Malware Analysis with Machine Learning

Carlven Tan [1], Hu Ng [1], Timothy Tzen Vun Yap [1]

[1] Faculty of Computing and Informatics, Multimedia University, 63100 Cyberjaya, Malaysia

Abstract. Malware analysis is crucial in cybersecurity for identifying and eliminating malicious software. Traditional methods have proven to be time-consuming and resource intensive. This research project aimed to leverage machine learning techniques to enhance the efficiency and accuracy of malware analysis. A malware detection system was developed, utilizing machine learning algorithms to classify malware based on their behavior and characteristics. The system was trained on a diverse dataset comprising various malware families and evaluated using appropriate metrics. The dataset was sourced from a publicly available repository on GitHub, provided by Marco Ramili. Prior to classification, several stages were completed, including exploratory data analysis, data preprocessing, data splitting, and feature selection. Notably, the machine learning algorithms employed were Decision Trees (DT), Random Forests (RF), and LightGBM. Four experiments were conducted, comparing different techniques across varying train-test split ratios. The results demonstrated that LightGBM outperformed other models consistently. For the 80-20 and 60-40 splits, LightGBM achieved accuracy rates of 90.59% and 89.55% respectively, using SMOTE without Boruta. Furthermore, LightGBM attained the highest accuracy of 90.33% for the 70-30 split, employing SMOTE and Boruta. The deployment of the malware detection system through a machine learning API will utilize LightGBM.

Keywords: Malware Detection, Machine Learning, Classification.

1. Introduction

Malware refers to any type of software or code that is intended to cause harm to a computer system or network. It can come in various forms to attack a computer system or network, such as spyware, adware, ransomware, trojan, worms, rootkits and backdoors. It can be used to accomplish tasks such as stealing protected data, installing unwanted software, or deleting confidential documents without authorization. Due to the widespread use of the internet, malware can spread quickly and target both individuals and organizations, making it a significant security threat. Cyber criminals are allowed to take advantage of stealing the data from the infected target over the internet (Chakkaravarthy, S. S., Sangeetha, D., & Vaidehi, V, 2019). Therefore, detecting and preventing malware is a top priority for many organizations and individuals.

In recent times, traditional malware detection systems have faced challenges in effectively detecting modern malware as attackers are constantly evolving their techniques and spreading their malicious software in large numbers. The prevalent method of malware detection, signature-based detection, involves identifying the signature of known malware in a database and removing it. However, this approach is not effectively as it struggles to detect polymorphic malware and zero-days attacks, with detection rate currently estimated to be between 25% and 50% currently (Masabo E, Kaawaase K S, Sansa-Otim J, 2018). This is because attackers can easily change the byte sequence within malware to avoid detection. Even after a new unknow threat is discovered, it is difficult to research and identify patterns in the malware as it is constantly changing. Additionally, malware cryptors can keep altering the malware until it becomes unrecognized to antivirus scans.

According to AMR, et al (2021), in the report of It Threat Evolution in Q2 2021, around one million new threats created were injected daily into the wild in 2021. In 2021, Kaspersky solutions prevented 1,686,025,551 attacks from online resources worldwide. They also reported web antivirus recognized 675,832,360 unique URL as malicious. Moreover, the attacker attempts to run malware for stealing money from online bank accounts were blocked on the computers of 119,252 unique users. Ransomware attacks were also detected and defeated on the computers of 97,451 unique users. A total of 68,294,298 unique malicious and potentially unwanted objects were detected by Kaspersky file antivirus.

To address the issue of malware, researchers have developed new method for accurately identifying and detecting it. These methods typically involve two important steps. The first step is analyzing the malware to extract information about its specific characteristics. This analysis can help to detect and mitigate any potential threat by understanding the malware's function, purposes and potential impact. There are two types of malware analysis that are commonly used, which are static analysis and dynamic analysis (Ijaz M, Durad M H, Ismail M, 2019). Static analysis examines samples without executing it, while dynamic analysis examines the execution of samples while they are running to observe their behaviours. After the samples have been analyzed, the information about their pattern and behaviour is used to identify the malware. In the second step, classification and detection systems are developed based on the information provided by malware analysis. These systems store signature patterns in their database and update them regularly (Singh J, Singh J., 2018). However, this approach may not be effective against polymorphic malware, which can change its identity at any time.

The current limitations of malware detection systems, such as their inability to detect modern and advanced forms of malware, have led to the need for alternative methods. Malware detection can be addressed by leveraging machine learning models as a potential solution. Further research is needed to identify the key features that best characterize the malware, as the efficiency of the employed machine learning techniques heavily relies on the careful selection of appropriate features. Feature selection, recognized as a critical stage in the creation of an effective machine learning model, is a fundamental process. With various machine learning algorithms available, making it challenging to identify the one that would be most suitable for a particular dataset and enhance the model's performance. In many cases, real-world datasets are complex and trying multiple algorithms may be necessary to achieve the desired results.

2. Literature Review

2.1. Malware Analysis

Malware analysis is a crucial aspect of computer security, as it aims to identify and classify new and unknown types of malwares to improve defences against them (Ucci D, Aniello L, Baldoni R, 2019). One of the main challenges in malware analysis is the ability to detect malicious software that has been created using obfuscation techniques such as polymorphism, metamorphism, and packing. These techniques make it difficult for traditional malware detection methods to identify and block malware before it causes harm (Singh, J., & Singh, J. 2018). To address this issue, the research community has developed various approaches for malware analysis, including both static and dynamic analysis techniques. Static analysis inspects the malware files in a safe environment without running them. Although this makes it more efficient than dynamic analysis, it still has drawbacks (Ijaz, Durad, and Ismail, 2019). For example, it prevents the reverse engineering of malware, which restricts the investigation of the source code. Dynamic analysis entails executing the malicious file and seeing how it interacts with the computer network. Malware files are run in a virtual environment made with the aid of emulators, hypervisors, and programs like VMware or Virtual Box in order to prevent damage to the host system. Malware can be identified by keeping track of its activities, such as the creation or deletion of files (Singh, J., & Singh, 2018).

2.2. Machine Learning Techniques

Machine learning, a branch of AI, utilizes data and algorithms to mimic human pattern recognition and enhance performance accuracy (Greener, Joe G., et al, 2022). In malware analysis, machine learning automates the process of analyzing and classifying malware, using techniques like clustering and classification to identify patterns and categorize unknown malware into specific families (Patil, Rajvardhan, and Wei Deng, 2020). DT, RF, and GT are commonly employed for malware classification.

Decision Trees (DT) are highly effective and efficient methodologies in machine learning, successfully addressing real-world challenges in artificial intelligence (Trabelsi et al., 2019). They resemble human decision-making processes, handling both discrete and continuous data inputs (Charbuty & Abdulazeez, 2021). DT is commonly used in data mining to create classifiers, categorize data with training sets, and derive assumptions about categorical class names (Charbuty & Abdulazeez, 2021). Decision trees are widely implemented due to their simplicity, accuracy, and applicability to diverse data forms. They offer quick learning, visually intuitive representation, support for data-driven root cause analysis, and comparable or superior accuracy, especially with ensemble decision tree classifiers (Abdallah et al., 2018).

Random Forest (RF) is a popular ensemble learning technique for classification and regression tasks. It combines multiple decision trees to make predictions through majority voting. Ensemble learning improves accuracy and stability by integrating predictions from multiple classifiers, reducing variance (Sheykhmousa et al., 2020). RF's effectiveness is attributed to its use of bagging, where models are trained on bootstrap samples and predictions are combined through averaging. This enhances model precision and stability (Tyralis et al., 2019). RF is widely used in various domains, handling missing attributes and large sample sizes. It excels in complex and uncertain data, making it a powerful tool for machine learning tasks (Huljanah et al., 2019).

XGBoost is a highly effective algorithm that improves upon the limitations of traditional gradient boosting methods, offering better performance and reduced execution time (Weldegebriel et al., 2019). It has gained recognition as one of the leading algorithms in machine learning, particularly in Kaggle competitions, surpassing deep learning models in terms of speed and scalability (Dhieb et al., 2019). Another powerful gradient boosting framework is LightGBM, known as the "Light Gradient Boosting Machine." LightGBM excels in managing massive data volumes and distributed data processing, utilizing a histogram-based algorithm and a growth strategy that prioritizes individual leaves to accelerate training and minimize memory consumption (Taha & Malebary, 2020; Liang et al., 2020).

2.3. Feature Selection

Feature selection, the process of selecting relevant attributes from a dataset, is crucial for constructing accurate models and impacting prediction performance, runtime, generalization, and overfitting avoidance (Moran & Gordon, 2019). Three main approaches exist: wrapper, filter, and embedded. Wrapper methods involve using a predictive model to assess the performance of various feature subsets, but they are computationally intensive and carry the risk of overfitting. Filter methods, on the other hand, evaluate individual features based on their intrinsic characteristics and are computationally efficient, providing a robust feature subset. Filter methods are often referred to as Univariate Feature Selection as they analyze each feature individually to evaluate its predictive power (Rodríguez-Rodríguez et al., 2021). Embedded methods combine the benefits of wrapper and filter methods, performing feature selection and classification simultaneously using a specific learning algorithm. While they can improve performance, embedded methods tend to be computationally intensive, but they avoid the need to repeatedly evaluate different subsets, enhancing computational efficiency (Li et al., 2021).

2.4. REST API

A REST API is a type of web API that uses HTTP requests to retrieve and modify resources, typically in JSON or XML format. It enables clients to access and manipulate resources through stateless operations, providing a uniform interface for CRUD operations. REST APIs are identified by HTTP URLs and mapped to corresponding methods like GET, POST, PUT, and DELETE (Kim et al., 2022). For instance, a REST API designed for managing pets can use GET /pets to retrieve a list of pets and POST /pets to add a new pet to the collection. REST APIs can be utilized with various programming languages and simplify system development (Bach et al., 2019). However, testing REST API implementations that incorporate external libraries or frameworks can be challenging when the source code is not fully accessible (Viglianisi et al., 2020).

2.5. Related Works

Masabo (2019) proposed a novel NFE approach that combines structure and behavior features for effective classification and detection of polymorphic malware. To address data imbalance, the training set was balanced using SMOTE. Multiple machine learning algorithms, including GB, LDA, and KNN, were employed for classification, selected for their ability to handle multi-class problems. The dataset was partitioned into training and test sets, with the Gradient Boosting model achieving the highest accuracy of 94% on the balanced dataset.

Masabo et al. (2018) conducted a study on malware detection using big data analytics and machine learning techniques. Their approach was evaluated on a dataset after performing data cleaning and balancing. The focus was primarily on the Crypto and Zeus malware families, with a total of 2957 samples, consisting of 1815 Crypto and 1142 Zeus instances. The Crypto and Zeus classes were encoded as 0 and 1 respectively. The robustness of the deep learning method was assessed using Keras Deep Learning and SVM models. The dataset was split into train and test sets, and the Keras Deep Learning model achieved a top accuracy of 97%.

Morales-Molina et al. (2018) proposed using the RF algorithm to enhance the performance of malware classification. They tested their approach on two datasets: ClaMP Raw-5184 and CSDMC_API_Train. ClaMP Raw-5184 had 5184 samples with 55 characteristics, while CSDMC_API_Train had 388 samples. Feature selection involved PCA for ClaMP Raw-5184 and TF-IDF for CSDMC_API_Train. The data was divided into training and test sets, and machine learning algorithms (LR, DT, RF, and SVM) were trained on the training sets. RF achieved the best precision of 89.83% for ClaMP Raw-5184 and 96% for CSDMC_API_Train.

Hadiprakoso et al. (2020) proposed a resource-efficient approach to malware detection using two datasets: Android Malgenome, DREBIN, and CICMalDroid. They addressed data imbalance through SMOTE and oversampling. Machine learning and deep learning algorithms, such as SVM, Decision Tree, RF, Naive Bayes, K-NN, MLP, and GB, were utilized. PCA was employed for feature extraction. A hybrid malware analysis model combining static and dynamic analysis achieved the best performance, with GB demonstrating around 96% accuracy in static model tests and approximately 99% accuracy in hybrid model tests.

Chowdhury et al. (2017) proposed a hybrid architecture combining an MLP neural network with a BAM for malware detection. They used two datasets, one for malware and one for benign files, with a total of 10920 benign files and 41265 malicious files. These datasets were merged to create a common dataset. Their method utilized behavior-based API sequences and signature-based n-gram features. SVM, Decision Tree (J48), Naive Bayes, and RF classifiers were employed and evaluated using K-fold cross-validation with K=10. The proposed classifier achieved the highest accuracy of 98.6% among the machine learning classifiers.

3. Methodology

Several important stages need to be performed to ensure the effectiveness and usability of the model. These stages include data collection, data pre-processing, data splitting, feature selection, model construction, model evaluation, and application deployment with a machine learning API.

3.1. Data Collection

The dataset used for the study was compiled from a public repository on GitHub, provided by Marco Ramili (Ramilli, Marco, 2016). It consists of executable malware features stored in JSON files organized in different folders. The dataset includes characteristics and behaviors of various malware types, collected by utilizing sandbox tools. The dataset exclusively comprises malware files without any benign files. It focuses on classifying five malware families, namely APT1, Crypto, Locker, Zeus, and ShadowBrokers as shown in Table 1.

Table 1: Malware Families Used

Malware Family	Number of Samples
APT1	292
Crypto	2020
Locker	431
Zeus	2019
shadowbrokers	1270
Total	6032

3.2.Data Preprocessing

Data preprocessing was performed to clean, format and organize data, making it suitable for building and training machine learning models. This stage consisted of several steps, which are data integration, data cleaning, data transformation and data balancing.

Data integration involved combining JSON files from different malware families into a single dataset in CSV format. Each JSON file represented a row of data in the dataset. The process began by reading and converting individual JSON files into CSV format. To create the final dataset, all the files from the malware families were combined into a folder, and their data was appended to a data list, which was then used to construct the CSV dataset. The raw dataset was prepared by transforming the data into Excel files in CSV format.

Data cleaning involved identifying and rectifying errors, inconsistencies, and missing data in the dataset. When merging multiple data sources into a single dataset, errors such as duplicates or inaccuracies can occur. Examination of the dataset revealed missing or null values in certain features. The approach taken to handle these missing values was to fill them with a constant value, such as NA, except when the feature had null values throughout.

Data transformation involved converting categorical variables into numerical format to enable their use in machine learning algorithms. The dataset initially consisted of categorical variables, such as string data types, which were incompatible with certain machine learning algorithms. To address this, a Label Encoding technique (see Algorithm 1) was employed, as it is commonly used for handling categorical variables. Each label was assigned a unique integer based on alphabetical ordering, resulting in the representation of malware families as: 'APT':0, 'Crypto':1, 'Locker':2, 'Zeus':3, 'shadowbrokers':4.

Algorithm 1: Label Encoding

Algorithm 1 Label Encoding

```
1: import LabelEncoder from sklearn.preprocessing
2: cols = df.columns[4:69]
3: label_encoder = LabelEncoder()
4: for column in cols:
5:     df[column] = label_encoder.fit_transform(df[column])
```

Data balancing was performed to address the class imbalance issue in the dataset. The majority class, consisting of malware families Cryto and Zeus, had a high number of observations, while the minority classes (APT1, Locker, and shadowbrokers) had a significantly lower number of observations. To overcome this imbalance, the SMOTE (Synthetic Minority Over-sampling Technique) technique was applied (Masabo, E, 2019), which involved augmenting the minority class data by replication. After applying SMOTE, each malware family had an equal number of data points, representing 20% of the dataset. This ensured a fair and unbiased evaluation of model performance.

3.3.Feature Selection

Feature selection was performed using the Boruta algorithm, a wrapper algorithm based on Random Forest (RF) models (Residentmario, 2018). Boruta utilized feature value permutations to determine the importance of each feature (see Algorithm 2). The resulting feature rankings were used to select the important features. In the balanced dataset, features with a ranking of 1 were considered important, while in the imbalanced dataset, only features with a ranking of 1 were selected. These selected features were then used for training and testing the machine learning models.

Algorithm 2: Boruta

Algorithm 2 Boruta

```
import RandomForestClassifier from sklearn.ensemble
import BorutaPy from Boruta
rfc = RandomForestClassifier(criterion='gini', max_depth=12,
        max_features='sqrt', min_samples_split=10, n_estimators=400,
        random_state=0)
boruta_selector = BorutaPy(rfc, n_estimators='auto', verbose=2,
random_state=0)
boruta_selector.fit(np.array(X_train), np.array(y_train))
```

3.4.Model Construction

The task was to classify data into five distinct malware families using a multi-class classification approach. To achieve this, three machine learning algorithms (DT, RF, and LightGBM) were chosen as suitable solutions for addressing the multi-class classification challenge in the dataset. The dataset was split into training and testing sets with various proportions, such as 80% training and 20% testing, 70% training and 30% testing, and 60% training and 40% testing. The training set was utilized to develop machine learning models and feature sets, while the testing set was used to evaluate model performance. Hyperparameter tuning was conducted using grid search, involving an exhaustive search for optimal parameter combinations to achieve the best model results. This process was described in Algorithm 3.

Algorithm 3: GridSearchCv

Algorithm 3 GridSearchCv

```
import GridSearchCV from sklearn.model_selection
classifier = YourClassifier()
param_grid = {
   'param_name_1': [param_value_1a, param_value_1b, ...],
   'param_name_2': [param_value_2a, param_value_2b, ...],
    ...
   }
grid_search = GridSearchCV(classifier, param_grid, n_jobs=-1, cv=10,
                  verbose=3, scoring='accuracy')
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_
```

3.5.Model Evaluation

Different evaluation metrics were utilized to assess the performance of the machine learning model and identify its strengths and weaknesses. These metrics, including accuracy, precision, recall, and F1 score, helped determine the model's ability to generalize well to future data. The confusion matrix was employed to compare the model's true positive and negative outcomes with its false positive and negative outcomes.

$$\mathrm{Accuracy} = \frac{\mathrm{TN} + \mathrm{TP}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{1}$$

$$\mathrm{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{2}$$

$$\mathrm{Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{3}$$

$$\mathrm{F1\ Score} = 2\left(\frac{\mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}\right) \tag{4}$$

3.6.Application Deployment with Machine Learning API

To deploy the machine learning model as a web service, the LightGBM model was used and Flask and Joblib libraries were required. The deployment involved serializing and saving the model using Joblib. The Flask API was used to receive input data and provide predictions as responses. The home endpoint rendered an HTML page, while the prediction endpoint handled POST requests and triggered the 'predict' function.

The prediction function handled POST requests to the '/prediction' route (see Algorithm 4). It parsed the input data from JSON format to a Pandas DataFrame and used the LightGBM model to make predictions. The predictions were stored in the result dictionary, along with a message indicating the malware type. The function returned the result dictionary as a JSON response. The name == 'main' check read the port number from the command-line argument, using the default port 3000 if no valid port was provided. The LightGBM model was loaded, and if it failed, an error message was displayed. Finally, the Flask application was started with the specified port and debug mode set to True.

Algorithm 4: Prediction

Algorithm 4 Prediction

```
function predict():
    if request.method == 'POST' then
        try:
            data = request.form['data']
            input_dict = json.loads(data)
            input_df = pd.DataFrame.from_dict(input_dict)
            predictions = lgb_model.predict(input_df)
            result = {'predictions': str(predictions)}
            if predictions == 0 then
                result['message'] = 'Predicted Malware: APT1'
            ….
            else
                result['message'] = 'Unknown Malware'
            end if
            return jsonify(result)
        except Exception as e:
            print(str(e))
            return jsonify({'error': str(e)})
        end
    else
        return jsonify({'error': 'Invalid request method'})
    end if
end function
```

4. Result and Discussion

The experiments compared the classification result of DT, RF and LightGBM using different techniques explained in classification result. Across different train-test split ratio, Tables 2, 3, and 4 showed the results of comparing the models with various experiments. The confusion matrices of different models for 80-20 split were compared as shown in Figures 1, 2, 3, and 4.

4.1. Classification Results

Applying both SMOTE and Boruta resulted in the highest accuracy scores for the DT, RF, and LightGBM models in the 80-20 split. The scores were 0.7916, 0.8540, and 0.9050 (see Table 2). The same approach yielded the highest accuracy scores in the 70-30 split (see Table 3), with scores of 0.7601, 0.8389, and 0.9033 for the DT, RF, and LightGBM models. Applying both SMOTE and Boruta led to the highest accuracy scores in the 60-40 split (see Table 4), with scores of 0.7720, 0.8470, and

0.8943 for the DT, RF, and LightGBM models.

Table 2: Comparison Result of the Model (80-20 Split)

	Accuracy		
Experiments	DT	RF	LightGBM
Without SMOTE & Boruta	0.6015	0.7854	0.8351
Using Boruta Without SMOTE	0.6744	0.7978	0.8376
Using SMOTE Without Boruta	0.7594	0.8426	0.9059
Using SMOTE & Boruta	0.7916	0.8540	0.9050

Table 3: Comparison Result of the Model (70-30 Split)

	Accuracy		
Experiments	DT	RF	LightGBM
Without SMOTE & Boruta	0.6392	0.7713	0.8271
Using Boruta Without SMOTE	0.6840	0.7934	0.8315
Using SMOTE Without Boruta	0.7578	0.8360	0.9020
Using SMOTE & Boruta	0.7601	0.8389	0.9033

Table 4: Comparison Result of the Model (60-40 Split)

	Accuracy		
Experiments	DT	RF	LightGBM
Without SMOTE & Boruta	0.6680	0.7721	0.8189
Using Boruta Without SMOTE	0.6797	0.7944	0.8247
Using SMOTE Without Boruta	0.7401	0.8277	0.8955
Using SMOTE & Boruta	0.7720	0.8470	0.8943

With an 80/20 split, the LightGBM model confusion matrix had the highest TP rate for each malware family among all models and all experiments. In experiments without SMOTE and Boruta (see Fig. 1), the LightGBM model achieved the highest TP rates with APT1, Crypto and ShadowBroker values of 0.86, 0.96 and 0.99. The LightGBM model achieved the highest TP rates (values 0.88, 0.97, and 0.99) for APT1, Crypto, and ShadowBrokers in experiments with Boruta without SMOTE (see Fig. 2). The LightGBM model achieved the highest TP rates (values 0.99, 0.94, 0.96, and 0.99) for APT1, Crypto, Locker, and ShadowBroker in experiments with SMOTE without Boruta (see Fig. 3). The LightGBM model achieved the highest TP rates (values 0.99, 0.93, 0.97, and 0.99) for APT1, Crypto, Locker, and ShadowBroker in experiments with SMOTE and Boruta (see Fig. 4).

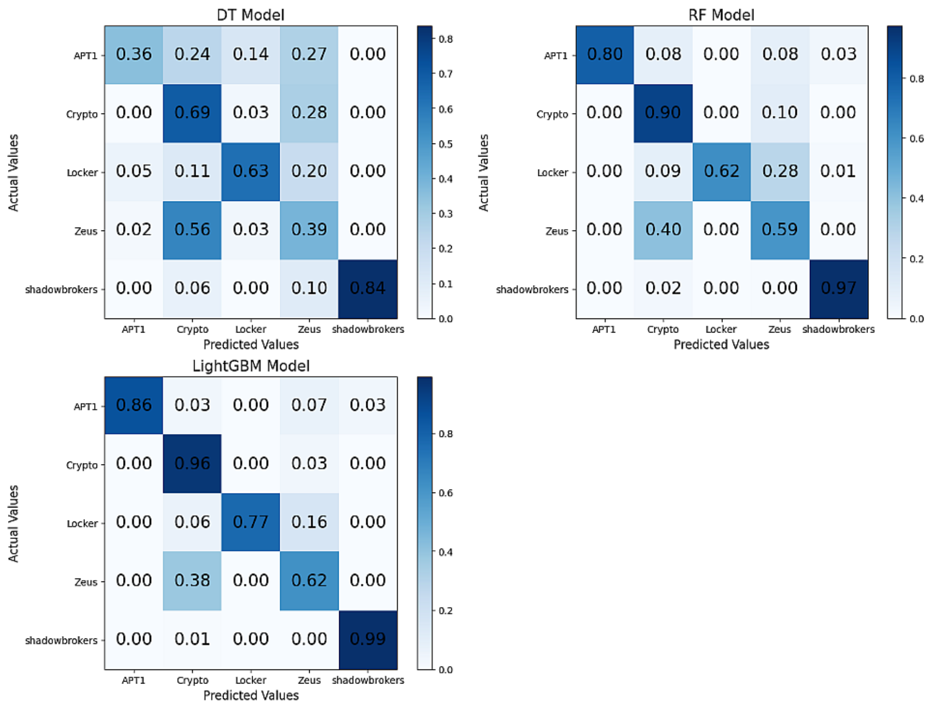


Fig. 1: Confusion Matrix for 80-20 Split (Without SMOTE & Boruta)

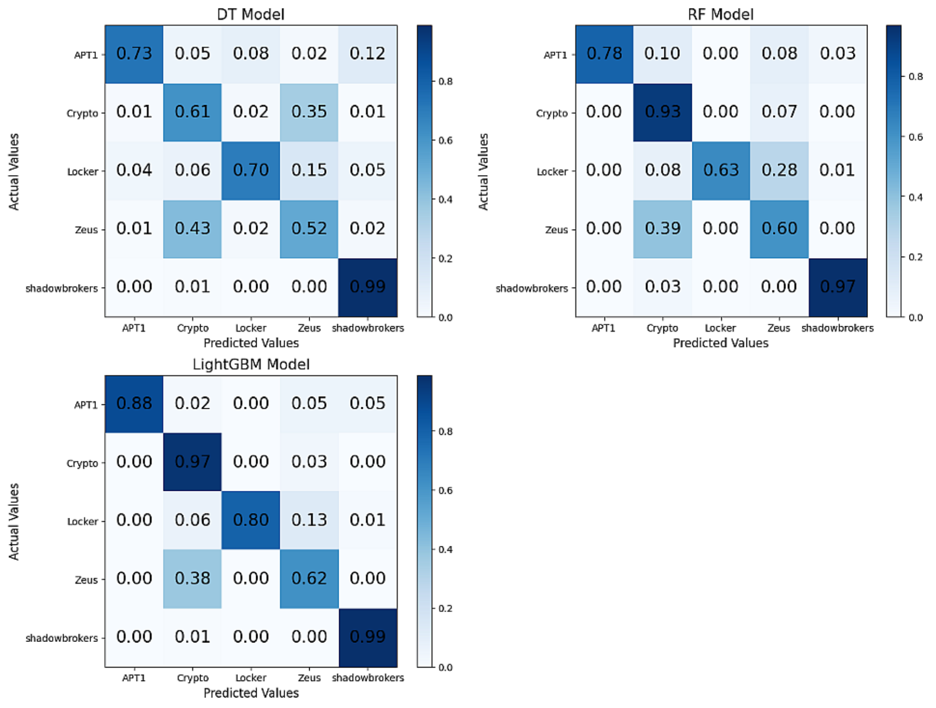


Fig. 2: Confusion Matrix for 80-20 Split (Using Boruta Without SMOTE)

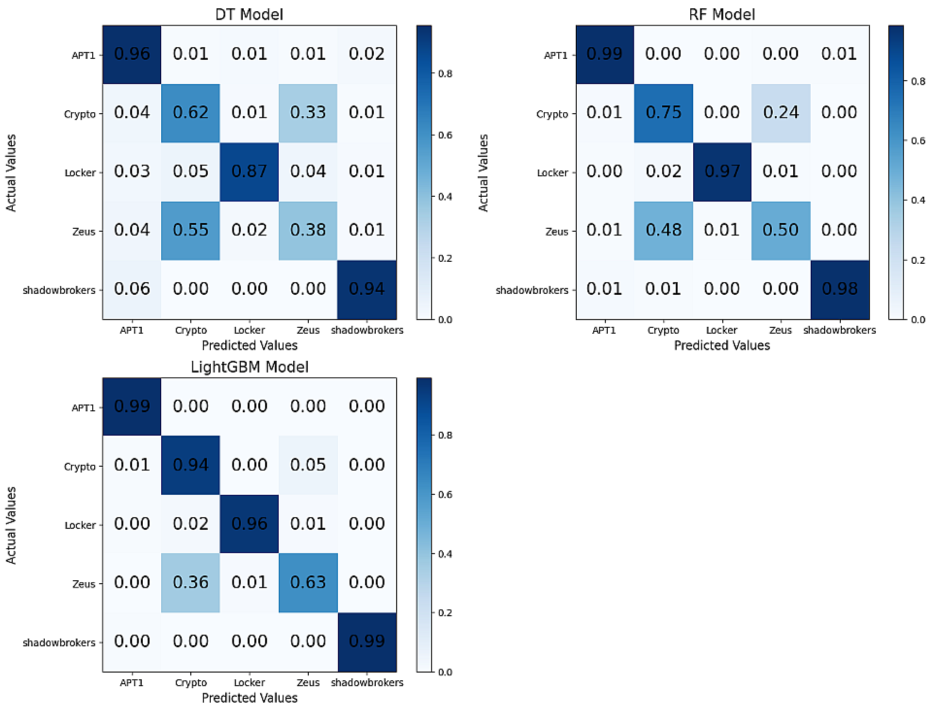


Fig. 3: Confusion Matrix for 80-20 Split (Using SMOTE Without Boruta)

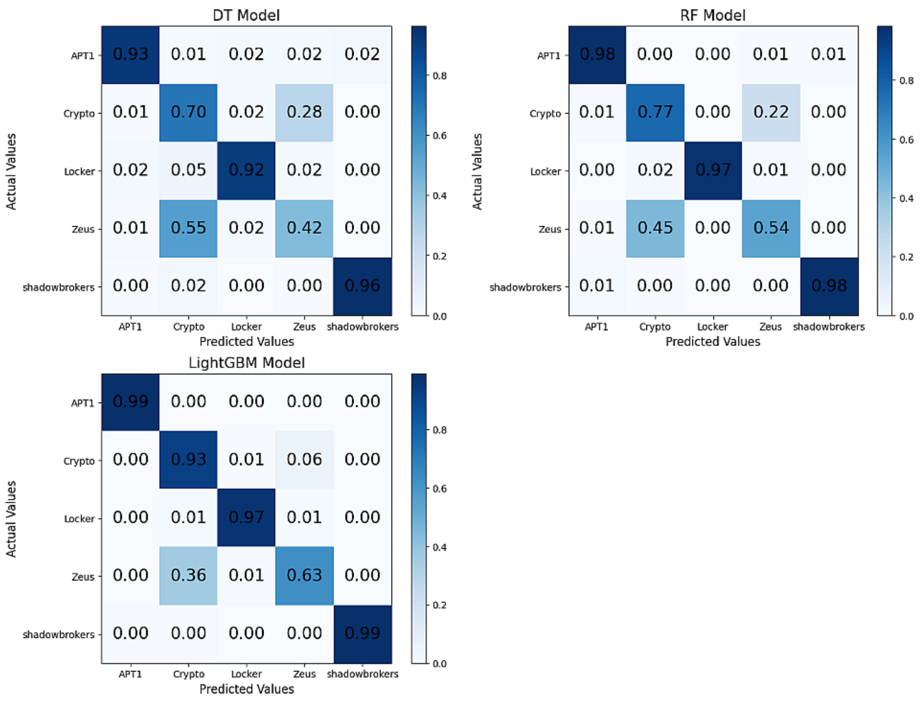
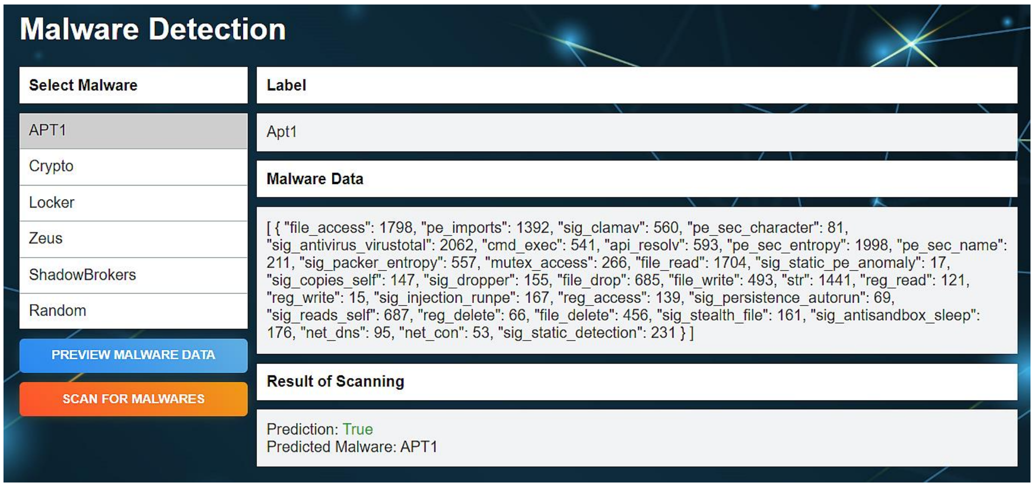


Fig. 4: Confusion Matrix for 80-20 Split (Using SMOTE & Boruta)

In summary, LightGBM consistently outperformed DT and RF models in terms of accuracy across all train-test split ratios. SMOTE without Boruta achieved the highest accuracy for both 80-20 and 60-40 splits, while combining SMOTE and Boruta yielded the best results for the 70-30 split. The LightGBM model consistently exhibited the highest true positive (TP) rates for each malware family according to the confusion matrix for 80-20 split, further confirming its superior performance compared to DT and RF models.

4.2. Application Deployment Results

The model utilizes LightGBM for malware detection through a REST API. Users can select from malware types like APT1, Crypto, Locker, Zeus, Shadowbrokers, and Random. If no type is chosen, the option to preview malware data is disabled. After selecting a type and clicking "Preview Malware Data," the system displays the label and corresponding data. Clicking "Scan for Malwares" shows the scanning results, indicating true or false for each malware (see Fig. 4). The scanning option is disabled if "Preview Malware Data" is not clicked.



Label

Apt1

Malware Data

```
[ { "file_access": 1798, "pe_imports": 1392, "sig_clamav": 560, "pe_sec_character": 81,
"sig_antivirus_virustotal": 2062, "cmd_exec": 541, "api_resolv": 593, "pe_sec_entropy": 1998, "pe_sec_name":
211, "sig_packer_entropy": 557, "mutex_access": 266, "file_read": 1704, "sig_static_pe_anomaly": 17,
"sig_copies_self": 147, "sig_dropper": 155, "file_drop": 685, "file_write": 493, "str": 1441, "reg_read": 121,
"reg_write": 15, "sig_injection_runpe": 167, "reg_access": 139, "sig_persistence_autorun": 69,
"sig_reads_self": 687, "reg_delete": 66, "file_delete": 456, "sig_stealth_file": 161, "sig_antisandbox_sleep":
176, "net_dns": 95, "net_con": 53, "sig_static_detection": 231 } ]
```

Result of Scanning

Prediction: True
Predicted Malware: APT1

Fig. 4: Scanning Result of Malware Detection

5. Conclusion

Machine learning techniques have proven to be highly valuable in the detection and classification of malware. By leveraging these techniques, we can effectively identify and categorize malicious software by analyzing patterns, behaviors, and characteristics from large datasets. This approach enables swift detection and enhances our ability to combat evolving malware threats. An evaluation of three models, DT, RF, and LightGBM, was conducted to select the most suitable model for malware detection. The results clearly indicate that LightGBM outperforms DT and RF, achieving the highest accuracy score of 0.9059 in the experiment without Boruta but with SMOTE, specifically for the 80-20 split. These findings establish LightGBM as the preferred choice for accurate and efficient malware detection.

This research aims to achieve several objectives: conducting EDA to gain insights into malware, developing a robust machine learning model for malware identification, and evaluating the performance of different models. These objectives have been successfully accomplished, contributing to the field of malware detection and classification. However, certain constraints exist. Obtaining sufficient training data is challenging due to the dynamic nature of malware and the difficulty in obtaining labeled samples. Training complex machine learning models also requires significant time and resources, which can be a limitation for resource-constrained organizations or time-sensitive scenarios.

References

[1] Ramilli, M. (2016). Malware Training Sets: a machine learning dataset for everyone. Marco Ramilli Web Corner.

[2] Ucci, D., Aniello, L., & Baldoni, R. (2019). Survey of machine learning techniques for malware analysis. Computers & Security, 81, 123-147.

[3] Singh, J., & Singh, J. (2018). Challenge of malware analysis: malware obfuscation techniques. International Journal of Information Security Science, 7(3), 100-110.

[4] Chikapa, M., & Namanya, A. P. (2018, August). Towards a fast off-line static malware analysis framework. In 2018 6th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW) (pp. 182-187). IEEE.

[5] Ijaz, M., Durad, M. H., & Ismail, M. (2019, January). Static and dynamic malware analysis using machine learning. In 2019 16th International bhurban conference on applied sciences and technology (IBCAST) (pp. 687-691). IEEE.

[6] Chakkaravarthy, S. S., Sangeetha, D., & Vaidehi, V. (2019). A survey on malware analysis and mitigation techniques. Computer Science Review, 32, 1-23.

[7] Patil, R., & Deng, W. (2020, March). Malware Analysis using Machine Learning and Deep Learning techniques. In 2020 SoutheastCon (Vol. 2, pp. 1-7). IEEE.

[8] Greener, J. G., Kandathil, S. M., Moffat, L., & Jones, D. T. (2022). A guide to machine learning for biologists. Nature Reviews Molecular Cell Biology, 23(1), 40-55.

[9] Sheykhmousa, M., Mahdianpari, M., Ghanbari, H., Mohammadimanesh, F., Ghamisi, P., & Homayouni, S. (2020). Support vector machine versus random forest for remote sensing image classification: A meta-analysis and systematic review. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 13, 6308-6325.

[10] Tyralis, H., Papacharalampous, G., & Langousis, A. (2019). A brief review of random forests for water scientists and practitioners and their recent history in water resources. Water, 11(5), 910.

[11] Huljanah, M., Rustam, Z., Utama, S., & Siswantining, T. (2019, June). Feature selection using random forest classifier for predicting prostate cancer. In IOP Conference Series: Materials Science and Engineering (Vol. 546, No. 5, p. 052031). IOP Publishing.

[12] Moran, M., & Gordon, G. (2019). Curious feature selection. Information Sciences, 485, 42-54.

[13] Noorbehbahani, F., Rasouli, F., & Saberi, M. (2019, August). Analysis of machine learning techniques for ransomware detection. In 2019 16th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC) (pp. 128-133). IEEE.

[14] Rodríguez-Rodríguez, I., Rodríguez, J. V., Woo, W. L., Wei, B., & Pardo-Quiles, D. J. (2021). A comparison of feature selection and forecasting machine learning algorithms for predicting glycaemia in type 1 diabetes mellitus. Applied Sciences, 11(4), 1742.

[15] Li, W., Chen, L., Zhao, J., & Wang, W. (2021). Embedded feature selection based on relevance vector machines with an approximated marginal likelihood and its industrial application. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 52(4), 2601-2614.

[16] Masabo, E. (2019). A Feature Engineering Approach for Classification and Detection of Polymorphic Malware using Machine Learning (Doctoral dissertation, Makerere University).

[17] Masabo, E., Kaawaase, K. S., & Sansa-Otim, J. (2018, May). Big data: deep learning for detecting malware. In 2018 IEEE/ACM Symposium on Software Engineering in Africa (SEiA) (pp. 20-26). IEEE.

[18] Morales-Molina, C. D., Santamaria-Guerrero, D., Sanchez-Perez, G., Perez-Meana, H., & Hernandez-Suarez, A. (2018, November). Methodology for malware classification using a random forest classifier. In 2018 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC) (pp. 1-6). IEEE.

[19] Hadiprakoso, R. B., Kabetta, H., & Buana, I. K. S. (2020, November). Hybrid-based malware analysis for effective and efficiency android malware detection. In 2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS) (pp. 8-12). IEEE.

[20] Chowdhury, M., Rahman, A., & Islam, R. (2017, June). Malware analysis and detection using data mining and machine learning classification. In International conference on applications and techniques in cyber security and intelligence (pp. 266-274). Edizioni della Normale, Cham.

[21] AMR, et al. “It Threat Evolution in Q2 2021. PC Statistics.” Securelist English Global Securelistcom, 13 May 2021, securelist.com/it-threat-evolution-in-q2-2021-pc-statistics/103607/.

[22] Residentmario. (2018, April 20). Automated feature selection with Boruta. Retrieved January 26, 2023, from https://www.kaggle.com/code/residentmario/automated-feature-selection-with-boruta/notebook

[23] Horkoff, J., Hammouda, I., & Knauss, E. (2018). Emerging perspectives of application programming interface strategy: A framework to respond to business concerns. IEEE software, 37(2), 52-59.

[24] Viglianisi, E., Dallago, M., & Ceccato, M. (2020, October). Resttestgen: automated black-box testing of restful apis. In 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST) (pp. 142-152). IEEE.

[25] Kim, M., Xin, Q., Sinha, S., & Orso, A. (2022, July). Automated test generation for rest apis: No time to rest yet. In Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 289-301).

[26] Bach, K., Mathisen, B. M., & Jaiswal, A. (2019). Demonstrating the myCBR Rest API. In ICCBR Workshops (pp. 144-155).

[27] Weldegebriel, H. T., Liu, H., Haq, A. U., Bugingo, E., & Zhang, D. (2019). A new hybrid convolutional neural network and eXtreme gradient boosting classifier for recognizing handwritten Ethiopian characters. IEEE Access, 8, 17804-17818.

[28] Dhieb, N., Ghazzai, H., Besbes, H., & Massoud, Y. (2019, September). Extreme gradient boosting machine learning algorithm for safe auto insurance operations. In 2019 IEEE international conference on vehicular electronics and safety (ICVES) (pp. 1-5). IEEE.

[29] Taha, A. A., & Malebary, S. J. (2020). An intelligent approach to credit card fraud detection using an optimized light gradient boosting machine. IEEE Access, 8, 25579-25587.

[30] Liang, W., Luo, S., Zhao, G., & Wu, H. (2020). Predicting hard rock pillar stability using GBDT, XGBoost, and LightGBM algorithms. Mathematics, 8(5), 765.

[31] Trabelsi, A., Elouedi, Z., & Lefevre, E. (2019). Decision tree classifiers for evidential attribute values and class labels. Fuzzy Sets and Systems, 366, 46-62.

[32] Charbuty, B., & Abdulazeez, A. (2021). Classification based on decision tree algorithm for machine learning. Journal of Applied Science and Technology Trends, 2(01), 20-28.

[33] Abdallah, I., Ntertimanis, V., Mylonas, C., Tatsis, K., Chatzi, E., Dervilis, N., ... & Eoghan, M. (2018). Fault diagnosis of wind turbine structures using decision tree learning algorithms with big data. Safety and Reliability–Safe Societies in a Changing World, 3053-30