting ransomware attacks. By training models on large datasets of both malicious and benign software, machine learning algorithms can identify patterns and features that distinguish ransomware from legitimate software. This approach allows for real-time detection and response to ransomware attacks, reducing the impact and damage caused by these attacks. By using machine learning, it is possible to detect ransomware attacks in real-time and respond quickly to mitigate the damage.

The aim of this research paper is to explore the application of machine learning techniques in the detection of ransomware. By leveraging a diverse dataset that encompasses both ransomware and legitimate software samples, we endeavor to build an efficient and accurate ransomware detection system. The foundation of this research lies in the acquisition of a comprehensive dataset that accurately represents the characteristics of ransomware and legitimate software. This dataset serves as the basis for training and evaluating machine learning models, enabling us to harness their predictive capabilities to accurately classify and detect ransomware instances. We present a comprehensive review of the different machine learning techniques used for ransomware detection. We evaluate the effectiveness of these techniques and compare the performance of these models which include feature selection and data balancing techniques. We also compare the performance of these models with and without data balancing techniques and evaluate the effectiveness of the detection system.

1.2. PROPOSED WORK

In this project, our goal was to develop a machine learning model that could accurately detect ransomware attacks in real-time. To do this, we first collected a large dataset of ransomware and benign files, which we used to train and validate our model. We then employed a combination of feature engineering and model selection techniques to optimize the performance of our model.

One of the key challenges we faced during this project was the imbalanced nature of our dataset, with ransomware samples being significantly less frequent than benign files. To tackle the class imbalance problem, we employ various techniques such as Synthetic Minority Over-sampling Technique (SMOTE) and undersampling methods like Tomek Links. SMOTE generates synthetic instances of the minority class, effectively balancing the dataset, while Tomek Links identify and remove potentially noisy or ambiguous samples, further improving the model's generalization ability. They help to balance the dataset and improve the accuracy of the model.

Then, we applied several classifiers such as, Random Forest, Linear SVC, and Bernoulli's NB. To evaluate the performance of our models, we conducted a comprehensive analysis using both the balanced dataset and the unbalanced dataset. We then compared the resulting prediction outcomes to gain a clearer insight into which model performs more effectively.

1.3. SCOPE AND OBJECTIVE

1.3.1. SCOPE

The scope of our project is to deliver a comparative study on the detection of ransomware using feature selection and data balancing techniques and classification algorithms. This includes and is not limited to, collecting datasets which contain both goodware and ransomware instances, and training classification models for the detection of ransomware.

1.3.2. OBJECTIVE

The objectives of this project are the following:

- Conduct a comparative analysis on the detection of ransomware.
- Explore the impact of various factors and parameters on the performance of the models.
- Evaluate the effectiveness of the proposed methodology on real-time data.

1.4. ORGANIZATION STRUCTURE OF THE REPORT

Section II of this report will provide an overview of the literature review and background research that were conducted in preparation for this project. The Proposed Work will be outlined in Section III, while Section IV will present the results and experimental analysis. Finally, in Section V, a summary of the project will be provided, along with suggestions for future research.

2. LITERATURE SURVEY

In this section, we will discuss the related work in the areas of feature selection and data balancing algorithms used for selecting a classifier for detecting ransomware.

Ransomware Detection

J. Chittooparambil et al.[1] provided a comprehensive review of different ransomware families and their characteristics, along with an overview of various ransomware detection techniques, including machine learning-based approaches.

Kharaz et al. [2] proposed a large-scale, automated approach to detecting ransomware based on behavioral analysis of malware, such as file access patterns and system calls. The approach achieved high detection rates and low false positives.

T. A. Alhawawreh et al. [3] proposed a machine learning-based approach for ransomware detection that uses system call sequences as features. The approach achieved high detection rates with low false positives.

S. Mahmud et al. [4] provided a comprehensive review of machine learning-based ransomware detection approaches, including their strengths and limitations. The authors also identify research gaps and provide future research directions.

S. S. S. S. and S. S et al. [5] proposed a machine learning-based approach that uses behavior-based features extracted from system calls and system events for ransomware detection. The approach achieved high detection rates with low false positives.

A. R. Al-Attiyat et al. [6] proposed a machine learning-based approach that uses API calls and their parameters as features for ransomware detection. The approach achieved high detection rates and low false positives.

R. C. L. P. C. de Lima et al [7] provides a systematic literature review of machine learning-based ransomware detection approaches, including their evaluation metrics and datasets. The authors also identify research challenges and future research directions.

B. Feature Selection

S. Sathya and P. Thangaraj et al. [8] evaluates different feature selection techniques, including VIF, for intrusion detection using machine learning.

V. Gayathri and V. Saravanan et al. [9] proposed a feature selection method based on VIF and mutual information for classifying cancer data using machine learning.

H. Yang, J. Zhang, and S. Liu et al. [10] reviews various feature selection techniques, including VIF, for bioinformatics applications using machine learning.

C. Data Balancing

D. K. Mahmood and N. A. Othman et al. [11] evaluates the performance of the KNN classifier with SMOTE and Tomek links for detecting intrusion in a network. The authors found that using SMOTE and Tomek links together improved the accuracy of the KNN classifier.

R. R. Nair and P. R. Nair et al. [12] compares the effectiveness of SMOTE and Tomek links for handling data imbalance in different classification algorithms. The authors found that SMOTE and Tomek links both improved the performance of classifiers, but SMOTE was more effective for minority class oversampling, while Tomek links were more effective for majority class undersampling.

G. Batista, R. C. Prati, and M. C. Monard et al. [13] proposed a new hybrid technique that combines SMOTE and Tomek links for binary classification problems. The authors found that SMOTE-Tomek links outperformed both SMOTE and Tomek links individually for several classification algorithms.

S. H. Khan and M. G. Madden et al. [14] compares the effectiveness of several sampling techniques, including SMOTE and Tomek links, for handling imbalanced data. The authors found that SMOTE and Tomek links together provided the best results in terms of improving the accuracy and F1 score of classifiers.

G. Ramya and K. R. Raja et al. [15] evaluates the performance of the SVM classifier with SMOTE and Tomek links for imbalanced data. The authors found that using SMOTE and Tomek links together improved the performance of SVM, particularly for the minority class.

3. PROPOSED METHODOLOGY

Our proposed method consists of four parts –

- Data Preprocessing
- Feature Selection
- Data Balancing and
- The Classification models.

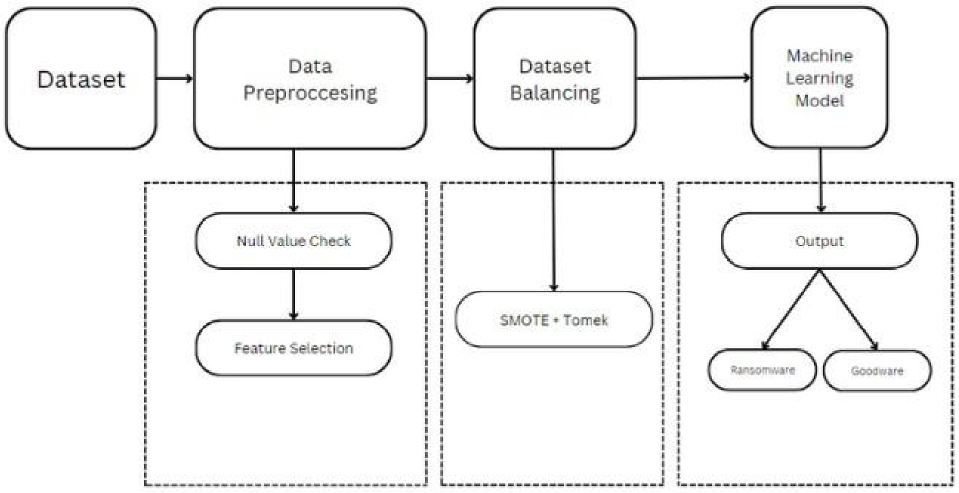


Figure 1. Proposed Model Architecture with Data Balancing

This figure depicts the proposed model architecture with data balancing which consists of three steps - i) Data Preprocessing ii) Dataset Balancing iii) Applying ML Model

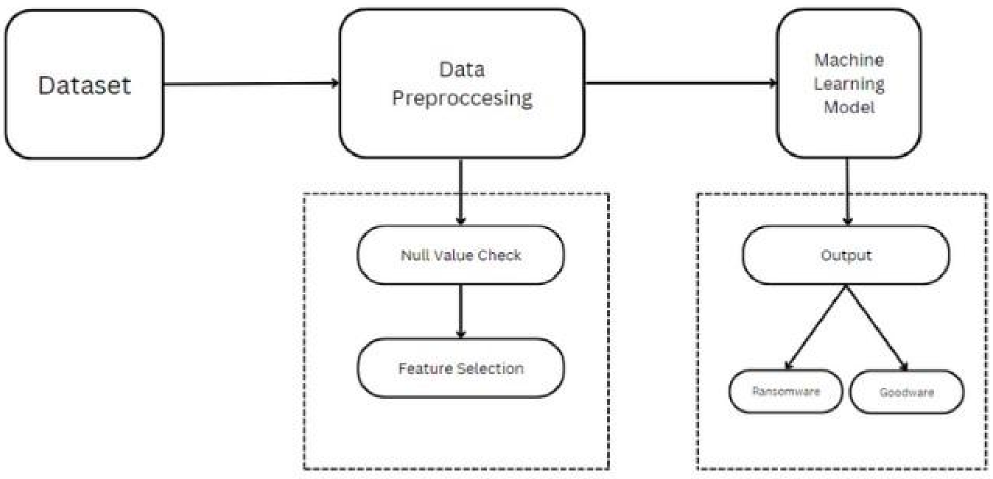


Figure 2. Proposed Model Architecture without Data Balancing

This figure depicts the proposed model architecture with data balancing which consists of two steps - i) Data Preprocessing ii) Applying ML Model

3.1. DATA PREPROCESSING

A single dataset was utilized, which was obtained from Kaggle and contained 138,047 instances with 96,724 being ransomware instances and 41,323 being goodware instances. The instances were composed of 57 features including Name, md5, Machine, and others, and contained information on various file extensions such as .exe, .dll, among others. The dataset underwent preprocessing which involved checking for null values, analyzing columns, conducting exploratory data analysis, and making the dataset ready for the Feature Selection step. In our analysis, we focused on the Ransomware dataset, which contains instances of ransomware and their associated class types, such as goodware or ransomware. Our first step was to ensure that there were no null values in the dataset, and then we conducted a thorough column analysis to gain a better understanding of the data. After that, we performed an exploratory data analysis and discovered that several features within the dataset were highly correlated, indicating that they may be redundant or providing similar information.

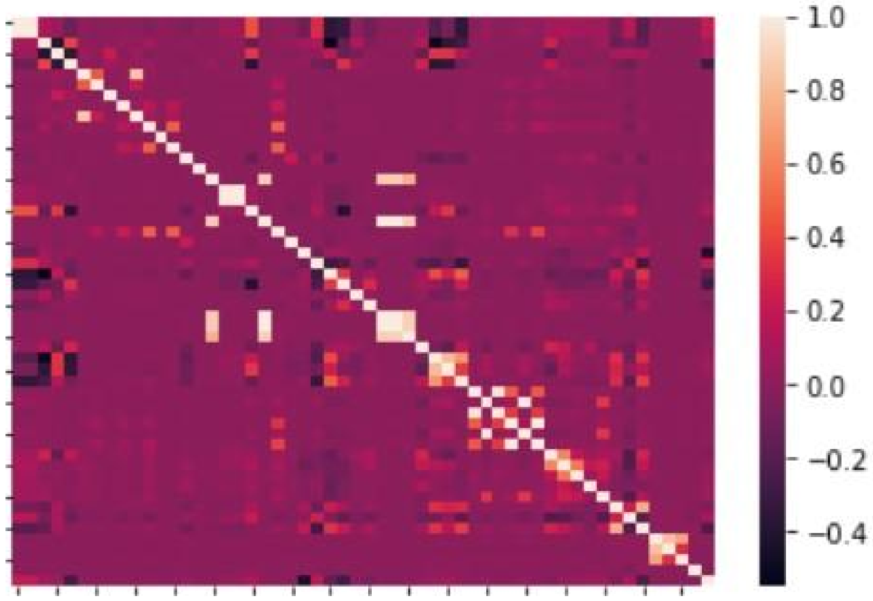


Figure 3. Before Feature Selection

This figure depicts the visual representation of the multicollinearity among features in the dataset before feature selection step

3.2. FEATURE SELECTION

Reducing dimensionality is crucial in the quantitative analysis of high-dimensional data, and feature selection is a way to achieve this. To improve the classification of file instances, we utilized the Variance Inflation Factor (VIF) algorithm, which addresses multicollinearity among features. In linear models, multicollinearity can be a significant issue as it can lead to unstable parameter estimation, unreliable models, and limited predictive ability.

To overcome this challenge, the variance inflation factor (VIF) was introduced as a tool for feature selection. Its purpose is to identify and address collinearity among predictor variables, which can help improve the accuracy and stability of the model. Multicollinearity can affect the reliability of the regression results by inflating the variance of the regression coefficient. The degree of inflation can be measured using a metric called Variance Inflation Factor (VIF). By regressing a variable against every other variable, we can determine the strength of their correlation and identify highly correlated features to be removed.

$$\mathbf{VIF} = \frac{\mathbf{1}}{\mathbf{1} - \mathbf{R_i^2}}$$

The R^2 value is used to assess how well one independent variable is explained by the other independent variables. When the R^2 value is high, it indicates a strong correlation between the variables. A high R^2 value corresponds to a high VIF value, which means that there is a high degree of multicollinearity between the independent variable and the other variables.

A VIF value of 1 indicates no correlation, while a VIF value greater than 5 indicates high correlation. A VIF value greater than 10 is a cause for concern, as it indicates a high degree of correlation and may result in unreliable regression results. In order to address multicollinearity in our regression analysis, we employed the Variance Inflation Factor (VIF) algorithm, which is commonly used to identify features with high levels of correlation. We calculated the VIF values for all of the features in our dataset, recognizing that as the VIF value increases, the reliability of the regression results decreases. Based on this analysis, we removed the features with high VIF values.

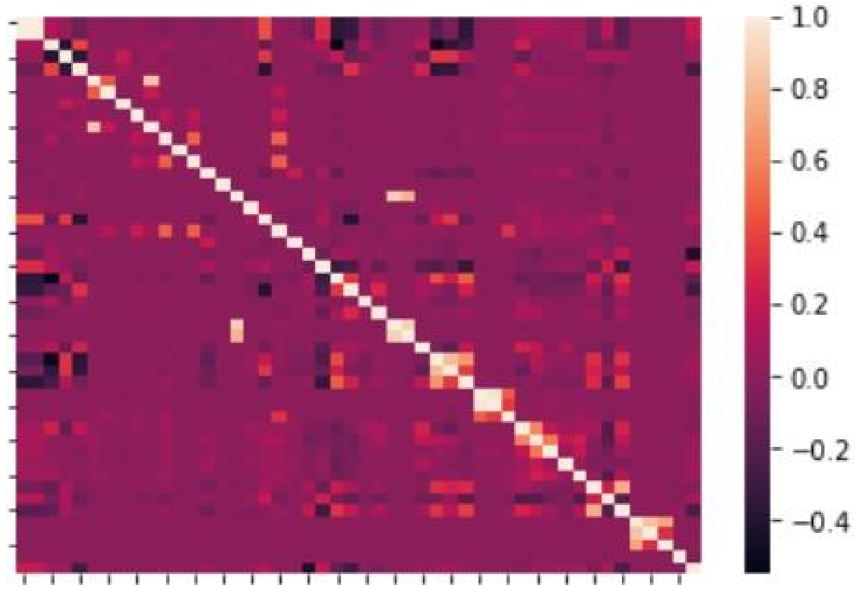


Figure 4. After Feature Selection

This figure depicts the visual representation of the multicollinearity among features in the dataset after feature selection step

3.3. DATA BALANCING

In real-world scenarios, classification models frequently encounter imbalanced datasets where the majority class has significantly more instances than the minority class. This can impede the model's ability to learn from the minority class, which may contain essential information, particularly if it's more important for the task.

Random Oversampling is a machine learning data augmentation method that tackles class imbalance by randomly duplicating samples from the minority class to generate additional synthetic samples. This results in an equal or nearly equal number of instances for each class, which can effectively improve the performance of the model.

Downsampling involves reducing the number of samples in the class with more data to match the class with the least data, while upsampling involves adding more data to the class with the least data to match the class with more data. Hybridization techniques combine both undersampling and oversampling techniques to optimize classifier model performance on the resulting samples.There exist various techniques to perform this process, however, this paper focuses solely on SMOTE and Tomek Links.

In machine learning, SMOTE, which stands for Synthetic Minority Over-sampling Technique, is a frequently utilized data augmentation method that is employed to tackle the problem of class imbalance. SMOTE creates synthetic samples of the minority class by synthesizing new samples using existing data points. A new synthetic sample is created by the algorithm using feature values of the minority class sample. To do this, the difference between the feature values of the minority class sample and its nearest neighbor is first calculated. This difference is then multiplied by a random number between 0 and 1 and added to the feature values of the minority class sample.

Tomek Links is a modified version of the Condensed Nearest Neighbors (CNN) undersampling technique that was developed by Tomek in 1976. Unlike the CNN method, which randomly selects samples from the majority class and their k nearest neighbors to be removed, the Tomek Links method applies a specific rule to select pairs of observations (such as a and b) that meet the following criteria:

Observation a has b as its nearest neighbor.

Observation b has a as its nearest neighbor.

Observation a and b are assigned to different categories, where a belongs to the group with a smaller number of instances, commonly referred to as the minority class, and b belongs to the group with a larger number of instances, often referred to as the majority class

SMOTE-Tomek is a technique used to tackle class imbalance in a dataset. It combines two algorithms, SMOTE (Synthetic Minority Over-sampling Technique) and Tomek links. SMOTE is an oversampling technique while Tomek links is an undersampling technique. The SMOTE-Tomek algorithm first applies Tomek links to remove unclear examples, and then uses SMOTE to create synthetic examples of the minority class. The outcome is a dataset with a balanced class distribution that is less influenced by unclear examples.

Instances before applying SMOTE+Tomek Links are 138047 whereas instances after applying SMOTE+Tomek Links are 175080. We used both the datasets to apply various machine learning models and compared the results.

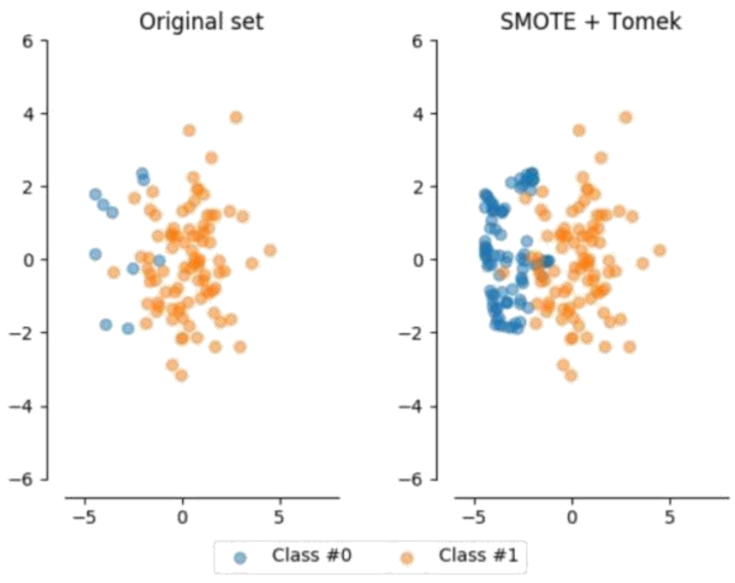


Figure 5. An illustration of the SMOTE + Tomek method

This figure depicts the graphical representation of Synthetic Minority Over-sampling Technique and Tomek Links. Left side of the image shows the original imbalanced set whereas right side of the image shows the balanced set

3.4. CLASSIFICATION ALGORITHMS

We compared the effectiveness of three classification algorithms: LinearSVC, Bernoulli Naive Bayes, and Random Forest.

LinearSVC (Support Vector Classification) is a type of linear model for classification tasks. It works by finding a hyperplane in a high-dimensional space that separates the classes as best as possible. The hyperplane is chosen to maximize the margin between the classes, which is the distance between the hyperplane and the closest points of each class. LinearSVC is a popular algorithm due to its efficiency and effectiveness in high-dimensional settings. It can handle large datasets and is well-suited for problems with many features. Additionally, it allows for the use of different kernel functions to handle non-linear separable data. SVMs are a set of powerful machine learning algorithms that are capable of handling large and complex datasets. LinearSVC is particularly useful in situations where the number of features is much larger than the number of samples. It works by finding the hyperplane that maximally separates the two classes while also minimizing the classification error. This is done by solving a constrained optimization problem, which ensures that the hyperplane is as far away as possible from the closest data points of each class. One of the key advantages of LinearSVC is its ability to work well with high-dimensional feature spaces. It can handle data with many features and large numbers of samples. Moreover, LinearSVC is computationally efficient, making it suitable for large-scale applications. It can also be used with different kernel functions to handle non-linearly separable data. Overall, LinearSVC is a powerful machine learning algorithm that is widely used in various applications such as text classification, image classification, and bioinformatics.

Bernoulli Naive Bayes (Bernoulli NB) is a classification algorithm that is particularly useful for binary classification problems, where each feature can only take on one of two possible values. It operates by calculating the probability of each feature in the input belonging to each possible class and then combining these probabilities to predict the most likely class. Bernoulli NB is based on the assumption that the features are independent of one another, which is often not the case in practice, but can still provide reasonable results. It is computationally efficient and can handle large datasets with high-dimensional feature spaces, making it a popular choice for text classification tasks. Bernoulli Naive Bayes is a variant of the Naive Bayes algorithm, which is based on the assumption that the features are conditionally independent given the class. This assumption may not hold in practice, but in many cases, the algorithm still performs well. Bernoulli NB estimates the probabilities of each feature being present or absent in the input given each class, rather than estimating the probability of the feature values themselves. One of the key advantages of Bernoulli NB is its simplicity and efficiency. It requires minimal training time and memory and can handle high-dimensional feature spaces. Moreover, Bernoulli NB is robust to missing data, making it useful for incomplete datasets. Despite its simplicity, Bernoulli NB has been shown to perform well in various real-world applications. It has been used in natural language processing tasks, such as text classification, sentiment analysis, and spam detection. It has also been used in image classification tasks, such as face recognition and object recognition. Overall, Bernoulli NB is a powerful and widely used algorithm in machine learning, particularly in applications that involve binary feature variables.

Random Forest is an ensemble learning algorithm that combines multiple decision trees trained on random subsets of the data to create a more accurate and robust model. This approach helps to reduce the correlation between the trees and prevent overfitting. The algorithm has become popular due to its versatility and performance across a wide range of applications, including classification, regression, and feature selection.

4. EXPERIMENTAL STUDY

4.1. DATASETS

We used the ransomware dataset as our main training dataset. This dataset can be found on Kaggle and it contains 1,38,047 instances, out of which 96724 are ransomware instances and 41323 goodware instances. This dataset contains details of various files of different extensions such as .exe, .dll, etc. Each instance has 57 columns such as Name, md5, Machine etc. The dataset has 56 independent variables and a dependent variable named as legitimate which tells whether a particular instance is ransomware or not.

4.2. SOFTWARE REQUIREMENTS

Software Requirements refer to the software required to run the detection system. Our detection system requires the following software installed:

- Jupyter Notebook/Google Collab
- Python 3 and its libraries (Tensorflow, Pandas, NumPy)

4.3. HARDWARE REQUIREMENTS

Hardware Requirements refer to the minimum hardware qualifications needed to run the detection system smoothly. Our detection system requires the following specifications:

- Intel i5 Processor or better version
- 8+ GB RAM
- Windows OS recommended, but can use Linux/MacOS

4.4. PARAMETER SETTING

4.4.1. FEATURE SELECTION

We calculate the VIF for each feature using the variance_inflation_factor function from the statsmodels.stats.outliers_influence module. This function takes two arguments - the array of predictor variable values and the index of the feature for which VIF is being calculated. The VIF value is calculated for each feature using a loop that iterates through all the features in the cols_vif list.

Finally, the results are printed as a DataFrame with the feature names and their corresponding VIF values in the vif_data DataFrame. This output can be used to identify which predictor variables may be too similar to each other and should be removed from the model to improve

4.4.2. DATA BALANCING

Using the ‘imblearn’ package, we applied the resampling hybridization technique. First, an instance of SMOTETomek is created with a sampling_strategy of 1.0, which means that it will balance the number of samples in both classes. The n_jobs parameter is set to -1, which allows the algorithm to use all available processors for parallel computation, and the random_state parameter is set to randomseed to ensure that the results are reproducible.

Next, the fit_resample() method of the SMOTETomek instance is called on the X_train and y_train datasets. This method performs the SMOTE and Tomek links combination resampling technique on the input data to create a balanced dataset. Specifically, it first applies the Tomek links algorithm to remove ambiguous examples, and then uses SMOTE to generate synthetic examples of the minority class. The result is a new X_train and y_train dataset with a balanced class distribution that is less affected by ambiguous examples. The new balanced X_train and y_train dataset can then be used to train a machine learning model that can make more accurate predictions on the minority class.

4.4.3. CLASSIFICATION ALGORITHMS

For our classification models, we utilized the ‘sklearn’ library for implementation. Firstly, we employed LinearSVC model for both imbalanced and balanced datasets.

Next, we employed the Bernoulli Naive Bayes algorithm. The Bernoulli Naive Bayes algorithm is implemented on the dataset by first splitting the data into training and testing sets. Then, the BernoulliNB class from the sklearn.naive_bayes module is used to create an instance of the model, which is trained on the training data using the fit method. Finally, the trained model is used to make predictions on the testing data using the predict method, and the accuracy of the predictions is evaluated using the accuracy_score function from the sklearn.metrics module.

Next, we employed the Random Forest algorithm. An instance of the RandomForestClassifier class from sklearn.ensemble is created with the desired hyperparameters, such as the number of trees in the forest. The model is trained on the training data using the fit method of the RandomForestClassifier class. The trained model is used to make predictions on the testing data using the predict method of the RandomForestClassifier class.

We then compared the results of all three models and made a comparative study on them.

4.5. RESULTS

This section will describe the results which we obtained from the comparative analysis and briefly explain the metrics of evaluation we used.

4.5.1. METRICS OF EVALUATION

In order to evaluate the performance of our model effectively, we used three different metrics of evaluation – Accuracy, TPR, FPR, Precision, Recall, F-Measure.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}$$

In the context of evaluating a classification model, accuracy is a measure that determines how well the model is performing. It quantifies the proportion of instances that are classified correctly out of the total number of instances. The measure is calculated using the values of True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). TP refers to the number of instances that are correctly classified as positive, TN is the number of instances that are correctly classified as negative, FP denotes the number of negative instances that are incorrectly classified as positive, and FN is the number of positive instances that are incorrectly classified as negative. Specifically, in the case of ransomware detection, TP represents the number of correctly identified ransomware files, TN represents the number of correctly identified goodware files, FP represents the number of misclassified goodware files as ransomware, and FN represents the number of misclassified ransomware files as goodware.

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

True Positive Rate (TPR) tells us what proportion of the positive (malicious) class got correctly classified. It measures how relevant the results are and how well the model distinguished between the classes.

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

False Positive Rate (FPR) tells us what proportion of the negative (benign) class got incorrectly classified. It measures how well the model distinguished between the classes.

$$\mathrm{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

Precision is a metric used in binary classification problems that measures the proportion of correctly predicted positive instances (true positives) out of all instances that the model predicted as positive, whether they are actually positive or negative. In other words, precision is a measure of how precise or exact the model's positive predictions are. A high precision score indicates that the model has a low false positive rate and is able to accurately identify positive instances.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Recall is a metric used in binary classification problems that measures the proportion of correctly predicted positive instances (true positives) out of all actual positive instances, whether they were predicted as positive or negative by the model. In other words, recall is a measure of how many actual positive instances the model is able to identify. A high recall score indicates that the model has a low false negative rate and is able to correctly identify a high proportion of positive instances.

$$\text{F} - \text{Measure} = 2 * \frac{(\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})}$$

F-measure (also known as F1 score) is a metric used in binary classification problems that combines both precision and recall into a single measure. It is the harmonic mean of precision and recall, and it provides a balanced measure of the model's performance on both positive and negative instances. A high F-measure score indicates that the model has both high precision and high recall, meaning that it is able to accurately identify positive instances while avoiding false positives and false negatives.

4.5.2. OBSERVATIONS

This section will highlight the results achieved on the Malicious-URLs dataset based on Accuracy, TPR and FPR and the HTTP Traffic based on Accuracy.

Model	ACC	TPR	FPR
Linear SVC	0.968	0.932	0.015
BernoulliNB	0.940	0.822	0.008
Random Forest	0.994	0.993	0.004

Table 1. Performance of Ransomware dataset without data balancing

Model	ACC	TPR	FPR
Linear SVC	0.967	0.957	0.028
BernoulliNB	0.945	0.847	0.010
Random Forest	0.995	0.996	0.005

Table 2. Performance of Ransomware dataset with data balancing

5. CONCLUSION AND FUTURE SCOPE

In this article, we proposed a comparative analysis between various classification models. We used feature selection and data balancing techniques and observed their effects on different classification models, for the detection of ransomware. We employed Variance inflation factor technique for feature selection and eliminated the unnecessary features. The major advantage it played was that they reduced the training complexity classification algorithms successfully. Next, we applied the resampling technique, SMOTE and Tomek Links, to rebalance the imbalanced datasets and compared the accuracies without balancing and with balancing on different machine learning models such as Linear SVC, Bernoulli Naive Bayes and Random Forest. From the experiments we conducted, we have the following findings. The Random Forest Classifer gave better results compared to Linear SVC and Bernoulli Naive Bayes. In the future, more oversampling and undersampling techniques can be explored to test on the classification algorithms and increase the performance of the model. It also includes eradication of ransomware after detection.

6. REFERENCES

[1] H.J. Chittooparambil, et al., A review of ransomware families and detection methods, in: International Conference of Reliable Information and Communication Technology, 2018, pp. 588–597.

[2] A. Kharaz, et al., {UNVEIL}: A large-scale, automated approach to detecting ransomware, in: 25th {USENIX} Security Symposium ({USENIX} Security 16), 2016, pp. 757–772.

[3] Detecting Ransomware with Machine Learning Techniques by F. T. A. Alhawawreh et al. (IEEE Access, 2018)

[4] Ransomware detection using machine learning algorithms: A review by M. S. Mahmud et al. (Computers & Security, 2021)

[5] Ransomware Detection: A Machine Learning Approach by S. S. S. S. and S. S. (2018 2nd International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS), 2018)

[6] Detecting Ransomware using Machine Learning Algorithms by A. R. Al-Attiyat et al. (International Journal of Advanced Computer Science and Applications, 2020)

[7] Ransomware Detection Based on Machine Learning Techniques: A Systematic Literature Review by R. C. L. P. C. de Lima et al. (IEEE Latin America Transactions, 2021)

[8] A comparative study on feature selection techniques for intrusion detection using machine learning by S. Sathya and P. Thangaraj from IEEE Xplore. This paper evaluates different feature selection techniques, including VIF, for intrusion detection using machine learning.

[9] Feature selection based on VIF and mutual information for classification of cancer data by V. Gayathri and V. Saravanan from ScienceDirect. This paper proposes a feature selection method based on VIF and mutual information for classifying cancer data using machine learning.

[10] A survey of feature selection techniques in bioinformatics by H. Yang, J. Zhang, and S. Liu from ScienceDirect. This paper reviews various feature selection techniques, including VIF, for bioinformatics applications using machine learning.

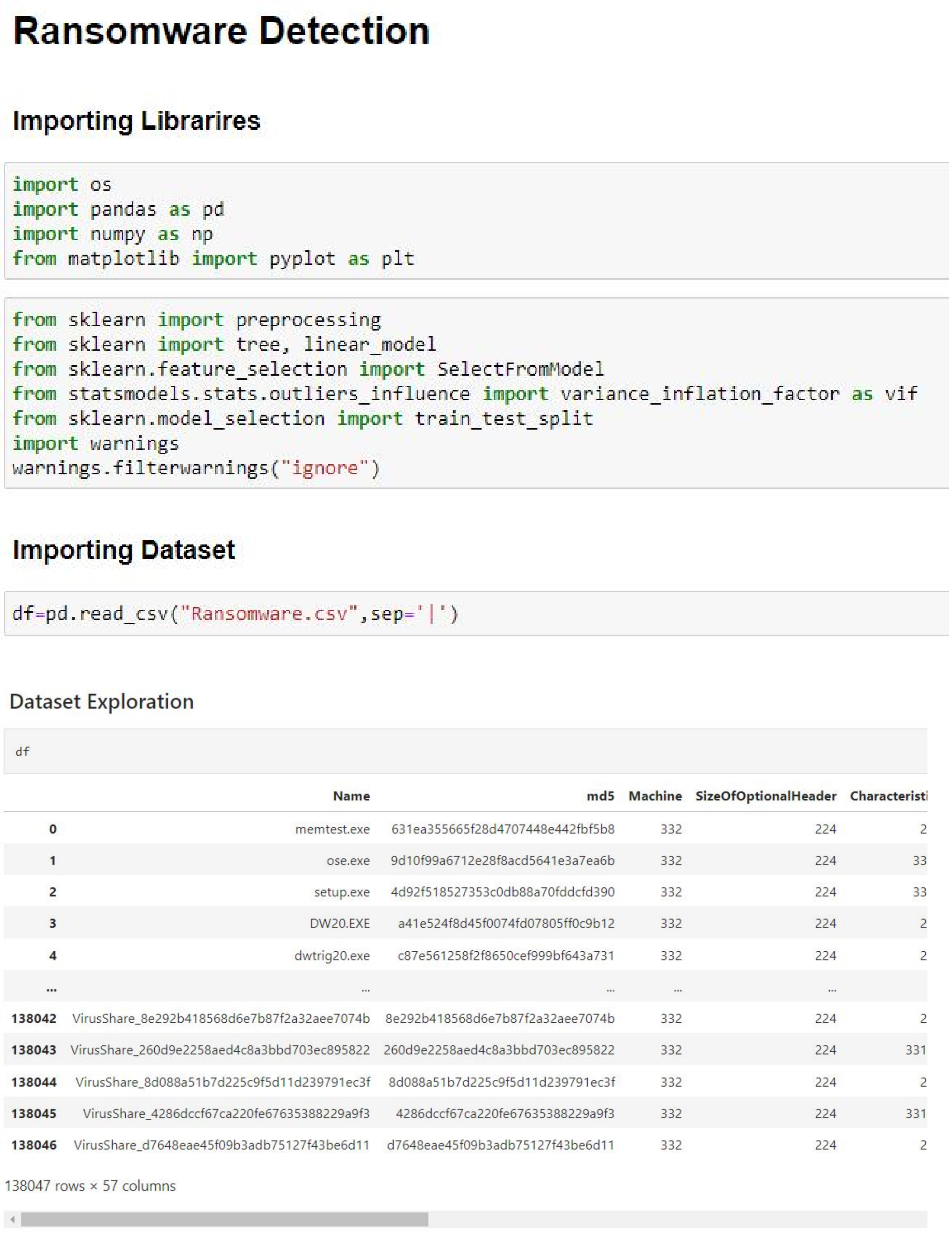
[11] The Effect of SMOTE and Tomek Links on Improving the Performance of KNN Classifier in Intrusion Detection System by D. K. Mahmood and N. A. Othman.
[12] A Comparative Study on the Effectiveness of SMOTE and Tomek Links in Handling Data Imbalance" by R. R. Nair and P. R. Nair.
[13] SMOTE-Tomek Links: A New Hybrid Technique for Binary Classification Problems by G. Batista, R. C. Prati, and M. C. Monard. This paper proposes a new hybrid technique that combines SMOTE and Tomek links for binary classification problems.
[14] A Comparative Study of Sampling Techniques for Imbalanced Data" by S. H. Khan and M. G. Madden.
[15] An Empirical Study of the Impact of SMOTE and Tomek Links on SVM for Imbalanced Data by G. Ramya and K. R. Raja.

APPENDIX

OPEN ACCESS LINK

GitHub : https://github.com/Nikki8502/RansomwareDetectionUsingML.git

CODE:



Ransomware Detection

Importing Librarires

```python
import os
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

```python
from sklearn import preprocessing
from sklearn import tree, linear_model
from sklearn.feature_selection import SelectFromModel
from statsmodels.stats.outliers_influence import variance_inflation_factor as vif
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")
```

Importing Dataset

```python
df=pd.read_csv("Ransomware.csv",sep='|')
```

Dataset Exploration

```python
df
```

```
                                             Name                               md5  Machine  SizeOfOptionalHeader  Characteristi
     0                                memtest.exe  631ea355665f28d4707448e442fbf5b8      332                   224              2
     1                                    ose.exe  9d10f99a6712e28f8acd5641e3a7ea6b      332                   224             33
     2                                  setup.exe  4d92f518527353c0db88a70fddcfd390      332                   224             33
     3                                   DW20.EXE  a41e524f8d45f0074fd07805ff0c9b12      332                   224              2
     4                               dwtrig20.exe  c87e561258f2f8650cef999bf643a731      332                   224              2
   ...                                        ...                               ...      ...                   ...
138042  VirusShare_8e292b418568d6e7b87f2a32aee7074b  8e292b418568d6e7b87f2a32aee7074b      332                   224              2
138043  VirusShare_260d9e2258aed4c8a3bbd703ec895822  260d9e2258aed4c8a3bbd703ec895822      332                   224            331
138044  VirusShare_8d088a51b7d225c9f5d11d239791ec3f  8d088a51b7d225c9f5d11d239791ec3f      332                   224              2
138045  VirusShare_4286dccf67ca220fe67635388229a9f3  4286dccf67ca220fe67635388229a9f3      332                   224            331
138046  VirusShare_d7648eae45f09b3adb75127f43be6d11  d7648eae45f09b3adb75127f43be6d11      332                   224              2

138047 rows × 57 columns
```

```
df.describe()
```

	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	MinorLinkerVersion	SizeOfC
count	138047.000000	138047.000000	138047.000000	138047.000000	138047.000000	1.380470e
mean	4259.069274	225.845632	4444.145994	8.619774	3.819286	2.425956e
std	10880.347245	5.121399	8186.782524	4.088757	11.862675	5.754485e
min	332.000000	224.000000	2.000000	0.000000	0.000000	0.000000e
25%	332.000000	224.000000	258.000000	8.000000	0.000000	3.020800e
50%	332.000000	224.000000	258.000000	9.000000	0.000000	1.136640e
75%	332.000000	224.000000	8226.000000	10.000000	0.000000	1.203200e
max	34404.000000	352.000000	49551.000000	255.000000	255.000000	1.818587e

8 rows × 55 columns

```
# 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: 0.0798260634765029 GB
```

Null value check

```
df.isnull().sum()
```

```
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
ResourcesMaxEntropy          0
ResourcesMeanSize            0
ResourcesMinSize             0
ResourcesMaxSize             0
LoadConfigurationSize        0
VersionInformationSize       0
legitimate                   0
dtype: int64
```

Distribution of Labelled Data

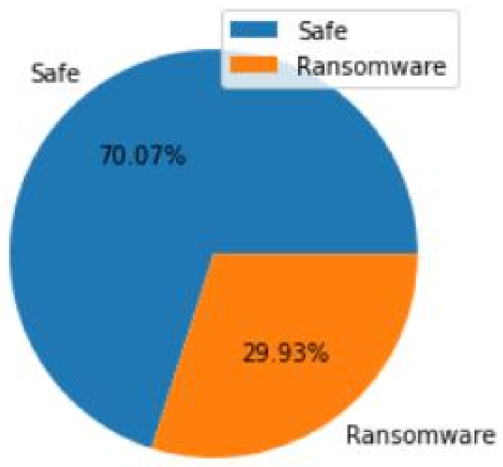
```
df.legitimate.value_counts() #1 means legitimate, 0 means malware
```

```
0    96724
1    41323
Name: legitimate, dtype: int64
```

```
# Converting labelled data in categories datatype
df.legitimate = df.legitimate.astype('category')
df.legitimate
```

```
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]
```

```
plt.pie(df.legitimate.value_counts().values.tolist(), labels=['Safe','Ransomware'], autopct='%.2f%%')
plt.legend()
plt.show()
```



Unique names

```
df.md5.nunique()
```

```
138047
```

```
df.md5.shape[0]
```

```
138047
```

```
# There are no same files as no 2 files can have same md5 without the same content
```

Column Analysis

```
df.shape[1]
```

```
57
```

```
df.columns
```

```
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',
       'SectionsMeanEntropy', 'SectionsMinEntropy', 'SectionsMaxEntropy',
       'SectionsMeanRawsize', 'SectionsMinRawsize', 'SectionMaxRawsize',
       'SectionsMeanVirtualsize', 'SectionsMinVirtualsize',
       'SectionMaxVirtualsize', 'ImportsNbDLL', 'ImportsNb',
       'ImportsNbOrdinal', 'ExportNb', 'ResourcesNb', 'ResourcesMeanEntropy',
       'ResourcesMinEntropy', 'ResourcesMaxEntropy', 'ResourcesMeanSize',
       'ResourcesMinSize', 'ResourcesMaxSize', 'LoadConfigurationSize',
       'VersionInformationSize', 'legitimate'],
      dtype='object')
```

```
df.dtypes
```

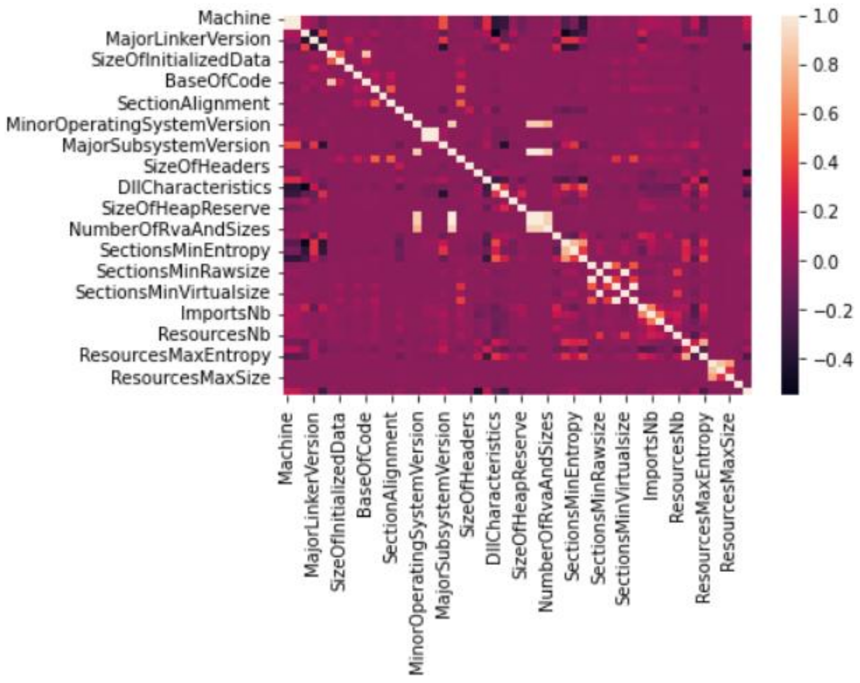
```
Name                            object
md5                             object
Machine                          int64
SizeOfOptionalHeader             int64
Characteristics                  int64
MajorLinkerVersion               int64
MinorLinkerVersion               int64
SizeOfCode                       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
ResourcesMaxSize                 int64
LoadConfigurationSize            int64
VersionInformationSize           int64
legitimate                    category
dtype: object
```

EDA

```
sns.heatmap(df.corr())
```

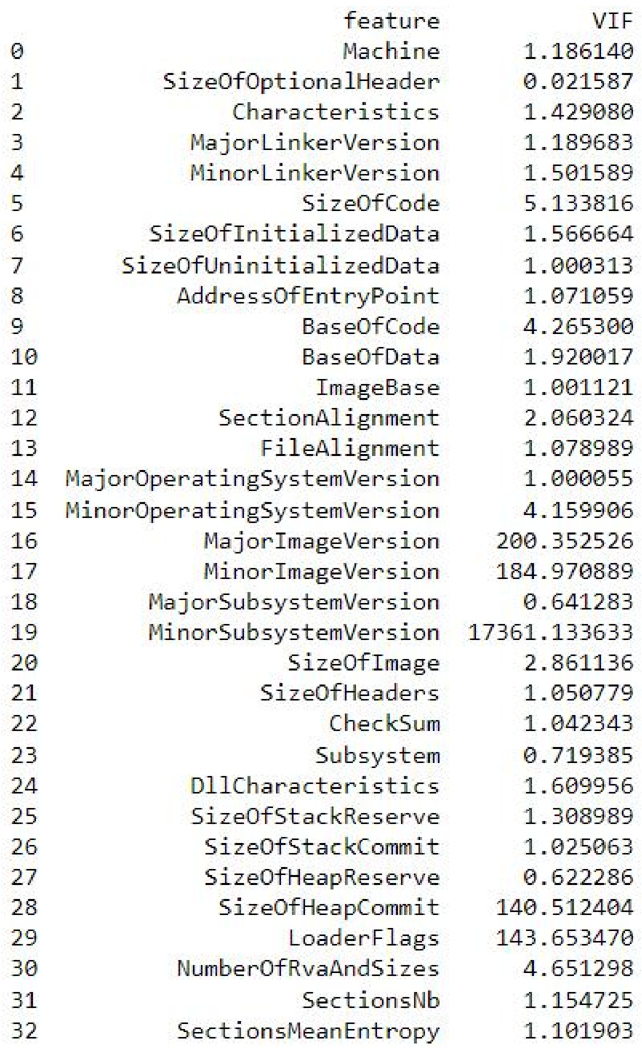
```
<AxesSubplot:>
```



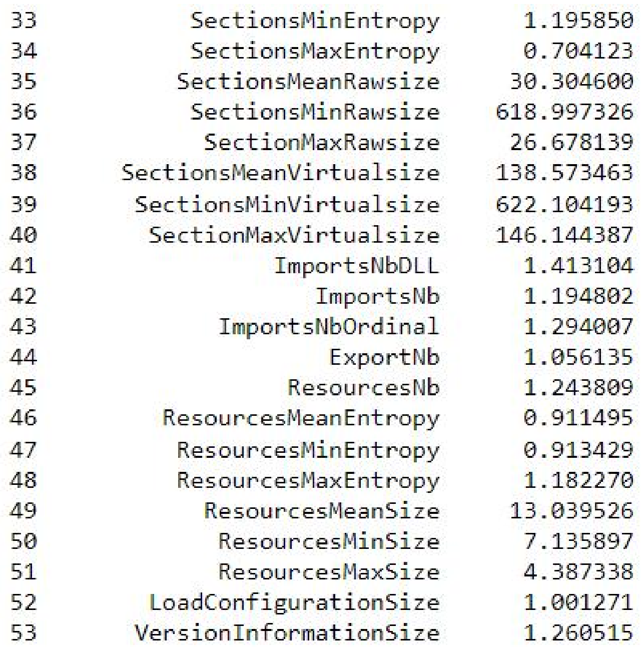
Feature Selection

Variance Inflation Factor

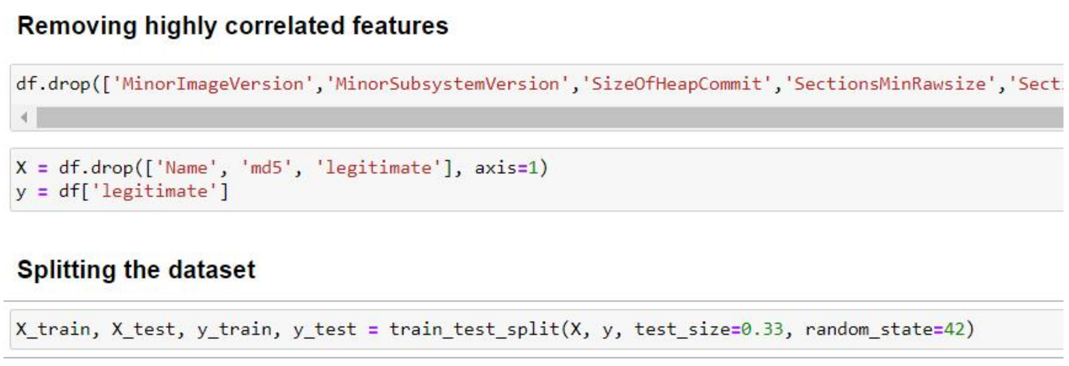
```
from statsmodels.stats.outliers_influence import variance_inflation_factor


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

# 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)
```



```
                          feature           VIF
0                         Machine      1.186140
1            SizeOfOptionalHeader      0.021587
2                 Characteristics      1.429080
3              MajorLinkerVersion      1.189683
4              MinorLinkerVersion      1.501589
5                      SizeOfCode      5.133816
6           SizeOfInitializedData      1.566664
7         SizeOfUninitializedData      1.000313
8             AddressOfEntryPoint      1.071059
9                      BaseOfCode      4.265300
10                     BaseOfData      1.920017
11                      ImageBase      1.001121
12               SectionAlignment      2.060324
13                  FileAlignment      1.078989
14    MajorOperatingSystemVersion      1.000055
15    MinorOperatingSystemVersion      4.159906
16              MajorImageVersion    200.352526
17              MinorImageVersion    184.970889
18          MajorSubsystemVersion      0.641283
19          MinorSubsystemVersion  17361.133633
20                    SizeOfImage      2.861136
21                  SizeOfHeaders      1.050779
22                       CheckSum      1.042343
23                      Subsystem      0.719385
24             DllCharacteristics      1.609956
25             SizeOfStackReserve      1.308989
26              SizeOfStackCommit      1.025063
27              SizeOfHeapReserve      0.622286
28               SizeOfHeapCommit    140.512404
29                    LoaderFlags    143.653470
30            NumberOfRvaAndSizes      4.651298
31                     SectionsNb      1.154725
32            SectionsMeanEntropy      1.101903
```



```
33             SectionsMinEntropy      1.195850
34             SectionsMaxEntropy      0.704123
35            SectionsMeanRawsize     30.304600
36             SectionsMinRawsize    618.997326
37              SectionMaxRawsize     26.678139
38        SectionsMeanVirtualsize    138.573463
39         SectionsMinVirtualsize    622.104193
40          SectionMaxVirtualsize    146.144387
41                   ImportsNbDLL      1.413104
42                      ImportsNb      1.194802
43               ImportsNbOrdinal      1.294007
44                       ExportNb      1.056135
45                    ResourcesNb      1.243809
46           ResourcesMeanEntropy      0.911495
47            ResourcesMinEntropy      0.913429
48            ResourcesMaxEntropy      1.182270
49              ResourcesMeanSize     13.039526
50               ResourcesMinSize      7.135897
51               ResourcesMaxSize      4.387338
52          LoadConfigurationSize      1.001271
53           VersionInformationSize      1.260515
```



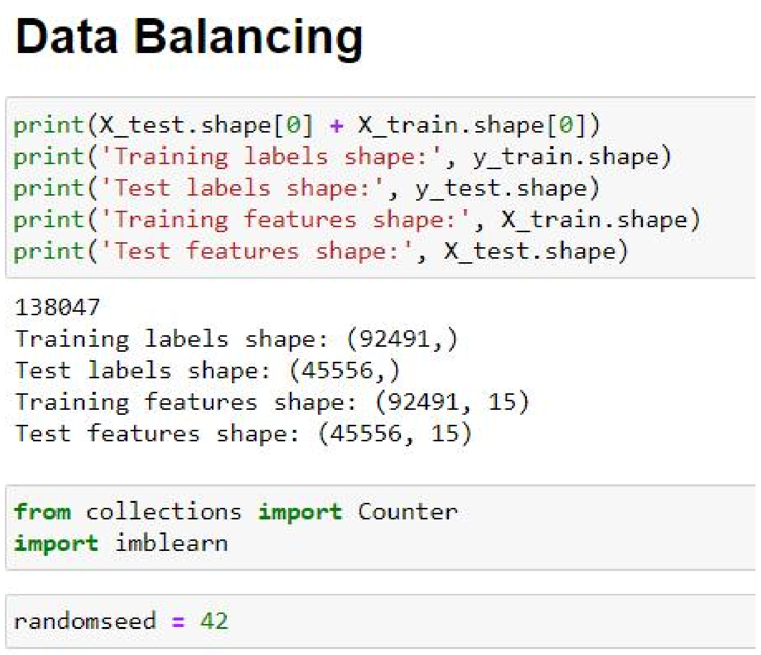
Removing highly correlated features

```
df.drop(['MinorImageVersion','MinorSubsystemVersion','SizeOfHeapCommit','SectionsMinRawsize','Sect
```

```
X = df.drop(['Name', 'md5', 'legitimate'], axis=1)
y = df['legitimate']
```

Splitting the dataset

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```



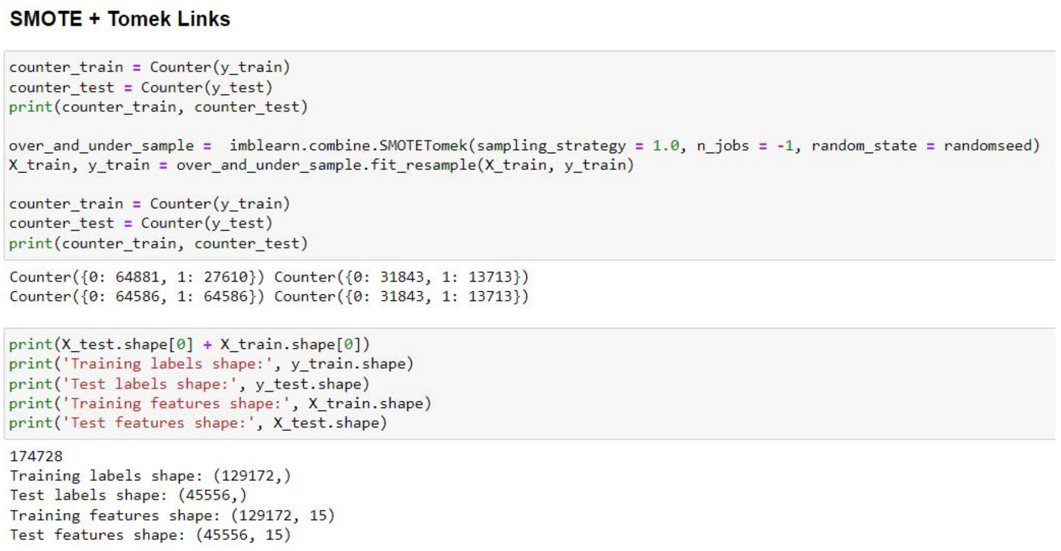
Data Balancing

```
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)
```

```
from collections import Counter
import imblearn
```

```
randomseed = 42
```



SMOTE + Tomek Links

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

over_and_under_sample =  imblearn.combine.SMOTETomek(sampling_strategy = 1.0, n_jobs = -1, random_state = randomseed)
X_train, y_train = over_and_under_sample.fit_resample(X_train, y_train)

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: 64586, 1: 64586}) Counter({0: 31843, 1: 13713})
```

```
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)
```

```
174728
Training labels shape: (129172,)
Test labels shape: (45556,)
Training features shape: (129172, 15)
Test features shape: (45556, 15)
```

BernoulliNB Model

```
from sklearn.naive_bayes import BernoulliNB
model = BernoulliNB(alpha=1.0, binarize=0.0)

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
pred = model.predict(X_test)
```

LinearSVC Model

```
from sklearn.svm import LinearSVC
model = LinearSVC()
# Train the model on the training data
model.fit(X_train, y_train)
# Make predictions on the test data
pred = model.predict(X_test)
```

Random Forest Model

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(random_state = randomseed)

# Train the model on the training data
rf.fit(X_train,y_train)

# Make predictions on the test data
pred = rf.predict(X_test)
```

Evaluation Metrics

```
from sklearn.metrics import accuracy_score
```

```
print(accuracy_score(y_test, pred))
```

```
from sklearn.metrics import confusion_matrix, classification_report
```

```
print('Confusion Matrix:')
print(confusion_matrix(y_test, pred))
print('\nClassification Report:')
print(classification_report(y_test, pred))
```

```
from sklearn.metrics import precision_score, recall_score, f1_score
```

```
precision = precision_score(y_test, pred)
recall = recall_score(y_test, pred)
f1 = f1_score(y_test, pred)
```

```
print('Precision: {:.3f}'.format(precision))
print('Recall: {:.3f}'.format(recall))
print('F1 score: {:.3f}'.format(f1))
```

```
tn, fp, fn, tp = confusion_matrix(y_test, pred).ravel()
```

```
tpr = tp / (tp + fn)
fpr = fp / (fp + tn)
print('True Positive Rate (Recall): {:.3f}'.format(tpr))
print('False Positive Rate: {:.3f}'.format(fpr))
```