Android Malware Detection

by
Lacey Hamilton | Megha Viswanath | Yena Hong

Masters of Science in Data Science
Michigan State University

May 1, 2023

ABSTRACT

Android malware detection and classification are essential tasks in maintaining the security of mobile devices and users' data. This project aims to develop a robust machine learning model to accurately classify Android malware into distinct types and differentiate between malignant and benign samples. Through a series of Jupyter notebooks, we perform an exploratory data analysis, binary classification between malignant and benign malware, and feature engineering on a sample dataset to understand and confirm the effectiveness of various techniques. The final notebook consolidates the insights and techniques derived from previous notebooks to create a high-performing classifier for Android malware.

Using a diverse set of classifiers, including Decision Trees, XGBoost, LightGBM, RandomForest, and CatBoost, we ensure a comprehensive and robust model for malware detection. We also employ preprocessing techniques such as Standard Scaler for data normalization and SMOTE for addressing class imbalance, enhancing the model's performance and generalization capabilities.

The resulting machine learning model demonstrates the potential for accurate and efficient Android malware classification, providing a valuable tool for securing mobile devices and protecting users from malicious applications. The project's structured and systematic approach enables continuous improvement and adaptability, ensuring the model remains relevant and effective in the ever-evolving landscape of Android malware threats.

1.1 INTRODUCTION

The proliferation of mobile devices and applications has led to a surge in Android malware, posing significant security risks to users' personal data and device functionality. To address this challenge, the development of effective and efficient malware detection and classification models is crucial. In this project, we aim to create a machine learning model that accurately classifies Android malware into distinct types and differentiates between malignant and benign samples.

Our approach to the project is systematic and structured, ensuring a comprehensive understanding of the data and a robust model development process. We begin by analyzing the entire dataset, gaining insights into the features and relationships between different malware types and benign samples. This initial analysis allows us to explore the data's structure, characteristics, and statistical properties.

Next, we create a sample dataset to work on a more focused and manageable context. We perform binary classification between malignant and benign malware, serving as a foundational step in the overall malware classification process. Using the sample dataset, we also experiment with various feature engineering techniques, enabling us to understand and confirm their effectiveness before applying them to the entire dataset.

With the insights derived from the sample dataset, we proceed to develop and evaluate machine learning models for classifying Android malware into distinct types. We employ a diverse set of classifiers, including Decision Trees, XGBoost, LightGBM, RandomForest, and CatBoost, ensuring a comprehensive and robust model for malware detection. Preprocessing techniques such as Standard Scaler and SMOTE are also applied to enhance the model's performance and generalization capabilities.

This project report documents the methodology, techniques, and findings of our work, highlighting the potential of machine learning in addressing the critical task of Android malware detection and classification. Through a structured and systematic approach, we demonstrate the effectiveness of our model in securing mobile devices and protecting users from malicious applications, contributing to a safer mobile ecosystem.

1.2 ABOUT THE DATA

The Canadian Institute of Cybersecurity has developed a comprehensive Android Malware Dataset called CIC-AndMal2017. The dataset comprises 10,854 samples, including 4,354 malware and 6,500 benign applications, collected from various sources. To ensure the authenticity of the data, the applications were run on real smartphones rather than emulators, as advanced malware samples can detect and modify their behavior in an emulator environment.

The benign applications were collected from the Google Play market, published between 2015 and 2017. The malware samples, on the other hand, were installed on real devices and categorized into four types: Adware, Ransomware, Scareware, and SMS Malware. These samples belong to 42 unique malware families, with a varying number of captured samples for each family.

To obtain a comprehensive view of the malware samples and overcome the stealthiness of advanced malware, three distinct data capturing states were defined: Installation, Before Restart, and After Restart. A specific scenario was created for each malware category to capture the network traffic

features (.pcap files) during these states. Using CICFlowMeter-V3, over 80 features were extracted from the network traffic data, providing a rich dataset for the development of machine learning models for Android malware detection and classification.

1.3 METHODOLOGY

Our methodology for this project is structured into four main stages, which are detailed in the corresponding notebooks. Each stage plays a crucial role in understanding the data and developing a robust machine learning model for Android malware classification

1. Initial Data Analysis (Notebook1): The first stage involves an exploratory data analysis of the entire dataset. This analysis provides insights into the data's structure, characteristics, and statistical properties, enabling us to gain a better understanding of the features and relationships between different malware types and benign samples.
2. Binary Classification (Notebook2): In the second stage, we focus on binary classification between malignant and benign malware. We develop and evaluate machine learning models to differentiate between malicious and benign applications, serving as a foundational step in the overall malware classification process.
3. Feature Engineering on Sample Data (Notebook3): The third stage presents our initial work on a sample dataset to understand and confirm the effectiveness of various feature engineering techniques. This stage allows us to experiment with different approaches in a more focused and manageable context before applying them to the entire dataset.
4. Final Model Development and Evaluation (Notebook4): The final stage consolidates the insights and techniques derived from the previous stages to create a robust and accurate machine learning model for Android malware classification. This stage includes feature engineering, resampling, encodings, and model development to obtain the best possible classifier for distinguishing between different types of malignant malware.

In the following sections, we will provide a detailed description of each stage, discussing the techniques used, the rationale behind our choices, and the results obtained.

1.4 EXPLORATORY DATA ANALYSIS

1.4.1 FEATURE DESCRIPTION

Our dataset consists of 1,411,064 entries and 86 columns, providing a comprehensive view of the Android malware landscape. The dataset includes information on various aspects of network traffic and packet characteristics, captured during different states of malware behavior (installation, before restart, and after restart). Among the 86 columns, 75 are of float64 data type, and 11 are of object data type. Some of the key features in the dataset include source and destination IPs, ports, protocols, flow duration, total forward and backward packets, packet length statistics, flow rate measures, and flag counts.

The dataset captures a wide range of network traffic features, offering a rich source of information for

developing machine learning models to classify Android malware effectively. The comprehensive feature set allows us to gain insights into the behavior of various malware types and their differences from benign applications, facilitating the development of robust and accurate classification models.

1.4.2 DATA CLEANING

The following bar plot illustrates the count of missing values for each column within the dataset. From the visualization, it is evident that the number of NaN values is limited to 12 or fewer for all columns. Subsequently, we investigated the distribution of these NaN values across the rows and discovered that they are not uniformly dispersed. Given the considerable size of the original dataset, which contains 1,411,063 entries, we have chosen to remove the 12 rows containing NaN values. This decision is based on the minimal impact that the removal of such a small number of rows will have on our model's performance and overall results.

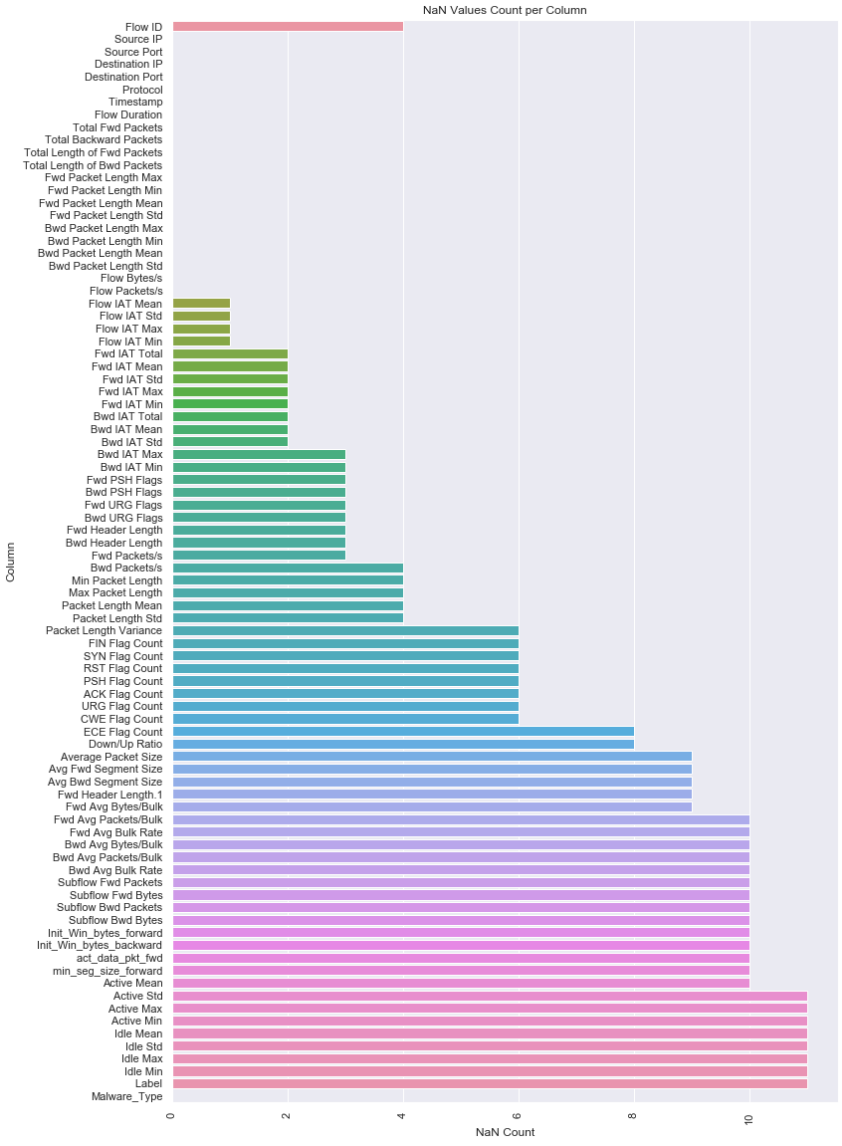


Figure 1

1.4.3 FEATURE VARIABLE EXPLORATION

In this section, we delve into the process of exploring the feature variables of our dataset to better understand their properties and potential impact on our malware classification model. As we examined the columns, we encountered a few issues that required further investigation and clarification. For instance, we noticed that the 'Protocol' column, which was expected to be categorical in nature, contained numerical data. Upon closer examination, we discovered that these numbers represented various protocols such as TCP, UDP, and others, with values 6, 17, and 0 corresponding to each protocol respectively. This assignment of values is based on the standards set by the Internet Assigned Numbers Authority (IANA).

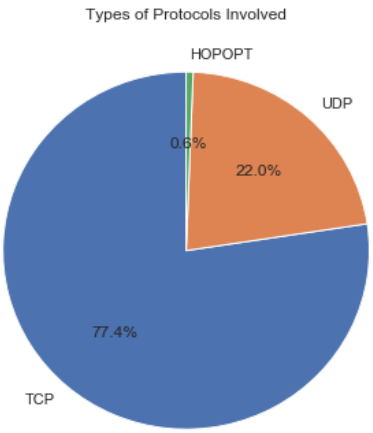


Figure 2

Upon further investigation of the dataset, we discovered two columns with the same name: 'Fwd_Header_Length' and 'Fwd_Header_Length.1'. We found that these columns contained duplicate values, which added redundancy to our analysis. To streamline the process, we removed one of these redundant columns.

Furthermore, we observed that some rows had object data types, while their counterparts had float data types for the same type of data but in the opposite direction of flow. We identified the following columns with object data types: 'Flow_ID', 'Source_IP', 'Destination_IP', 'Timestamp', 'Flow_IAT_Min', 'Packet_Length_Std', 'CWE_Flag_Count', 'Down/Up_Ratio', 'Fwd_Avg_Bytes/Bulk', 'Label', and 'Malware_Type'. Utilizing object data types in models can be cumbersome, especially when most columns already have numerical data types. Consequently, we decided to explore these rows further.

We found that the 'Flow_ID' column was simply a combination of 'Destination_IP', 'Source_IP', 'Destination_Port', 'Source_Port', and 'Protocol'. Thus, we removed the 'Flow_ID' column. Additionally, we recognized the potential of source and destination ports to help identify certain types of traffic, such as well-known protocols. Therefore, we changed their column type to float.

Regarding the IP address columns, we realized that the IP address could only help us trace the location if the traffic was originating from a public network. If traffic was originating from a private network, tracing its location using APIs was not possible. Moreover, tracing the location using API calls for such a large dataset was limited. Therefore, we decided to drop the 'Source_IP' and 'Destination_IP' columns. However, we did extract unique destination IP addresses to test the function of tracing back IP addresses and to gain insights into the locations where data from malware devices was being downloaded. The map below depicts the cities included in these destinations.

Figure 3

Lastly, we decided to drop the 'Timestamp' column, as it only provided the date and time when the data was collected, which did not pertain to our goal of malware detection. We also noticed that several columns, including 'Flow_IAT_Min', 'Packet_Length_Std', 'CWE_Flag_Count', and 'Down/Up_Ratio', had object data types but were of float type. As a result, we changed the data type of these columns to float.

1.4.4 TARGET VARIABLE EXPLORATION

We delved into the analysis of the target variable within the malware classification project. Our initial step was to explore the distribution of Benign and Malignant Malware, the two primary categories under consideration. The Malignant category encompasses four distinct subtypes: Adware, Ransomware, Scareware, and SMSmalware. A bar graph representation of the data revealed that the Malignant category dominates the dataset, accounting for the majority of the observations.

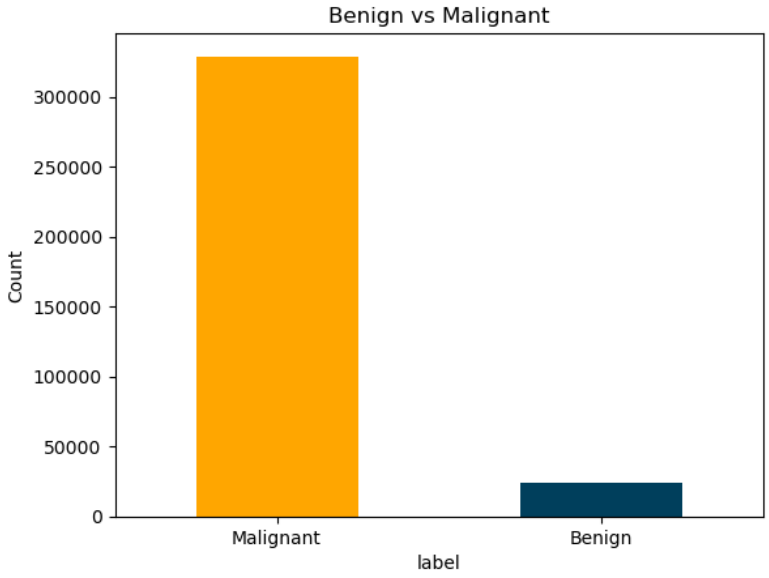


Figure 4

To obtain a more comprehensive understanding, we proceeded to subdivide the Malignant category into its four constituent subtypes, aiming to uncover any disparities in their distribution. The resulting graph exhibited a discernible contrast among the various malware types. Adware, Ransomware, and Scareware demonstrated a relatively balanced distribution, while SMSmalware stood out, constituting roughly two-thirds of the combined size of the other three subtypes.

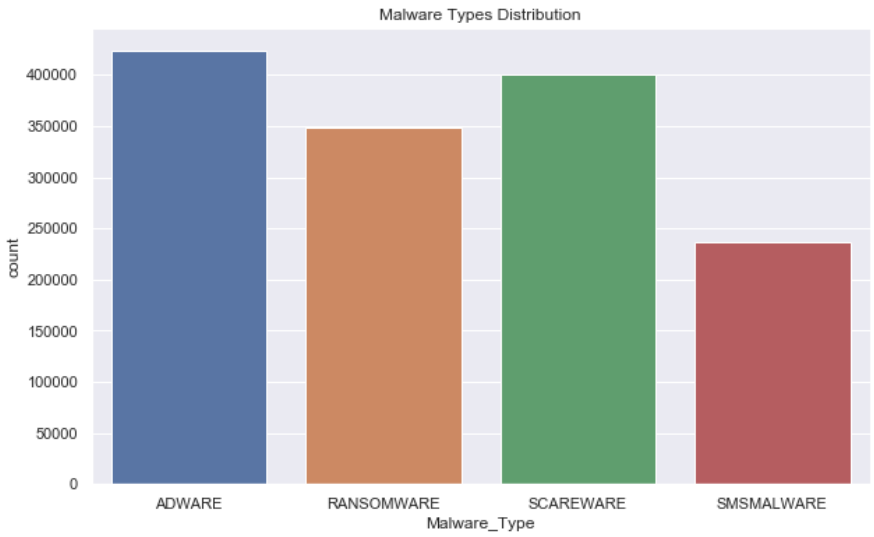


Figure 5

In order to delve deeper, we performed an in-depth analysis of each malware type, pinpointing the specific families associated with them. This information proved valuable for visualizing the distribution of malware families within each subtype. Consequently, we generated four distinct figures to effectively represent the family distributions for Adware, Ransomware, Scareware, and SMSmalware respectively.

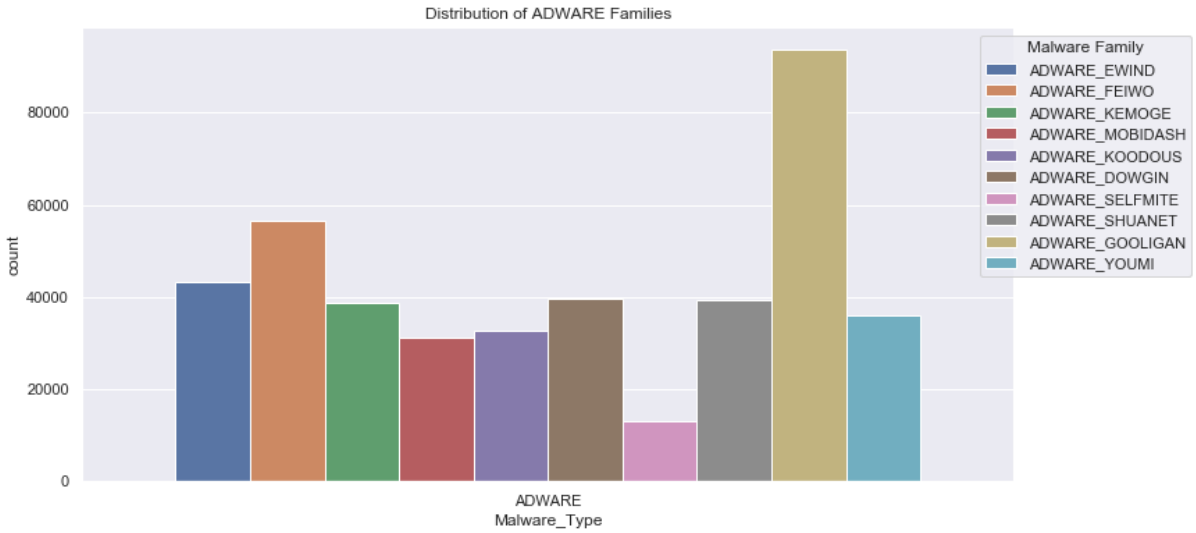


Figure 6

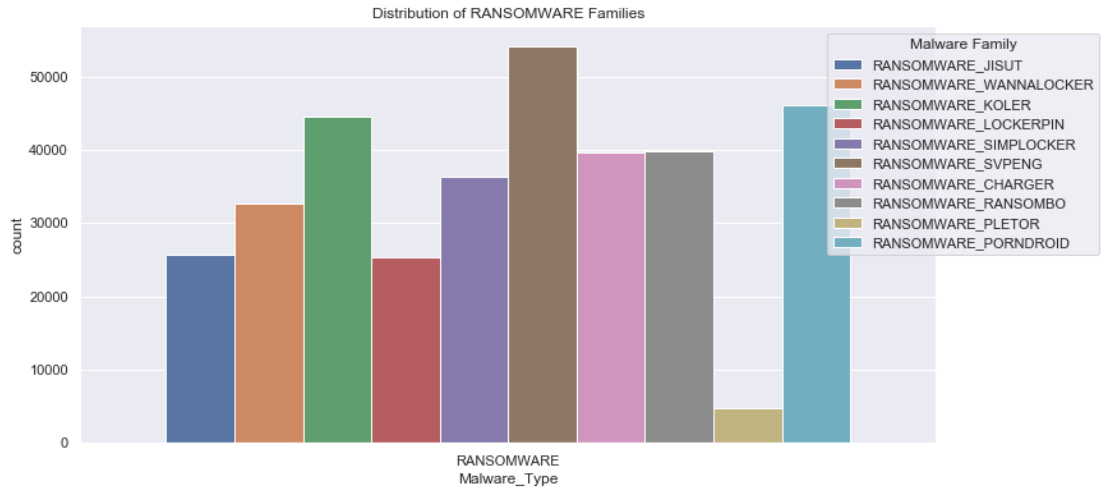


Figure 7

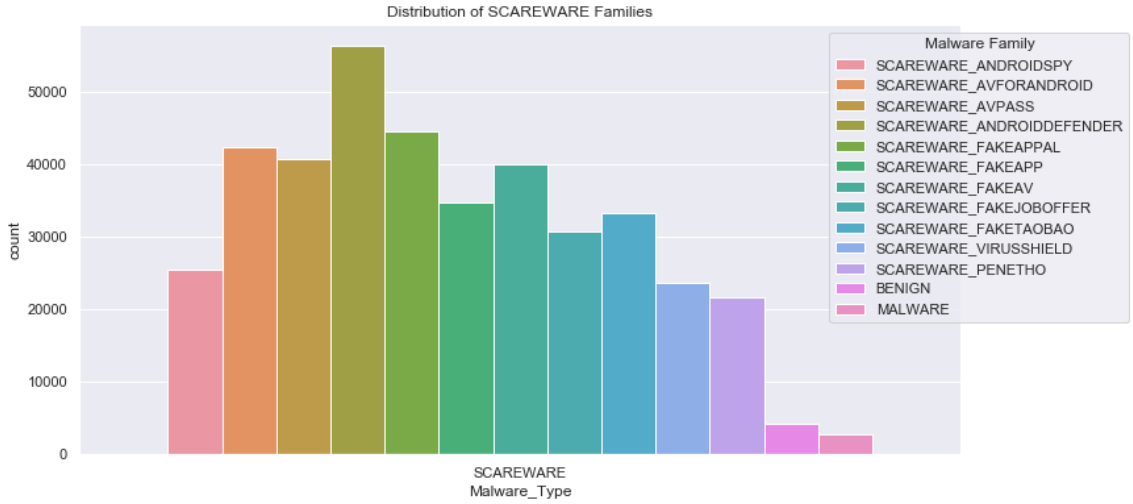


Figure 8

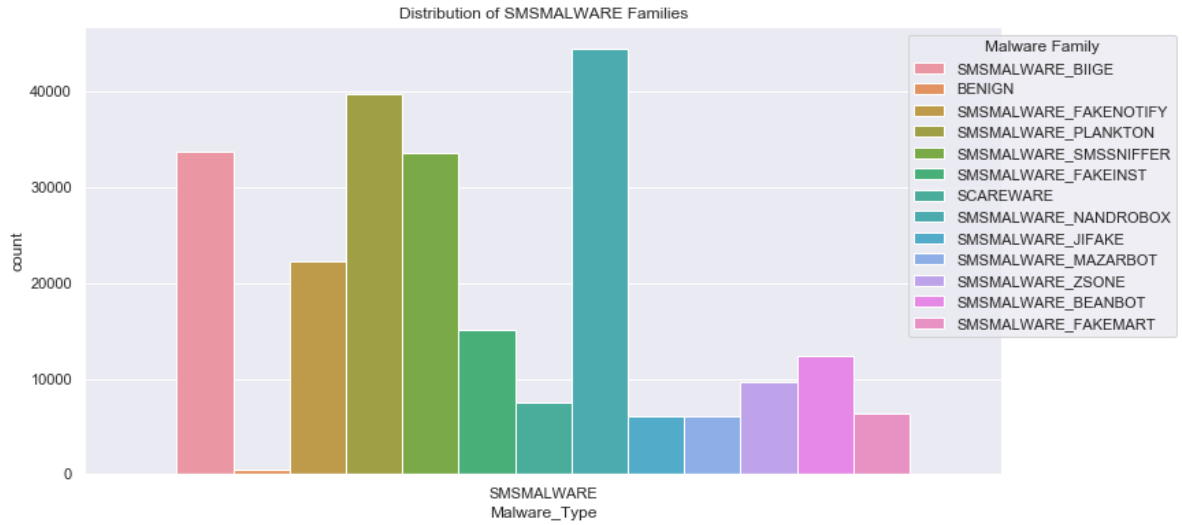


Figure 9

This refined and well-structured exploration of the target variable has laid the foundation for further investigation in the malware classification project, providing valuable insights into the distribution of the different malware types and their associated families.

1.5 FEATURE ENGINEERING

After retaining the columns from the previous process, we aimed to improve the model's performance in detecting malware by creating new features. Four types of features were created: ratio, aggregated, interaction, and statistical. These features provide us with a better understanding of the relationships and patterns between variables in the dataset.

1. Ratio features are generated by computing the ratios between related features to capture their interactions.

- Fwd_Bwd_Packet_Length_Mean_Ratio= Fwd_Packet_Length_Mean / Bwd_Packet_Length_Mean
    - This feature computes the ratio between the mean length of forward packets and the mean length of backward packets. It captures the relationship between the lengths of packets in both directions of the flow.
- Fwd_Bwd_Packets_per_s_Ratio = Fwd_Packets/s / Bwd_Packets/s
    - This feature computes the ratio between the rate of forward packets and the rate of backward packets. It captures the relationship between the flow rates of packets in both directions.
- Fwd_Bwd_Header_Length_Ratio = Fwd_Header_Length / Bwd_Header_Length
    - This feature computes the ratio between the header length of forward packets and the header length of backward packets. It captures the relationship between the header lengths of packets in both directions.
- Fwd_Bwd_IAT_Mean_Ratio = Fwd_IAT_Mean / Bwd_IAT_Mean
    - This feature computes the ratio between the mean inter-arrival time of forward packets and the mean inter-arrival time of backward packets. It captures the relationship between the inter-arrival times of packets in both directions.
- Flow_Bytes_Packets_per_s_Ratio = Flow_Bytes/s / Flow_Packets/s
    - This feature computes the ratio between the rate of bytes in a flow and the rate of packets in the flow. It captures the relationship between the flow rates of bytes and packets.
- Avg_Fwd_Bwd_Segment_Size_Ratio = Avg_Fwd_Segment_Size / Avg_Bwd_Segment_Size
    - This feature computes the ratio between the average segment size of forward packets and the average segment size of backward packets. It captures the relationship between the average segment sizes of packets in both directions.

2. Aggregates combine related features using functions like sum, average, min, max, or standard deviation. They create aggregated features by taking the sum or average of multiple columns.

- Total_Packets = Total_Fwd_Packets + Total_Backward_Packets
    - This feature represents the total number of packets in a flow, which is calculated by adding up the number of forward packets and backward packets.

- Total_Bytes = Total_Length_of_Fwd_Packets + Total_Length_of_Bwd_Packets
    - This feature represents the total number of bytes in a flow, which is calculated by adding up the length of forward packets and backward packets.
- Avg_Packet_Length = (Fwd_Packet_Length_Mean + Bwd_Packet_Length_Mean) / 2
    - This feature represents the average length of packets in a flow, which is calculated by taking the mean of the forward packet length and backward packet length.
- Total_IAT = Fwd_IAT_Total + Bwd_IAT_Total
    - This feature represents the total inter-arrival time between packets in a flow, which is calculated by adding up the inter-arrival time of forward packets and backward packets.
- Total_Header_Length = Fwd_Header_Length + Bwd_Header_Length
    - This feature represents the total length of headers in a flow, which is calculated by adding up the header length of forward packets and backward packets.

3. Interaction features are computed by multiplying, dividing, or adding related columns.

- Fwd_Bwd_Packet_Length_Mean_Diff = Fwd_Packet_Length_Mean - Bwd_Packet_Length_Mean
    - This feature calculates the difference between the mean packet length of forward packets and backward packets. It can provide insights into the directionality of the traffic and the differences in packet size between the two directions.
- Fwd_Bwd_Packet_Length_Mean_Product = Fwd_Packet_Length_Mean * Bwd_Packet_Length_Mean
    - This feature calculates the product of the mean packet length of forward packets and backward packets. It can provide information about the overall packet length and the potential interactions between the directions.
- Fwd_Bwd_IAT_Mean_Diff = Fwd_IAT_Mean - Bwd_IAT_Mean
    - This feature calculates the difference between the mean inter-arrival time (IAT) of forward packets and backward packets. It can provide insights into the directionality of the traffic and the differences in timing between the two directions.
- Fwd_Bwd_IAT_Mean_Product = Fwd_IAT_Mean * Bwd_IAT_Mean
    - This feature calculates the product of the mean IAT of forward packets and backward packets. It can provide information about the overall timing of the traffic and the potential interactions between the directions.

4. Statistical features were created by applying different statistical functions to groups of related features.

- Packet_Length_Mean

- This is the average value of all packet length features across both forward and backward packets.

- Packet_Length_Median
  - This is the median value of all packet length features across both forward and backward packets

- Packet_Length_Std
  - This is the standard deviation of all packet length features across both forward and backward packets.

- Packet_Length_Range
  - This is the difference between the maximum and minimum values of all packet length features across both forward and backward packets.

1.6 ENCODING CATEGORICAL VARIABLES

Before building a machine learning model, it is often necessary to preprocess the data to ensure that it is in the right format for the model. In this case, two features require encoding: 'Protocol' and 'Malware Type'. Encoding is the process of converting categorical variables into numerical values, which can be easily processed by machine learning algorithms.

1. One Hot Encoding for 'Protocol':

   The 'Protocol' feature is a categorical variable with three unique values, representing the three different protocols used in the dataset. One Hot Encoding is a method used to convert a categorical variable into multiple binary columns, with each column representing a unique category (in this case, a unique protocol). Each row of the dataset will have a '1' in the column corresponding to the protocol it uses and a '0' in the other columns. This type of encoding is particularly useful when there is no ordinal relationship between the categories, meaning the categories cannot be sorted or ranked in a meaningful way. By using One Hot Encoding, the machine learning model can treat each protocol independently without assuming any inherent order.

2. Label Encoding for 'Malware Type':

   The 'Malware Type' feature represents the type of malware present in the dataset. Unlike the 'Protocol' feature, the number of malware types can be large, and there may be an implicit order or hierarchy between the types. In this case, Label Encoding is more appropriate. Label Encoding is a method where each unique category is assigned a numerical value, typically an integer starting from 0. For example, if there are five malware types, they will be assigned values from 0 to 4. While this encoding method is more compact and straightforward, it may introduce an ordinal relationship between categories, which might not be suitable for all use cases. However, for 'Malware Type', it can be beneficial, as the model can learn the relationships between different types of malware.

By encoding the 'Protocol' and 'Malware Type' features appropriately, the machine learning model can better understand and process the dataset, leading to improved performance in detecting Android Malwares.

1.7 RESAMPLING AND SCALING

Resampling and scaling techniques play a crucial role in building an accurate and robust machine learning model for detecting Android malware. In our project, we applied two techniques, Standard Scaler and Synthetic Minority Over-sampling Technique (SMOTE), to preprocess our dataset and address class imbalance.

Standard Scaler was used to normalize the data by transforming features to have a mean of 0 and a standard deviation of 1. This technique ensures that all features contribute equally to the model and reduces the influence of outliers. In our project, we applied Standard Scaler to our dataset before training our machine learning model.

In addition, we applied SMOTE, an oversampling technique, to address class imbalance. The imbalance between the number of benign and malicious apps in our dataset can negatively affect the performance of our model in detecting Android malware. SMOTE generates synthetic samples for the minority class by interpolating between similar observations in the dataset. This technique improves the model's performance on underrepresented data and increases the robustness of the model in real-world scenarios.

By combining Standard Scaler and SMOTE, we enhanced the effectiveness and generalization capabilities of our machine learning model. Normalizing the data with Standard Scaler reduces the impact of outliers and ensures that all features contribute equally to the model, while SMOTE addresses class imbalance and enhances the model's performance on underrepresented data. Overall, resampling and scaling techniques are essential preprocessing steps that help to improve the accuracy and robustness of our Android malware detection model.

1.8 FEATURE SELECTION

Feature selection is an important step in building a machine learning model that accurately detects Android malware. In our project, we used two different techniques, LASSO and Random Forest, to select the most relevant features and improve the performance and interpretability of our model.

LASSO, or Least Absolute Shrinkage and Selection Operator, is a linear model that has built-in feature selection capabilities. It works by imposing a penalty on the absolute size of the regression coefficients, effectively shrinking less important features towards zero. We used LASSO to identify the top features that contributed significantly to detecting Android malware.

In addition to LASSO, we also employed Random Forest, an ensemble-based method that estimates feature importance through numerous decision trees. Random Forest analyzes the impact of each feature by randomly selecting a subset of features at each node of the decision tree, then aggregating the results from all trees. This technique provided an additional perspective on the most relevant features in our dataset.

By comparing the results of both LASSO and Random Forest, we were able to identify the top 35 most important features. This comprehensive and robust selection of features helped enhance the performance and interpretability of our final model. We eliminated irrelevant or

redundant features, allowing our model to focus on the most important features for detecting Android malware. Overall, feature selection played a critical role in optimizing the performance of our machine learning model and in identifying the features most relevant to our project.

Table1. LASSO Implementation

Feature coefficients (LASSO)	
Protocol_1	0.231294899
min_seg_size_forward	-0.214956155
FIN_Flag_Count	-0.129146697
Packet_Length_Std	-0.125222328
Bwd_Packet_Length_Max	-0.105489865
Flow_Bytes_Packets_per_s_Ratio	0.104349297
URG_Flag_Count	-0.090510373
Flow_IAT_Mean	-0.078108522
Fwd_PSH_Flags	-0.054437468
Bwd_Packets/s	-0.052991505
Init_Win_bytes_forward	0.051045197
Source_Port	0.050368992
Fwd_Packet_Length_Min	0.048234878
Fwd_IAT_Std	-0.046310077
Fwd_Packets/s	0.041815924
Fwd_Packet_Length_Std	0.037874436
Min_Packet_Length	-0.035616776
Flow_IAT_Std	-0.034594528
Active_Max	0.031976843
Bwd_IAT_Min	-0.029706756
Bwd_Packet_Length_Min	0.022120766
Fwd_Bwd_IAT_Mean_Product	0.020945319
Init_Win_bytes_backward	0.020519736
SYN_Flag_Count	-0.015250855
Fwd_Bwd_Packet_Length_Mean_Product	0.015019541

Table2. Random Forest Implementation

Feature importances (Ramdom Forest)	
Source_Port	0.052672
Flow_Duration	0.040811
Flow_IAT_Max	0.039271
Flow_IAT_Min	0.039124
Flow_IAT_Mean	0.037537
Init_Win_bytes_forward	0.036855
Fwd_Packets/s	0.035989
Flow_Packets/s	0.035539
Fwd_IAT_Min	0.032314
Fwd_Bwd_IAT_Mean_Diff	0.027968

Fwd_IAT_Mean	0.026866
Fwd_IAT_Max	0.026513
Total_IAT	0.02642
Fwd_IAT_Total	0.026344
Destination_Port	0.0246
Bwd_Packets/s	0.023407
Init_Win_bytes_backward	0.018175
Total_Header_Length	0.015315
Flow_Bytes/s	0.015214
Flow_IAT_Std	0.015167
Fwd_IAT_Std	0.01416
Fwd_Bwd_Header_Length_Ratio	0.013273
Fwd_Header_Length	0.012566
Fwd_Bwd_Packets_per_s_Ratio	0.0109
min_seg_size_forward	0.010793
Avg_Fwd_Segment_Size	0.009571
Flow_Bytes_Packets_per_s_Ratio	0.009513
Average_Packet_Size	0.009378
Fwd_Bwd_Packet_Length_Mean_Ratio	0.008968
Fwd_Packet_Length_Mean	0.008965
Avg_Fwd_Bwd_Segment_Size_Ratio	0.008921

Table3. Selected Features

No.	Selected Features
1	Source_Port
2	Flow_Duration
3	Flow_IAT_Max
4	Flow_IAT_Min
5	Flow_IAT_Mean
6	Init_Win_bytes_forward
7	Fwd_Packets/s
8	Flow_Packets/s
9	Fwd_IAT_Min
10	Fwd_Bwd_IAT_Mean_Diff
11	Fwd_IAT_Mean
12	Fwd_IAT_Max
13	Total_IAT
14	Fwd_IAT_Total
15	Destination_Port
16	Bwd_Packets/s
17	Init_Win_bytes_backward
18	Total_Header_Length
19	Flow_Bytes/s
20	Flow_IAT_Std
21	Fwd_IAT_Std

22	Fwd_Bwd_Header_Length_Ratio
23	Fwd_Header_Length
24	Fwd_Bwd_Packets_per_s_Ratio
25	min_seg_size_forward
26	Avg_Fwd_Segment_Size
27	Flow_Bytes_Packets_per_s_Ratio
28	Average_Packet_Size
29	Fwd_Bwd_Packet_Length_Mean_Ratio
30	Fwd_Packet_Length_Mean
31	Avg_Fwd_Bwd_Segment_Size_Ratio
32	Subflow_Fwd_Bytes
33	Packet_Length_Std
34	Fwd_Bwd_Packet_Length_Mean_Product
35	Packet_Length_Mean'

1.9 MODELLING

1.9.1 BASE MODEL

In our initial attempt to build a machine learning model for detecting Android malware, we experimented with two tree-based classifiers, XGBoost and Random Forest. Our objective was to determine the importance of features in the dataset and identify which ones had the greatest impact on detecting malware.

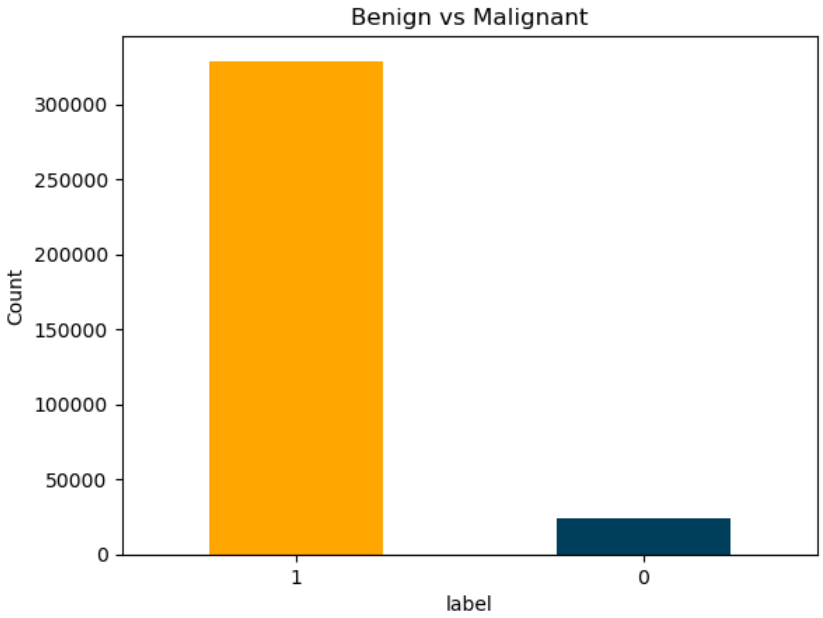
Unfortunately, our results were not satisfactory. We achieved a maximum accuracy of only 52% with Random Forest, and our initial attempts with XGBoost did not yield better results. We realized that using default feature selection methods was not sufficient, as it was the combination of various columns that actually detected a pattern in the data.

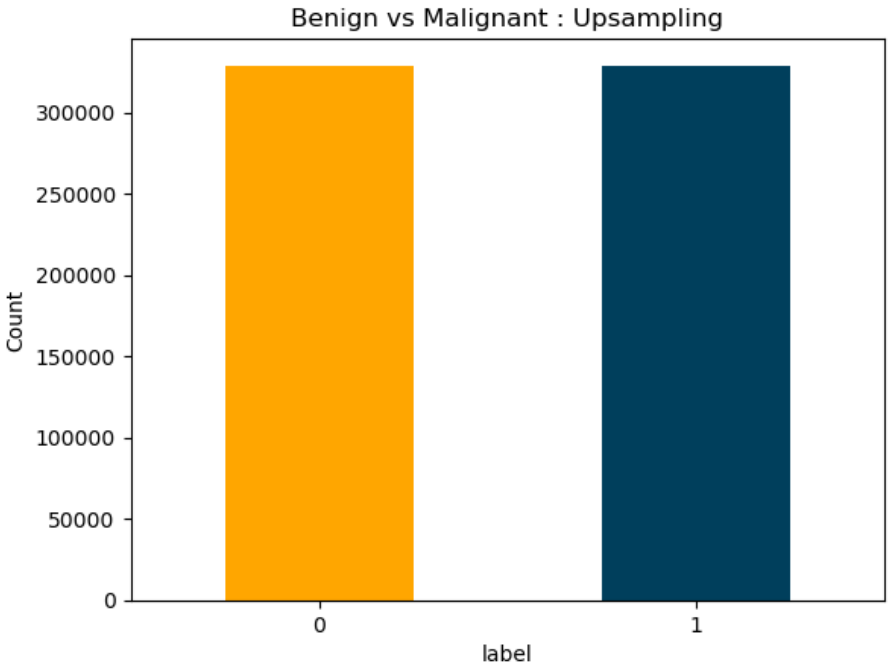
However, this initial attempt helped us realize the importance of feature engineering in building an accurate and robust machine learning model for detecting Android malware. We were able to learn from this experience and further refine our approach to improve the accuracy and effectiveness of our final model.

1.9.2 BINARY CLASSIFICATION MODEL

Predicting Benign vs Malignant

In order to accurately predict whether a given target is benign or malignant malware, we needed to ensure that our data was balanced with an equal number of entries for each class. Without balanced data, our model may become biased towards the class with more entries, resulting in inaccurate predictions. To address this issue, we used a technique called upsampling, which involves creating additional copies of the minority class (in this case, the malignant class) so that it has the same number of entries as the majority class (the benign class). This ensured that our model had an equal representation of both classes in the training data, allowing it to learn from a more diverse set of examples and potentially improve its performance.





We started by building a baseline model using Naive Bayes algorithm, which yielded an accuracy of only 50%. We then decided to explore other classification algorithms to improve the accuracy of our model. We evaluated four different classifiers, including Decision Tree, XGBoost, KNN and Neural Network. The results of our experiment are presented in the table below, which shows the accuracy of each algorithm. Notably, the Decision Tree algorithm demonstrated the highest accuracy with 94%, indicating that it outperformed our baseline model.

Model	Accuracy	Description
Naive bayes	50.05%	Base model
Decision tree	94.66%	Well Performed

XGBoost	68.36%	Well Performed
KNN	88.03%	Well Performed
Neural network	51.68%	Did not Perform Well-

1.9.3 MULTI-CLASS CLASSIFICATION MODEL

Predicting Types of Malware

To gain a better understanding of the features and determine the most effective feature engineering approach, we initially performed our modeling and analysis using a subset of the dataset, selecting only one CSV file from each malware family instead of using all the provided CSVs. This strategy allowed us to efficiently explore the relationships between features, evaluate different feature engineering techniques, and refine our models in a more manageable and focused manner, ultimately enhancing the performance of our final Android malware detection model.

Results of the sample dataset are:

Model	Accuracy	Description
Decision Tree	63.1%	Base Model
XGB Classifier	74.76%	Well Performed
KNN	50.31%	-
Neural Network	31.75%	-
Naive Bayes	24.01%	-
Random Forest	66.89%	Well Performed
Gradient Boosting	62.88%	Well Performed
AdaBoost	49.16%	-
CatBoost	70.76%	Well Performed
LightGBM	72.21%	Well Performed
SVM	39.4%	-
Logistic Regression	39.89%	-

During the course of our analysis, we encountered a challenge with the large volume of data that we had initially obtained. Specifically, we had 355,630 values, which proved to be too cumbersome to work with during multiclass classification modeling. The time taken to run each model was not feasible, and when we attempted to apply feature engineering, the time requirements were even more substantial. Additionally, we occasionally encountered "Out of Memory" errors, which further complicated the analysis process.

Therefore to streamline our analysis, we selected one family from each type of malware and worked exclusively with those values. This approach allowed us to effectively manage the size of the dataset while still enabling us to produce reliable results. By reducing the volume of data, we also mitigated the frequency of memory errors, which facilitated a more seamless analysis process.

We further applied only those classifiers that gave good result in our sampling case.

The results are:

Model	Accuracy	Description
Decision Tree	71%	Base Model
XGB Classifier	78.3%	Better Performance
LightGBM	75.15%	Good
Random Forest	81.77%	Best Performance
CatBoost Accuracy	76.69%	Good

1.10 CONCLUSION

In conclusion, our analysis highlights the importance of carefully selecting a representative sample dataset to effectively perform feature engineering and model refinement. By using a smaller sample, we were able to refine our models more efficiently and identify the top performing classifiers. This approach ultimately led to the development of a robust Android malware detection model with strong performance across all malware families.

1.11 REFERENCES

[CITATION] :

A. H. Lashkari, A. F. A. Kadir, L. Taheri and A. A. Ghorbani, "Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification," 2018 International Carnahan Conference on Security Technology (ICCST), Montreal, QC, Canada, 2018, pp. 1-7, doi: 10.1109/CCST.2018.8585560.